IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 4, 2020, accepted October 30, 2020, date of publication November 9, 2020, date of current version November 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3036882

FPGA Implementation of Hardware-Oriented Chaotic Boltzmann Machines

ICHIRO KAWASHIMA, (Student Member, IEEE), TAKASHI MORIE, (Member, IEEE), AND HAKARU TAMUKOH, (Member, IEEE)
Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Kitakyushu 808-0196, Japan
Corresponding author: Ichiro Kawashima (kawashima.ichiro172@mail.kyutech.jp)

This work was supported by the New Energy and Industrial Technology Development Organization (NEDO).

ABSTRACT Boltzmann machines (BMs) are useful in various applications but are limited by their requirement to generate random numbers. In contrast, chaotic Boltzmann machines (CBMs) are neural networks that imitate the stochastic behavior of BMs with the chaotic dynamics and deterministic behavior, without random numbers. CBMs can potentially require fewer hardware resources than the original algorithms due to the unnecessity of random number generators. In this study, hardware-oriented algorithms and a differential multiply-accumulation operation are proposed to overcome the difficulties of implementing CBMs on field-programmable gate arrays (FPGAs). A hardware-oriented algorithm for CBMs, which includes fixed-point operations and shift operations, is proposed to reduce hardware resource utilization in the implemented circuits. In particular, the differential multiply-accumulate operation allows us to implement the multiply-accumulate operation with block random access memory and digital signal processors to reduce the consumption of lookup tables and flip-flops in FPGAs without losing the calculation speed. Our proposed approach was evaluated in numerical simulations, logical synthesis, and FPGA implementation. The calculation speed of FPGA-implemented CBMs was compared with software-implemented CBMs, which resulted in 1 / 6,500 of calculation time reduction in a 300-neuron CBM. Moreover, 2,048 neurons of CBM were realized by the logical synthesis. Therefore, the proposed hardware implementation of CBMs was shown to be feasible. The proposed CBMs can solve combinatorial optimization problems at a larger scale with fewer resources.

INDEX TERMS Chaotic Boltzmann machines, combinatorial optimization problems, Ising model, field-programmable gate array.

I. INTRODUCTION

Boltzmann machines (BMs) [1]–[3] are fully connected neural networks used for numerous purposes, such as time-series data prediction and image classification [4]. BMs have applications in various fields, such as image recognition, sound recognition, and natural language processing [5]. In many studies, BMs exceeded conventional algorithms in image processing tasks (such as edge detection, caption generation, and object tracking). A remarkable feature of BMs is that the model can learn the relationships in data, which enables us to build applications that learn generative models of training data, such as automatic music generation [6], [7]. However, BMs have a high computational cost, and graphic processing units (GPUs) are frequently used to accelerate the calculation speed, albeit GPUs require much more power than CPUs. BMs can be run at high speed with less energy by implementing a dedicated circuit into hardware, such as a field-programmable gate array (FPGA) and a complementary metal-oxide-semiconductor (CMOS) chip. Despite the benefits of a hardware implementation of BMs, they consume a significant amount of hardware resources because random number generators must be included in the implemented hardware.

The Ising models [8] were known as an equivalent model of BMs. Ising models are used for solving combinatorial optimization problems, which are often recognized as NP-hard and require a tremendous amount of time to solve with Von Neumann architecture computers. Combinatorial optimization problem solvers are recognized as a type of quantum computations [9], [10], and Ising models are generally used for this purpose [11], [12]. Solving combinatorial

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Asif.

optimization problems is a critical requirement to solve problems in our society and artificial intelligence (AI). Many hardware implementations of Ising models for accelerating the calculations were reported [13], [14]. However, the necessity of random numbers in Ising models often deteriorates the efficiency of hardware implementations and BMs. To avoid this problem, implementation methods that do not require random numbers [15] and share random number generators [14] were proposed.

In this study, chaotic Boltzmann machines (CBMs) [16] are introduced to solve the problem of the requirement of random numbers in BMs and Ising models. CBMs are neural networks that imitate BMs' stochastic behavior by non-linear and chaotic dynamics. The deterministic behavior of CBMs can be implemented into hardware without random numbers, unlike in Ising models and BMs; therefore, the replacement of Ising models and BMs with CBMs enables us to implement applications with fewer hardware resources than the original models. The number of hardware circuits is reduced, owing to the removal of random number generators. This enables us to enlarge the scale of the network implemented in one FPGA or chip. Hence, larger-scale combinatorial optimization problems can be solved, as the resource utilization has been reduced. CBMs have been implemented in an analog chip, and their efficiency was reported [17], [18].

