Reducing Registers in Convolution Operation for Binarized Neural Networks with Register-Bridge LSI Architecture

Jun Masuda Kazuhito Ito

Graduate School of Science and Engineering
Saitama University
Saitama 338-8570, Japan

Abstract— Convolutional neural networks are widely used to implement machine learning such as image recognition. BNNs, which binarize data and convolution weights, are advantageous in terms of reducing power consumption and miniaturizing implementations. In this paper, a method to reduce the number of registers required to store data and weights in LSI implementations of BNNs is proposed using the register bridge architecture. The number of register bits was reduced by 44% compared to using the conventional architecture.

I. Introduction

In recent years, the application of machine learning has been attracting attention. In particular, convolutional neural networks (CNNs) are widely used in the field of image recognition [1]. CNNs are composed of an input layer, convolution layers, pooling layers, fully connected layers, and an output layer. In the convolution layer, convolution calculations using input data and weight coefficients called kernel are repeatedly performed, which is actually a product-sum calculation. This convolution calculation can be processed in parallel on an LSI.

Typically, the input data and kernels required for CNN convolution calculations are large, making it difficult to store all of the data in the on-chip memory inside the LSI. Therefore, external memory must be provided, and calculations must be performed while communicating appropriately between the LSI and the external memory. In this case, communication between the LSI and the external memory causes significant delays and power consumption, so a calculation execution method that minimizes inter-chip communication is required [2].

It is known that in CNNs, inference accuracy does not much decrease even if the number of bits used to represent input data and kernels is reduced. CNNs that limit both input data and kernels to 1-bit binary values are called binarized NNs (BNNs). In BNNs, the product of input data and kernels can be calculated using a simple exclusive OR (XNOR) circuit rather than a multiplication circuit [3].

As architectures suitable for parallel calculations for LSIs, the regularly distributed register (RDR) architecture [4], in which processing elements (PEs) are regularly arranged, and the register bridge (RB) architecture [5], in which registers called bridge registers (BREGs) are provided between PEs, have been proposed. Compared to the RDR architecture, the RB architecture has the advantage of reducing data communication time and reducing the number of register bits by sharing data between PEs using BREGs.

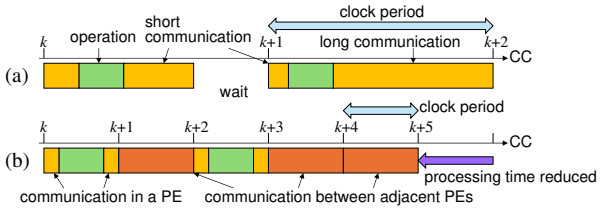


Fig. 1. Operation and communication time chart for LSI implementation. (a) random placement. (b) structured placement.

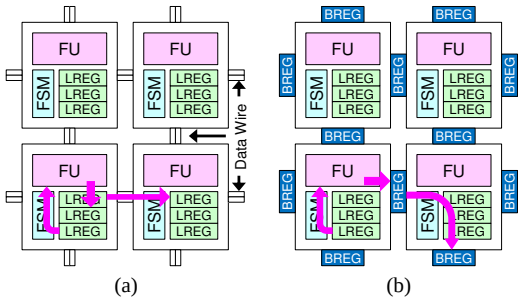


Fig. 2. Parallel LSI architectures. (a)RDR. (b)RB

In this paper, we propose an LSI implementation targeting the RB architecture of a BNN that takes communication with external memory into consideration. In particular, we propose a method that takes advantage of the feature of the RB architecture that allows data sharing between adjacent PEs to transmit input data and kernels to PEs with fewer register bits than the RDR architecture and execute convolution calculations in BNN in parallel.

The remainder of the paper is organized as follows. The RB architecture is briefly explained in Sect. 2. Consideration in hardware implementation of BNN is presented in Sect. 3. The proposed method is described in Sect. 4 and evaluated in Sect. 5. Section 6 concludes the work.