However, there are difficulties implementing the original algorithm of CBMs, such as the use of floating-point numbers and multiply-accumulate operation (the multiplication is conducted for a binary value and a decimal value in CBMs). In this study, hardware-oriented algorithms and a differential multiply-accumulate operation are proposed to solve those difficulties. These proposals reduce the hardware resource utilization of the implementation circuits of CBMs without a significant impact on accuracy and the calculation time. The proposals were verified with numerical simulations and evaluated by comparing with the conventional implementations. Our experimental results show that the hardware-oriented algorithms and the differential multiply-accumulation can reduce a remarkable amount of hardware resource consumption; as a result, a 2,048 node-fully-connected network was synthesized into a single FPGA. Additionally, the calculation speed of our hardware implementation was compared with software implementation. It was revealed that the hardware implementation accelerated the calculation speed at most 6,500 times as fast as software implementation. Finally, the comparison with related works revealed the ascendancy of this work on the hardware implementation scale; our hardware-implemented CBM has the most edges in the related works.

The remainder of this article is organized as follows. The algorithm of CBMs is explained in Section II. Our proposals (i.e., the hardware-oriented algorithms and the differential multiply-accumulation), designed CBM circuits, and our implemented system are described in Section III. The results of numerical simulations, logical synthesis, and the comparison of software and FPGA-implemented CBM are presented in Section IV. Additionally, an evaluation of our implementation based on hardware resource consumption and the calculation speed, and the comparison with related studies are described in Section V.

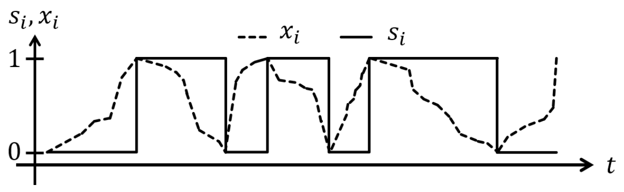


FIGURE 1. Temporal change of the internal state and the output of *i*th neuron.

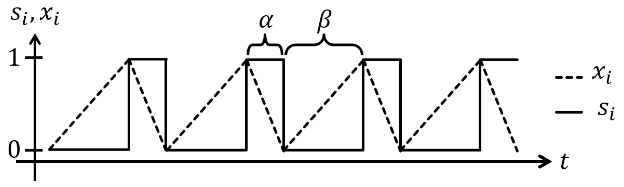


FIGURE 2. Temporal change of the internal state x_i and output s_i of *i*th neuron when the input value z_i is fixed.

II. CHAOTIC BOLTZMANN MACHINES

BMs are neural networks with stochastic behavior [1]. BMs are composed of neurons whose output changes with the following probability:

$$P[s_i = 1] = \frac{1}{1 + \exp \frac{-z_i}{T}}, \tag{1}$$

where $P[s_i = 1]$ refers to the probability that the output of ith neuron becomes 1. Moreover, $s_i \in \{0, 1\}$, z_i, and T refer to the output value, input value, and temperature of ith neuron, respectively. As shown in the equation, random number generators are required to produce the outputs of neurons stochastically when the BMs are implemented into hardware.

Chaotic Boltzmann machines (CBMs) are neural networks that behave deterministically by imitating the stochastic behavior of BMs by the chaotic dynamics [16]. Therefore, random number generators are required for each neuron when BMs are implemented into hardware, owing to their stochastic behavior. In contrast, CBMs do not require random number generators because of their deterministic calculations, which reduces the use of hardware resources compared to BMs. CBMs operate in continuous time and have an additional parameter for each neuron, called an internal state. The neuron's output is determined using this internal state. The behavior of the internal state is described by the following differential equation:

$$\frac{dx_i}{dt} = (1 - 2s_i)\left(1 + \exp \frac{(1 - 2s_i)z_i}{T}\right), \tag{2}$$

where $x_i \in [0, 1]$, $s_i \in \{0, 1\}$, z_i, and T refer to the internal state, output value, input value, and temperature of ith neuron, respectively. The output is calculated deterministically by the following equation:

$$\begin{cases} s_i \leftarrow 0 & (x_i = 0) \\ s_i \leftarrow 1 & (x_i = 1) \end{cases}, \tag{3}$$

where s_i refers to the state of the neuron. Fig. 1 illustrates the time change of the internal state and the output. As shown in the figure, the internal state continues oscillating between 0 and 1, and the change of the output occurs accordingly. The time change of the internal state and the output in the case the input value z_i is fixed, as shown in Fig. 2. In the figure, α and β are defined as time widths when the output is 1 and 0, respectively. By the incline of x_i described in (2), the time widths are described as follows:

$$\alpha = \frac{1}{\left|\frac{dx_i}{dt}\right|_{s_i=1}} = \frac{1}{1+\exp\frac{-z_i}{T}}, \tag{4}$$

$$\beta = \frac{1}{\left|\frac{dx_i}{dt}\right|_{s_i=0}} = \frac{1}{1+\exp\frac{z_i}{T}}. \tag{5}$$

The time width when the output is 1 in the whole calculation time of the CBM is calculated as

$$\frac{\alpha}{\alpha+\beta} = \frac{1}{1+\exp\frac{-z_i}{T}}, \tag{6}$$

which illustrates the ratio matches to the firing probability of BMs' neurons described in (1). For this reason, it is apparent that a CBM is a neural network model that expands the stochastic behavior of BMs into the time axis.