II. Register-Bridge Architecture

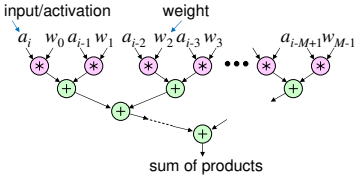
LSIs consist of functional units (FUs), registers, control circuits, and so on. When many FUs and registers are placed randomly on an LSI chip, long-distance communication on the chip is required as the LSI becomes larger, and the commu-

nication delay increases. In clock-synchronized processing, if long-distance communication is required within one clock period, the clock period must be extended to match the longest communication delay in order to execute the processing correctly as shown in Fig. 1(a). Even if the operation to generate a data and the operation to consume the data are placed close to each other on the chip and the required communication time is short, operations must be synchronized with a clock signal, resulting in unnecessary waiting time before the operation to consume the data can begin execution as shown in Fig. 1(a). Hence the processing time derived as the product of the clock period and the number of clock cycles (CCs) is increased.

The regular distributed register (RDR) architecture [4] shown in Fig. 2(a) has been proposed as a parallel processing architecture for LSI. In the RDR architecture, multiple PEs containing FUs, registers, and control circuits (FSM) are placed in a regular manner, and the operations in the given processing are executed in parallel by the multiple PEs. Communication within one CC is limited to between the FUs and registers in the same PE. Data reading from registers, communicating the data to FUs in the same PE, and communicating and storing the calculation results to registers in the same PE are performed within one CC. Registers in a PE are called local registers (LREGs). Communication between PEs is performed using a CC dedicated to communication that does not involve calculation on FUs, and communication within one CC is limited to adjacent PEs. A long distant communication is performed using multiple CCs by repeating communication between adjacent PEs. This shortens the communication distance within one CC, shortens the clock period, and reduces the processing time as shown in Fig. 1(b).

Instead of wiring between adjacent PEs in the RDR architecture, the register bridge (RB) architecture [5] is proposed, in which registers are placed between adjacent PEs as shown in Fig. 2(b) and the registers can be read and written from both adjacent PEs. The registers are called bridge registers (BREGs). Data reading from LREG or BREG, executing calculation on FUs in the same PE, and storing the calculation results to LREG or BREG are performed within one CC. There is no significant difference between the communication time between adjacent BREGs and FUs and between LREGs and FUs, and the shortest clock cycles of the RDR and RB architectures are considered to be equivalent. Communication between distant PEs is performed by repeating communication between BREGs using multiple CCs. The data to be communicated to another PE is stored in an appropriate BREG rather than LREG, hence the required number of CCs for communication is the same or less than the RDR architecture.

Another advantage of the RB architecture is that it is possible to share data between adjacent PEs using the BREG. When adjacent PEs use the same data, the RDR architecture requires each PE to store the data in the LREG in duplicate. On the other hand, in the RB architecture, since the PEs on both sides of the BREG can read data directly from the BREG, the number of register bits required to hold the data can be reduced by storing commonly used data in the BREG.



(a)

arithmetic			logic		
a	w	p	a	w	p
-1	-1	$+1$	0	0	1
-1	$+1$	-1	0	1	0
$+1$	-1	-1	1	0	0
$+1$	$+1$	$+1$	1	1	1

(b)

Fig. 3. Convolution and its implementation in BNNs. (a) convolution. (b) a 1-bit product.

```
for( no=0 ; no<OUT ; no++ )      ---------- Loop-4
  for( y=0 ; y<OH ; y++ )        ---------- Loop-3y   } generate 3D output
    for( x=0 ; x<OW ; x++ )      ---------- Loop-3x
      s(no,x,y) = 0
      for( ni=0 ; ni<IN ; ni++ )   ---------- Loop-2
        for( ky=0 ; ky<K ; ky++ )  -------- Loop-1y   } 3D sum of
          for( kx=0 ; kx<K ; kx++ )  ----- Loop-1x   }   products
            s(no,x,y) += ai(ni,x+kx,y+ky)*w(ni,no,kx,ky)
      ao(no,x,y) = f(s(no,x,y)+bias(no))
```

Fig. 4. Pseudocode for convolution layer calculation.

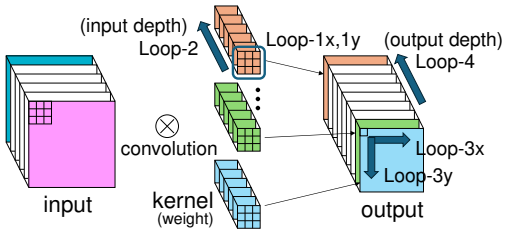


Fig. 5. Loops in convolution.

III. Hardware Implementation of BNNs

A. Convolution operation in BNNs

The convolution operation of kernel size M is illustrated in Fig. 3(a). M inputs and M weights in the fist layer or M activations and M weights in the second and later layers are multiplied one by one to get M products and these products are summed up. In BNN, the input/activation and weight take only two values, '$+1$' and '-1', and the combination of the values of input/activation a, weight w, and their arithmetic product p are shown in the left half of Fig. 3(b). When the logic values 1 and 0 are assigned to '$+1$' and '-1', respectively, the truth table of the multiplication is as shown in the right half of Fig. 3(b). This means that the 1-bit multiplication in BNNs can be performed by the inversion of the exclusive OR of a and w. That is, the 1-bit product can be obtained by an XNOR operation.

B. 4 level nested loops in the convolution

The computation in the convolution layer of CNN consists of four levels of nested loops, as shown in the pseudocode in Fig. 4. The four-level loops are illustrated in Fig. 5. In Fig. 5, the convolution calculation proceeds by sliding the kernel against the input and repeating the multiply-add calculation between the corresponding input region and the kernel. The roles of each of the four loops are as follows. Loop-1 (Loop-1x, Loop-1y) corresponds to the vertical and horizontal directions of the kernel, Loop-2 corresponds to the depth direction of the input, Loop-3 (Loop-3x, Loop-3y) corresponds to

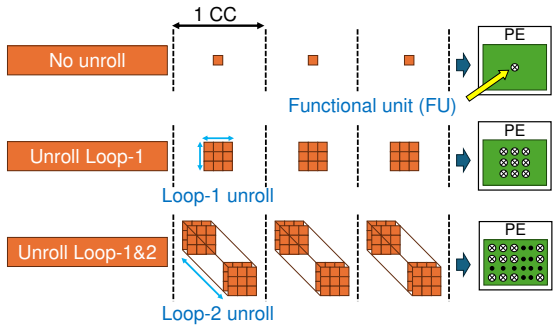


Fig. 6. Required FUs in PE against Loop-1 and Loop-2 unrolling.

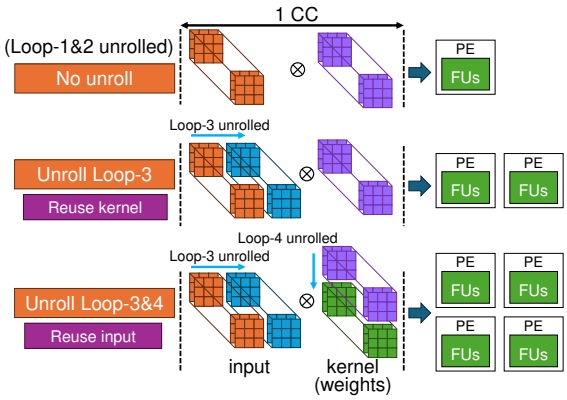


Fig. 7. Required PEs against Loop-3 and Loop-4 unrolling.

the vertical and horizontal directions of the output, and Loop-4 corresponds to the depth direction of the output. By unrolling these four levels of loops, the convolution calculation can be executed in parallel on an LSI. Unrolling a loop to perform *n* iterations simultaneously in parallel is defined as *n*-step unrolling of the loop, or it is denoted as the loop is *n*-step unrolled.

C. Loop unrolling for parallel computation

Figure 6 shows the input data and kernel area that are used in the calculations per clock cycle (CC) in a PE when Loop-1 and Loop-2 are unrolled. When both Loop-1 and Loop-2 are not unrolled, one piece of data is calculated per CC, as shown in the top part of Fig. 6. When Loop-1 is unrolled, the calculation area per CC increases in the vertical and horizontal directions of the kernel, as shown in the middle part of Fig. 6. When both Loop-1 and Loop-2 are unrolled, the calculation area increases not only in the vertical and horizontal directions of the kernel, but also in the depth direction of the kernel, as shown in the bottom part of Fig. 6. Here, the operations for unrolled Loop-1 and 2 are performed in parallel by preparing multiple FUs in a PE.