III. PROPOSED METHODS

A. HARDWARE-ORIENTED ALGORITHMS

Hardware-oriented algorithms of CBMs are proposed in this study. The algorithms are composed of a fixed-point operation and shift operation, which enables the FPGA-implemented CBMs to reduce hardware resource consumption.

Fixed-point numbers are used to replace floating-point numbers. The original algorithm of CBMs assumes the use of floating-point numbers to represent decimal values, which are expressed with a sign bit, significant bits, and exponent bits. There is a disadvantage that implemented circuits of floating-point numbers tend to be more complicated than the ones of fixed-point numbers. However, floating-point numbers can represent a wide range of numbers due to the exponential expression; mainly, the phenomena are critical in adder circuits. The operation is that the circuit needs to adjust the digits to two inputs before and after the addition. On the other hand, fixed-point numbers are expressed with integer bits and fraction bits, which do not use exponential expressions.

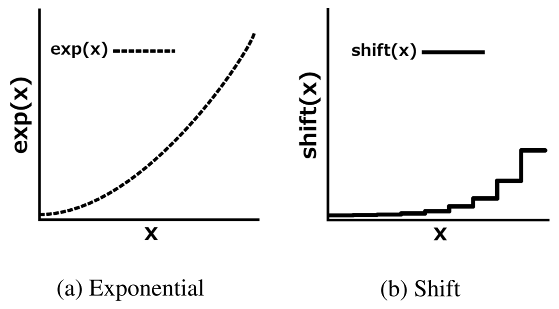


FIGURE 3. Exponential and shift operations.

Therefore, the replacement of floating-point numbers with fixed-point numbers enables implementation circuits to be simplified because of the identity for numerical calculation circuits for integer numbers. The simplification of digital calculation circuits of CBMs reduces a significant number of hardware resources for the implementation.

The exponential operation in (2) is replaced with the shift operation, as shown in Fig. 3a. To implement transcendental functions (e.g., sin, cos, and exp) into digital circuits, dedicated circuits for the approximation of those functions are frequently used, such as the table approximation and the coordinate rotation digital computer (CORDIC) [19]. Table approximation, which calculates functions by storing the output values in memory devices, often combined with linear interpolation or quadratic interpolation, consumes an enormous amount of memory resources. Additionally, CORDIC, which expresses the transcendental functions as a combination of addition, subtraction, and table reference, has a disadvantage in iterative operations. In contrast, the shift operation, shown in Fig. 3b, is represented as

$$\mathrm{shift}(x) = \begin{cases} 2^{\lfloor x \rfloor} & (x \geq 0) \\ 2^{\lceil x \rceil} & (x < 0) \end{cases} \tag{7}$$

and it can be implemented with a single barrel shifter circuit. Therefore, by replacing dedicated circuits for the calculation of exponentials, the consumption of hardware resources can be reduced. Moreover, the barrel shifter's calculation speed, which does not have iterative calculations, is faster than that of CORDIC.

The proposed hardware-oriented algorithm affects the computing ability of CBMs. The impact was confirmed by numerical simulations in this study.

B. DIFFERENTIAL MULTIPLY-ACCUMULATION

In this study, hardware resource consumption is reduced by time-division. Additionally, the differential multiply-accumulation is proposed to restrain the increasing calculation speed resulting from the time-division of the multiply-accumulation. The number of lookup tables (LUTs) and flip-flops (FFs) consumption, which increases in proportion to the number of neurons in a network, is reduced to a single digital signal processor (DSP) and a single random access memory (RAM) by the time-division of the calculation. In our proposal, the increased calculation time due to the time-division is reduced by the omitted calculation steps, with a focus on the difference between the former output and the present output of a network.

Multiply-accumulation is a common obstruction of most neural network implementations. In operation, the summation of multiplied values of outputs of neurons connected to a neuron, and synaptic weights between the neurons and other neurons, needs to be calculated. In CBMs' calculation, a neuron's output is represented by a binary value (i.e., 0 or 1); therefore, the multiplication can be implemented with a multiplexer instead of a multiplier circuit. However, the number of adder circuits (proportional to the number of neurons) is essential to calculate the multiply-accumulation owing to the fully-connected network topology. Hence, as the size of the network increases, the number of adder circuits expands in proportion to the square number of neurons. To overcome the problem, a method to reduce the number of adder circuits is proposed as the differential multiply-accumulation.