Figure 7 shows the input data and kernel area that are the subject of calculation per CC when Loop-1 and 2 are unrolled, and then Loop-3 and 4 are unrolled. The top of Fig. 7 corresponds to the case where only Loop-1 and 2 are unrolled as shown in the bottom of Fig. 6, and Loop-3 and 4 are not unrolled. When Loop-3 is unrolled, calculations are performed using the same kernel for different input data, as shown in the middle of Fig. 7. Here, Loop-3 is 2-step unrolled, and two PEs perform calculations in parallel using a common kernel for different input data. When both Loop-3 and 4 are unrolled, as shown in the bottom of Fig. 7, multiple kernels are used for the calculation using a common input. Here, Loop-3 and Loop-4 are each 2-step unrolled, and calculations are performed in parallel using four PEs. In the figure, two PEs arranged horizontally each perform calculations using the same input data, and two PEs arranged vertically each perform calculations using the same kernel.

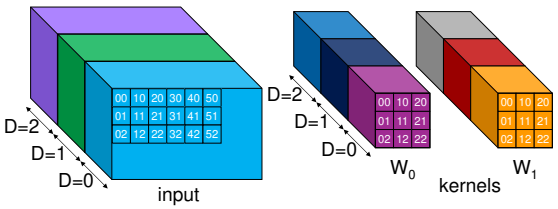


Fig. 8. An example of input and two kernels.

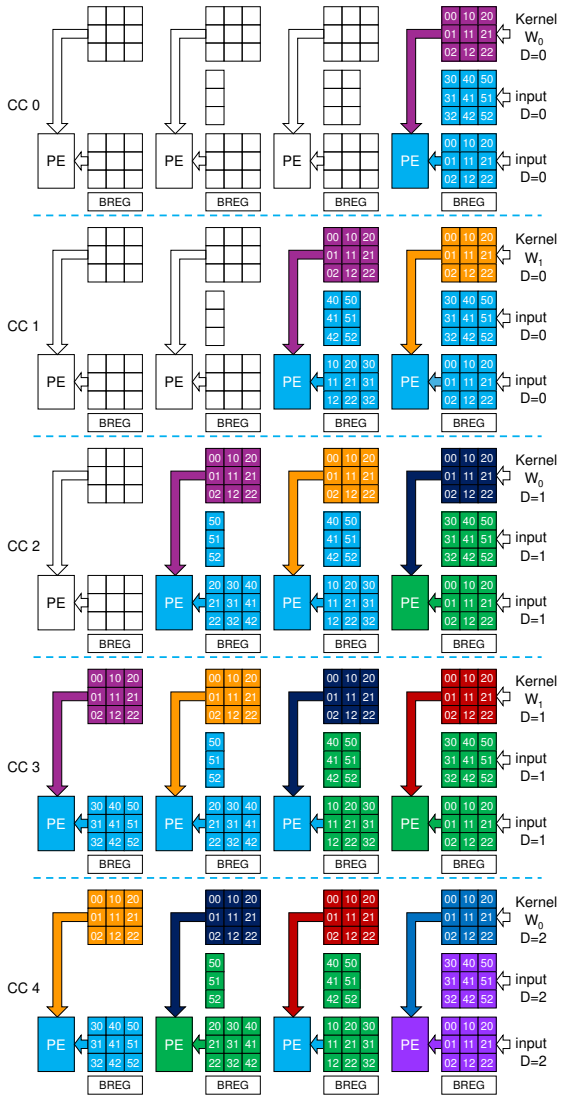


Fig. 9. Parallelized computation of convolution on RB architecture.

IV. The Proposed Method

The proposed method of performing convolution calculations in binarized CNN on an LSI of the RB architecture is described.

A. Unrolling the loops for convolution

In the proposed method, the loops for convolution are unrolled as follows. Loop-1 is completely unrolled. Loop-2 is partially unrolled, and each iteration of the partially unrolled Loop-2 is shown as D=0, D=1, and so on as shown in Fig. 8. Loop-3 is partially unrolled horizontally and vertically. Loop-4 is not unrolled, but iterations in Loop-4 are executed concurrently. By using the input data and two kernels W_0 and W_1 shown in Fig. 8 as an example, how the convolution is performed is explained as follows.

Figure 9 shows how the input data and the two kernels are stored in the BREG when convolution calculations are performed in parallel on the RB architecture. The input data and kernels are transferred on the BREG from right to left in Fig. 9 every time a clock cycle passes. Here, the vertical and horizontal sizes of the kernel are both 3, thus 6 columns of input data are placed in the BREG on the right end, and the required columns of input data are transferred to the left. The PE executes the multiply and accumulate (MAC) calculation when the input data and the kernel required for the calculation are available in the adjacent BREGs.

In Figure 9, the kernel given to the rightmost PE is W_0 for CC 0 and W_1 for CC 1. Using two consecutive CCs, input data corresponding to the same D value is repeatedly used to calculate two outputs corresponding to each of the two kernels. Furthermore, the input data and kernels given from BREG switch as D=0, D=1, ... every two CCs. In this way, Loop-2 of the MAC calculations proceed in each PE.

When the convolution calculation is performed using the method illustrated in Fig. 9, the number of bits of the register holding the intermediate result of MAC calculation, the on-chip buffer capacity, and the number of bits read from the external memory change depending on the combination of the execution methods of Loop-3 and Loop-4. Each item is considered below.

A.1. Register for intermediate result of multiplication and accumulation calculation

In the MAC calculation of input data and kernels, the calculation for two kernels is performed alternately, that is, for kernel W_0 and D=0, for kernel W_1 and D=0, W_0 and D=1, W_1 and D=1, and so on. In other words, the MAC calculation for W_0 and the MAC calculation for W_1 are performed concurrently using the same PE in a time-sharing manner. Therefore, it is necessary to store the intermediate MAC calculation results for each kernel in a separate register as shown in Fig. 10. The kernel size, i.e., the number of weights in one kernel, is N_k. In a binarized NN, the maximum value of the MAC calculation result is N_k, and the number of bits required for the register B is $\lceil \log_2(N_k+1) \rceil$. When calculating Q kernels concurrently, the total number of bits required for the registers to store the MAC calculation results is QB. This is required for each PE.

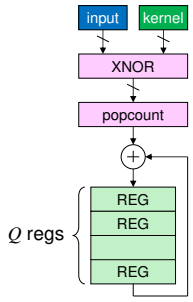


Fig. 10. FUs for MAC calculation and registers for the intermediate results in PE.

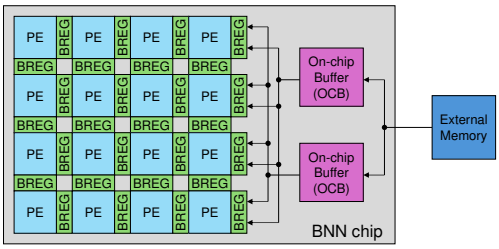


Fig. 11. On-chip buffer and external memory.

TABLE I
THE BITS OF REGISTERS, ON-CHIP BUFFER, AND MEMORY READ

# kernels Q	ACC regs [bit]	On-chip buf [bit]	Kernel Read [kbit]	Input Read [Mbit]
1	264	1152	73.73	33.55
2	528	2304	73.73	16.78
3	792	3456	73.73	11.18
4	1056	4608	73.73	8.39

A.2. The size of on-chip buffer

Input data and kernels are stored in external memory, and as shown in Fig. 11, data required for calculation is read from external memory and provided to the PE via inter-chip communication. Storing repeatedly used data in an on-chip buffer reduces the amount of data reads from external memory. In other words, the on-chip buffer acts as a cache.

Considering that the total capacity of on-chip buffers is generally limited, we consider a configuration in which only kernels are stored in on-chip buffers and input data is not buffered.