In the conventional multiply-accumulation, the input of *i*th neuron z_i in the network is described as

$$\begin{aligned} z_i &= b_i + \sum_{j \neq i}^{N} s_j w_{ij} \\ &= b_i + s_1 w_{i1} + s_2 w_{i2} \\ &\quad + \cdots + s_{i-1} w_{i(i-1)} + s_{i+1} w_{i(i+1)} \\ &\quad + \cdots + s_N w_{iN}, \end{aligned} \tag{8}$$

where N, b_i, s_j, and w_{ij}, respectively, represent the number of neurons in the neural network, the bias of *i*th neuron, the output of *j*th neuron, and the synaptic weight value w_{ij} between *i*th neuron and *j*th neuron. In the adder circuits of the multiply-accumulation circuit, adder circuits with two inputs are arranged into a tree structure, as shown in Fig. 4a. The circuit has a disadvantage that the number of adder circuits increases in proportion to the number of neurons. Fig. 4c shows a part of a network circuit implemented with the conventional multiply-accumulation. In this network circuit, neuron circuits are connected via synapse circuits, whose registers hold the synaptic weight. The synapse circuit outputs the weight value when the input value of the circuit is 1; otherwise, 0 is output. The number of synapse circuits and adders required to implement the network increases in proportion to the square number of neurons.

The time-division is introduced to replace the conventional multiply-accumulation. The synapse circuits and the adder circuits in Fig. 4c are replaced with a single block RAM and a single embedded DSP in FPGAs. In the time-division method, the input value of *i*th neuron is defined as

$$\begin{aligned} z_i &= A_i^N, \\ A_i^\tau &= \begin{cases} A_i^{\tau-1} + b_i & (\tau = i) \\ A_i^{\tau-1} + s_\tau w_{i\tau} & (\tau \neq i) \end{cases}, A_i^0 = 0, \end{aligned} \tag{9}$$

where z_i, $\tau \in [1, N]$, and A_i^τ represent the input value, iterator of the time-division, and accumulated value at time τ, respectively. The multiply-accumulation circuit with time-divided addition is constructed as shown in Fig. 4b. The addition circuit holds the accumulated value with a register and adds the input value into the accumulated value sequentially. Fig. 4d shows a part of the network circuit implemented with the time-divided multiply-accumulation. In the circuit, the iteration of the time-division is implemented as two multiplexers whose select signal is τ. Next, the output of neuron s_τ and the corresponding synaptic weight $w_{i\tau}$ are input to the addition circuit one by one.

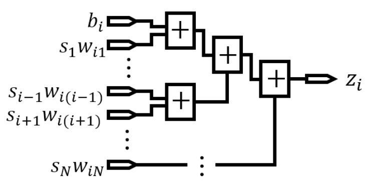
The behavior of the network circuit implemented for our proposal is shown in Fig. 5. The network's weight matrix is expressed by an array of RAMs in the left side of the figure. Notably, the diagonal elements of the weight matrix are used to store biases of neurons, to reduce extra resources to store the biases, and this architecture is reflected in (9). As shown in the figure, multiplied values of outputs in the networks and synaptic weights are accumulated as τ iterates from 1 to N. However, the time-division of multiply-accumulation increases the calculation time in proportion to the number of neurons.

The differential multiply-accumulation is proposed to overcome the problem of increased calculation time. In the neuron circuit with the conventional multiply-accumulation, as shown in Fig. 6a, the output value of neuron circuits, s_i, and synaptic weight, w_{ij}, circuits are input to the neuron circuits one by one. Therefore, these iterations in calculations increase processing time. On the other hand, the proposed differential multiply-accumulation improves the calculation speed, as shown in Fig. 6b. In the circuit, the iteration covers the neurons whose output changed. The former input of a neuron z_i^{t-1} is updated as the current input z_i^t with the difference between the former output of the network $\boldsymbol{s}_{t-1}$ and the present one $\boldsymbol{s}_t$. The calculation is described by the following equations:

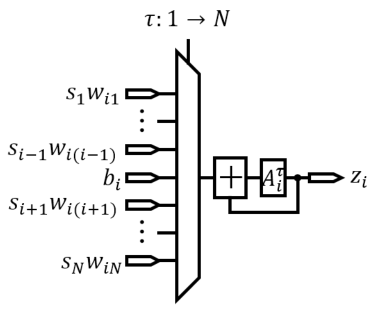
$$\begin{aligned} z_i^0 &= b_i + \sum_{j \neq i}^{N} s_j^0 \, w_{ij}, \\ z_i^t &= z_i^{t-1} + \sum_{\substack{j' \neq i \\ s_{j'}^{t-1} \neq s_{j'}^t}}^{N} d_{j'}^t w_{ij'}, \end{aligned} \tag{10}$$

$$d_{j'}^t = \begin{cases} -1 & (s_{j'}^{t-1} = 1, s_{j'}^t = 0) \\ 1 & (s_{j'}^{t-1} = 0, s_{j'}^t = 1) \end{cases}, \tag{11}$$