Figure 9 shows an example in which calculations are performed using two kernels, W_0 and W_1. In the next iteration of Loop-3, W_0 and W_1 are used again. Therefore, these kernels are stored in the on-chip buffer. If the kernel size is N_k and the number of kernels is Q, the number of bits of the on-chip buffer is QN_k.

A.3. The number of bits read from external memory

When Loop-4 is repeatedly executed in such a manner as performing convolution calculations for one kernel and then the next kernel, all bits of the input data are read from external memory for each iteration. Therefore, the same input data is read from external memory the same number of times as the number of iterations of Loop-4. When performing convolution calculations for two kernels concurrently as shown in Fig. 9,

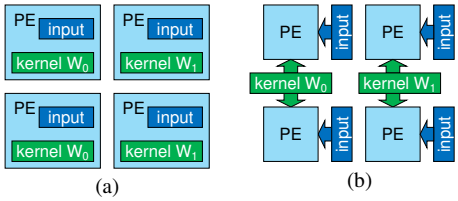


Fig. 12. Storing kernels in RDR architecture (a) and RB architecture (b).

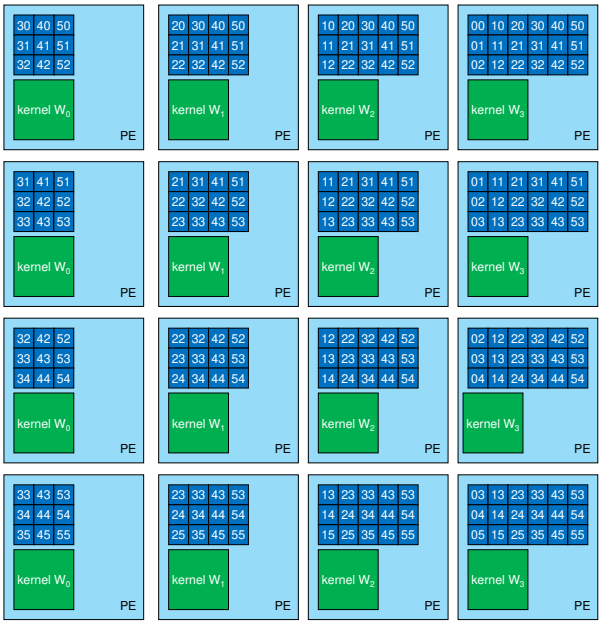


Fig. 13. An example of register usage in RDR architecture.

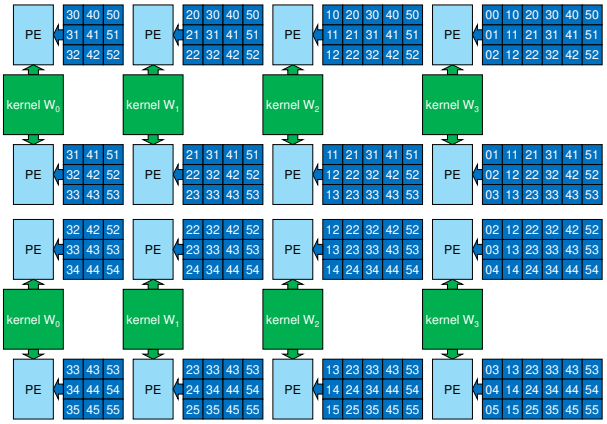


Fig. 14. An example of register usage in RB architecture.

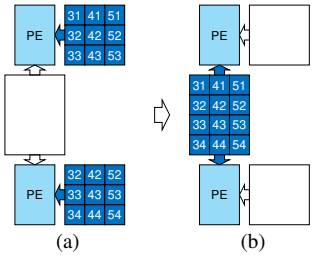


Fig. 15. BREG sharing for input data.