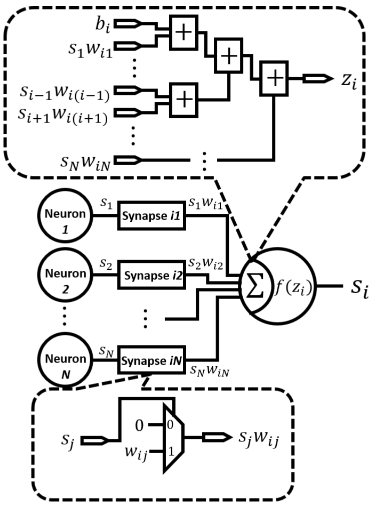
where t, z_i^t, and $d_{j'}^t$ represent time, the input of *i*th neuron at time t, and the difference between the former output of *j*th neuron s_j^{t-1} and the present output for s_j^t, respectively. The equation indicates that the first multiply-accumulation is the same as in the original calculation (8). However, iterator j is replaced with j', which only covers the neurons whose output has changed. The difference value $d_{j'}^t$ replaces $s_{j'}^t$ to update the present input value z_i^t. The replacement of iterator j with j' reduces the number of iterations significantly because every neuron changes its output twice in a period, as shown



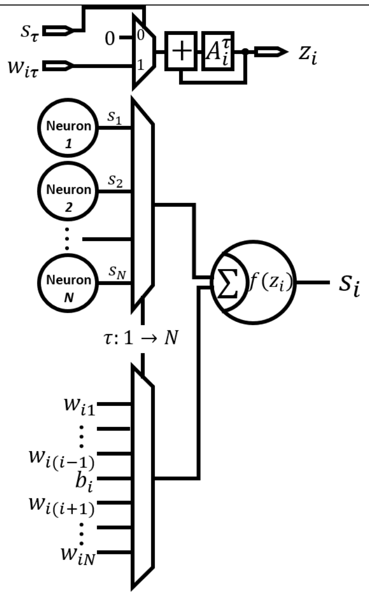
(a) Summation circuit implemented by the conventional addition



(b) Summation circuit implemented by the time-division addition



(c) Multiply-accumulation circuit implemented by the conventional addition



(d) Multiply-accumulation circuit implemented by the time-division addition

FIGURE 4. Designed summation and multiply-accumulation circuits.

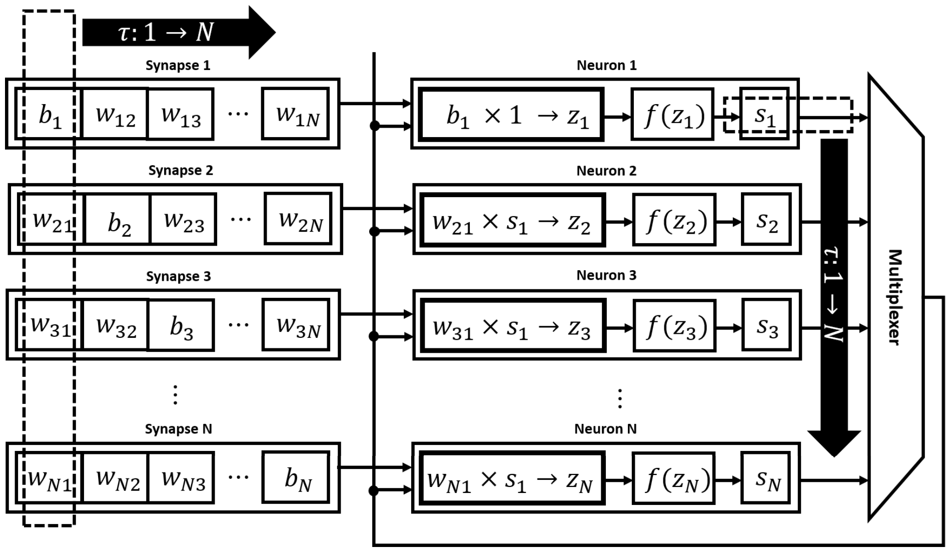

$\tau: 1 \rightarrow N$
Synapse 1
Neuron 1
$b_1 \times 1 \rightarrow z_1$
Synapse 2
Neuron 2
$w_{21} \times s_1 \rightarrow z_2$
Synapse 3
Neuron 3
$w_{31} \times s_1 \rightarrow z_3$
Synapse N
Neuron N
$w_{N1} \times s_1 \rightarrow z_N$
Multiplexer

FIGURE 5. Network behavior designed using the proposed method.

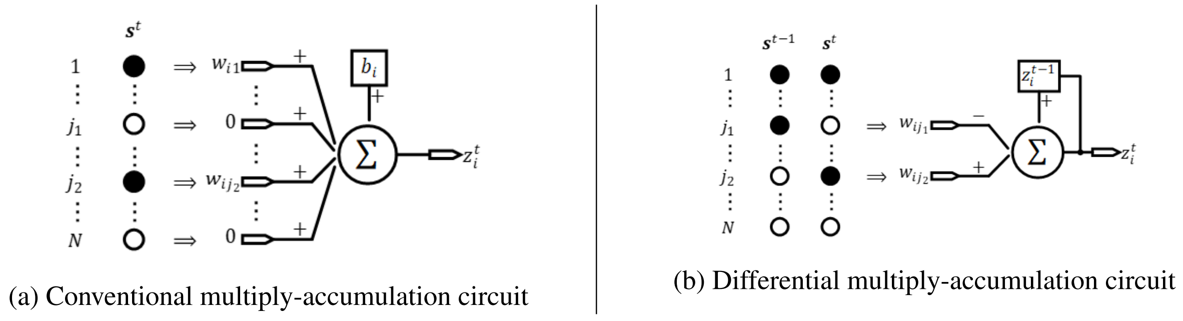


FIGURE 6. Conventional and differential multiply-accumulation.

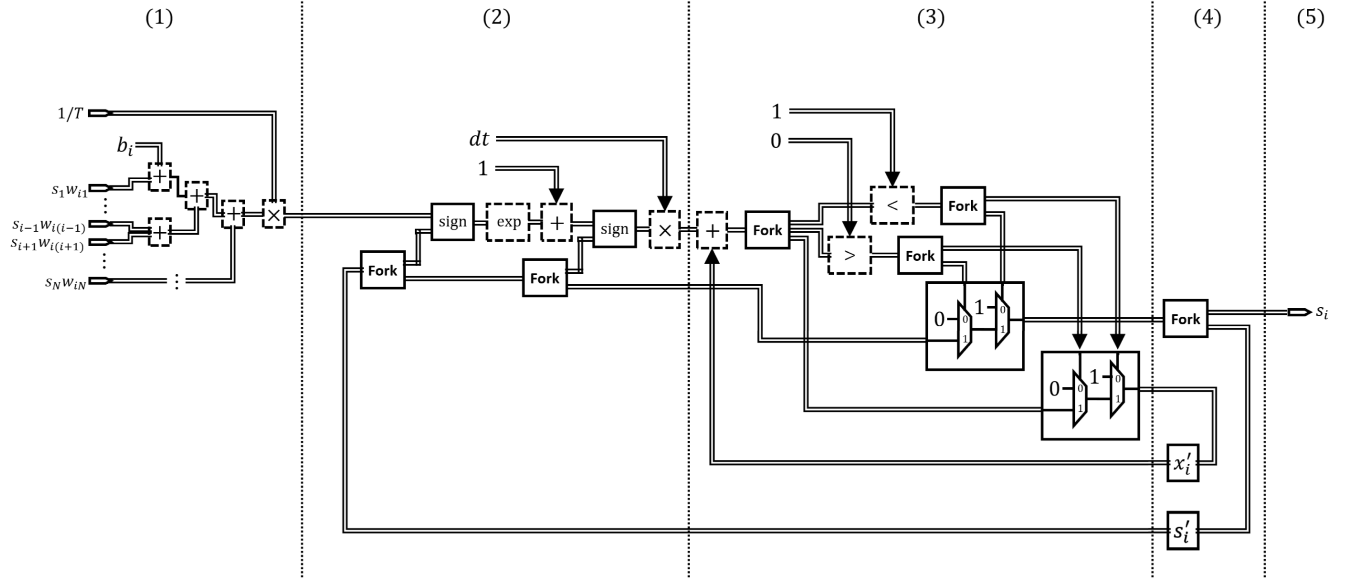


FIGURE 7. Neuron circuit designed by using the conventional algorithm (Numbers represent the calculation steps).

in Fig. 2. The low-frequent output change of neurons of CBMs increases the efficiency of the differential multiply-accumulation.

C. DESIGNED CIRCUITS

FPGA circuits are designed to compare hardware resource consumptions of the conventional method and our proposal. First, neuron circuits that use both the conventional and our hardware-oriented algorithms are designed to evaluate our proposed approach. Second, a conventional network circuit and a network circuit that uses our differential multiply-accumulation are designed to compare hardware resource usage. Note that both the conventional and the proposed network circuit have our hardware-oriented algorithm in their neuron circuits.