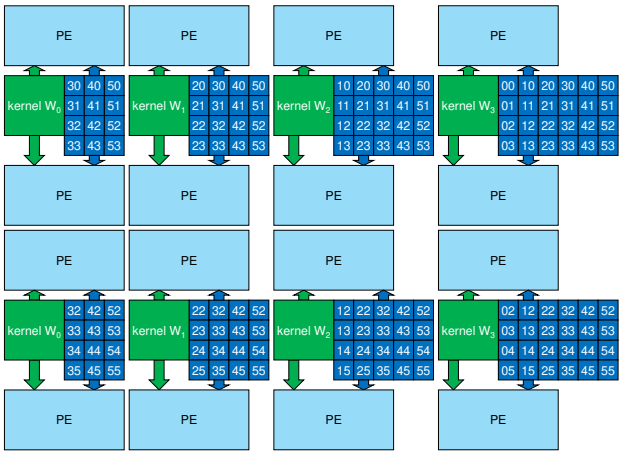


Fig. 16. An example of register usage in RB architecture with BREG sharing.

the input data is reused, so the number of times the input data is read can be halved. In general, when Q kernels are calculated in parallel, the number of times the input data is read can be reduced to one-Qth.

A.4. Summary

The number of kernels Q used in MAC calculations in parallel and the number of bits mentioned above are shown in Table I. Here, $N_k = 1152$ ($= 3 \times 3 \times 128$), the input data size is $64 \times 64 \times 128$, the number of iterations of Loop-4 (total number of kernels) is 64, and the number of PEs is 24. The number of bits of the registers to hold the MAC calculation intermediate result 'ACC regs', and the number of bits of the on-chip buffer are proportional to Q. By using the on-chip buffer, there is no need to read the same kernel from external memory multiple times, and the number of kernel read bits 'Kernel Read' is equal to the total number of bits of 64 kernels regardless of the value of Q. The number of input data read bits 'Input Read' is inversely proportional to the Q value.

B. Reduction of registers by utilizing BREG

B.1. Sharing kernel

Figure 12 shows the implementation of the RDR architecture and the RB architecture for a circuit that executes convolution calculations in parallel when Loop-3 is 2-step unrolled vertically and Loop-4 is 2-step unrolled. In Fig. 12, the calculation is performed using two kernels W_0 and W_1. In the RDR architecture, the input data and kernels required for the calculations performed by each PE for each CC must be stored independently in registers inside the PE. Therefore, even if the PEs use the same kernel for calculations as shown in Fig. 12(a), each PE needs a register to store the kernel. In the RB architecture, the kernels are stored in the BREGs and used by two vertically adjacent PEs as shown in Fig. 12(b). By unrolling Loop-3 vertically in an even number of steps, the number of register bits required to store the kernel can be reduced to half compared to the RDR architecture.

Loop-3 is X-step unrolled horizontally and Y-step unrolled vertically, where Y is an even number. In this case, the number

of PEs is XY. Loop-2 is d-step unrolled. If the kernel size is K both horizontally and vertically, the number of register bits required to hold a kernel is $K \times K \times d$. The total number of register bits required to hold all the necessary kernel is XYK^2d for the RDR architecture and $XYK^2d/2$ for the RB architecture.

B.2. Sharing input data

Consider the case where Loop-3 is 4-step unrolled both vertically and horizontally, and four iterations of Loop-4 (using four kernels W_0, W_1, W_2, and W_3) are executed concurrently. In this case, Figs. 13 and 14 show how data is stored in registers in the RDR architecture and the RB architecture, respectively. As mentioned in Sec. B.1, the number of registers that hold kernels in the RB architecture can be halved by using BREGs compared to the RDR architecture.

Here, we pay close attention to the PEs and BREGs for input data in the top two rows of Fig. 14. In Fig. 14, all input data is stored in independent BREGs, but there is some overlap in the input data required by vertically adjacent PEs. Therefore, by providing a BREG for input data between PEs as shown in Fig. 15, it becomes possible to hold shared data without overlap. By eliminating overlaps, the number of register bits for input data storage can be reduced. Figure 16 shows the final kernel and input data held in BREG in the RB architecture with reduced duplication of input data.

When the kernel size is horizontally K, the leftmost PE requires K columns of input data, the PE to the right requires $K+1$ columns, the PE to the right of that requires $K+2$ columns, and so on, as shown in Figs. 13 and 16. The total number of columns of the input data is $C = X(K+(X-1)/2)$ for the RDR and RB architectures.

In the RDR architecture, the number of rows of input data stored in the register is YK. In the RB architecture, $K+1$ rows of input data are stored in the BREG between vertically adjacent PEs, with the upper PE using the first K rows and the lower PE using the last K rows. Therefore, the number of rows of input data stored in the BREG is $Y(K+1)/2$.

The total number of register bits required to store input data is $CYKd$ for the RDR architecture and $CY(K+1)d/2$ for the RB architecture.

V. Evaluation

The number of registers required to hold the kernel and input data when Loop-3 is unrolled in various ways is compared between the implementations of RDR and RB architectures.

The number of PEs is 24, and Loop-3 is X-step unrolled horizontally and Y-step unrolled vertically. $XY = 24$ and Y are even numbers. Loop-1 is fully unrolled, and Loop-2 is $d = 6$-step unrolled. Figure 17 shows the number of registers when the kernel size is $K = 3$, and Fig. 18 shows the number of registers when $K = 9$. The RB architecture can reduce the number of registers by 33% when $K = 3$ and 44% when $K = 9$ compared to the RDR architecture.

The number of kernels processed concurrently affects the number of bits of the on-chip buffer, but does not affect the number of bits of the BREG.

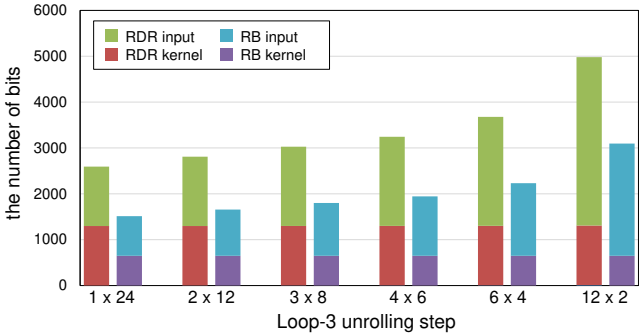


Fig. 17. The number of register bits for Loop-3 unrolling step of $X \times Y$ ($K = 3$).

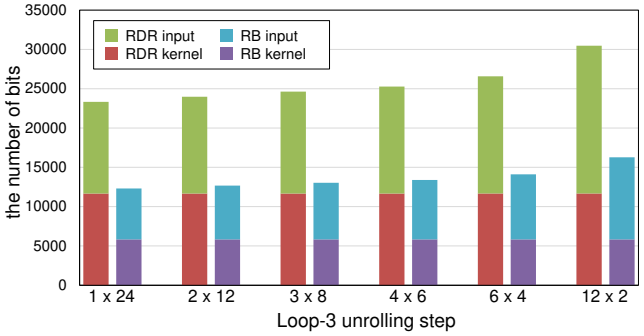


Fig. 18. The number of register bits for Loop-3 unrolling step of $X \times Y$ ($K = 9$).

VI. Conclusions

We proposed a method to reduce the number of register bits using the RB architecture in the LSI implementation of a binarized NN. It was confirmed that the number of register bits could be reduced by 33% when the kernel size $K = 3$ and by 44% when $K = 4$ compared to the RDR architecture. Future challenges include evaluating the number of bits when supplying input data to the PE array and reducing the number of bits read from external memory by using an on-chip buffer for input data.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 23K03969.

References

[1] Stanford University, “Convolutional neural networks for visual recognition.” https://cs231n.github.io/convolutional-networks/. Accessed: 2025-05-16.

[2] Y. Ma, Y. Cao, S. Vrudhula, and J.s. Seo, “Optimizing the convolution operation to accelerate deep neural networks on FPGA,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol.26, no.7, pp.1354–1367, 2018.

[3] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, “FINN: A framework for fast, scalable binarized neural network inference,” Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays, pp.65–74, 2017.

[4] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, “Architecture and synthesis for on-chip multicycle communication,” IEEE Trans. Computer-Aided Design Integrated Cirtuit. Syst., vol.23, no.4, pp.550–564, April 2004.

[5] T. Fujii, S. Nishizawa, and K. Ito, “Register-bridge architecture and its application to multiprocessor systems,” Proc. SASIMI, pp.10–15, 2016.