A neuron circuit designed using the conventional method (i.e., floating-point numbers and exponential operations) is shown in Fig. 7. IP cores produced by Xilinx are used as floating-point operation circuits, which include an exponential calculator, adders, multipliers, and comparators, and are represented by the broken lines in the figure. Double lines in the figure show handshake-protocol bus lines (i.e., AXI Stream), which are used to connect Xilinx IP cores. The fork module is used for distributing data and is required to arbitrate control signals. The former internal state and the former output of ith neuron are represented as x_i' and s_i', respectively. Moreover, x_i and s_i refer to the current internal state and the current output, respectively. Operation sign changes the sign of input of a neuron depending on the output of the neuron as

$$\mathrm{sign}(x) = (1 - 2s_i) \times x. \tag{12}$$

The calculation steps of the neuron circuit are as follows:

1) Calculate z_i/T by the multiply-accumulation and multiplying the inverted temperature $1/T$.
2) Calculate dx_i by multiplying the right-hand side of (2) and dt.
3) Calculate $x_i' + dx_i$ by adding dx_i to x_i' and calculating x_i and s_i as
   - 0 if $x_i' + dx_i \leq 0$,
   - 1 if $x_i' + dx_i \geq 1$.

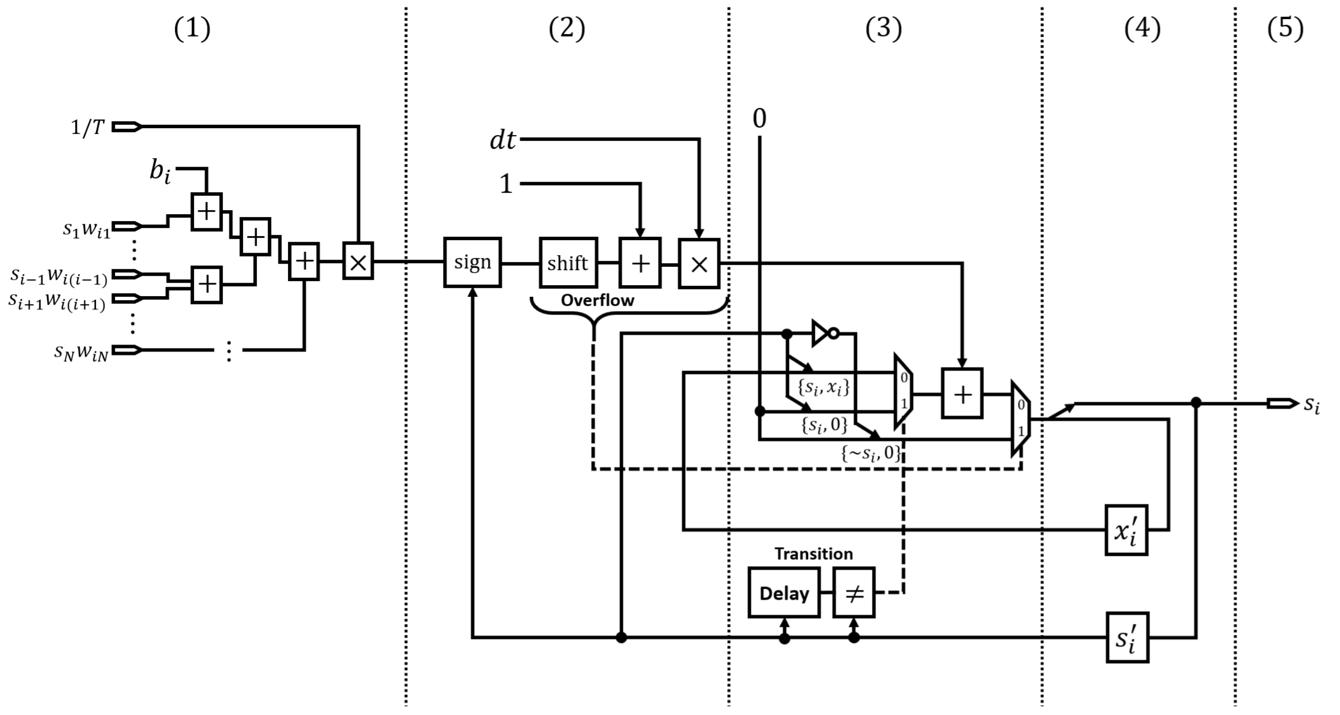
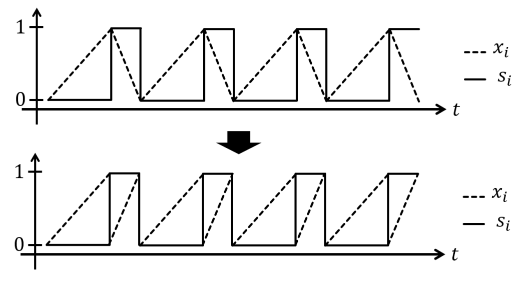
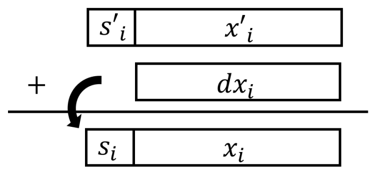


FIGURE 8. Neuron circuit designed by using the hardware-oriented algorithm (Numbers represent the calculation steps).



(a) Change of the update method of the internal state of the neuron



(b) Updating method of the internal state and the output of the neuron

FIGURE 9. Updating method of the proposed neuron circuit.

Clip x_i to a value between 0 and 1.

4) Update x_i' and s_i' with x_i and s_i to use them in the next calculation of the neuron.
5) Output s_i.

A neuron circuit is designed using our hardware-oriented algorithm (i.e., fixed-point operation and shift operation), and its structure is shown in Fig. 8. In the circuit, four types of fixed-point operation circuits are included: a shift operator, adders, multipliers, and comparators. Dashed lines' input to two multiplexers represents the overflow signal of fixed-point operation and the transition signal of the neuron's output. The former output and the former internal state of the ith neuron are denoted as x_i' and s_i', respectively. Moreover, x_i and s_i refer to the current internal state and the current output, respectively. In the designed circuit, the CBM algorithm is modified as shown in Fig. 9a. Next, s_i can be updated by the carry bit of x_i when the bits of s_i and x_i are connected and dx_i are added, as shown in Fig. 9b. The calculation steps of the neuron circuit are as follows:

1) Calculate z_i/T by the multiply-accumulation and multiplying the inverted temperature $1/T$.
2) Calculate dx_i by multiplying the right-hand side of (2) and dt, monitoring overflows.
3) Calculate $\{s_i, x_i\}$ as
   - $\{\backsim s_i', 0\}$, if the overflow has occurred in this step (the output needs to be changed immediately, not to delay the timing of the output change);
   - $\{s_i', 0\} + dx_i$, if the transition has occurred (this condition means that output s_i was changed in the update, and x_i needs to be reset to 0).
   - $\{s_i', x_i'\} + dx_i$, if the above conditions are not matched (the neuron needs to keep its internal state x_i and output s_i).
4) Update x_i' and s_i' with x_i and s_i to use them in the next calculation of the neuron.
5) Output s_i.

Here, $\{a, b\}$ means the bit connection of bit vectors a and b. In addition, $\backsim a$ refers the bit complementation of a bit a.

The network circuit designed using the conventional multiply-accumulation has the structure shown in Fig. 10a. This circuit is composed of N neuron circuits and $N(N-1)/2$ synapse circuits, and all neuron circuits are interconnected through synapse circuits. A synapse circuit that connects two neurons is composed of registers that store a synaptic weight and two multiplexers, which express multiplications $s_i w_i$ and $s_j w_i$, as shown in Fig. 10b. A neuron circuit is shown in Fig. 10c; the neuron circuit included adders to conduct multiply-accumulation and a multiplier beside a calculation circuit of dx_i/dt. The synapse circuit and the multiply-accumulation of the neuron circuit are implemented with LUTs and registers.

The network circuit designed using the proposed method is configured as shown in Fig. 11a. This circuit consists of N neuron circuits, N synapse circuits, and a network controller circuit. A synapse circuit that stores synaptic weights for a neuron can be represented by a single block RAM of FPGAs, shown in Fig. 11b. Its input is τ in Fig. 4d, and its output is a bias or synaptic weight. The multiply-accumulation in the neuron circuit of the proposed network circuit is expressed as a DSP block of FPGAs, as shown in Fig. 11c. A DSP block of Xilinx's products, such as DSP48E, has a multiplexer and an arithmetic and logic unit (ALU) that can be used as an accumulator. Inputs of the neuron circuit are the bias or synaptic weight, the inverted temperature, and control signals. The control signals are generated in the network controller circuit using the value of τ and the number of iterations, calculated as follows:

initialize
: Put 0 to the accumulated value by (9).

enable
: Put 0 as the input value when the bias is input after the second iteration because the bias value changes at the first iteration by (10).

negate
: Control whether the synaptic weight is added or subtracted by (11).

D. IMPLEMENTED SYSTEM

An FPGA-implemented CBM is controlled by software in the host PC in this system, and an FPGA and the host PC are connected by a PCIe interface. A Xillybus [20] IP core is used for the communication between the PC and the FPGA. Synaptic weights, biases, initial outputs of neurons, parameters (such as the frequency of annealing, initial temperature, and number of iterations), and control signals are input from the host PC. The FPGA outputs the output of the network and its energy and the status of the circuit. As shown in Fig. 12, the FPGA-implemented CBM is composed of a network circuit and the following circuits:

scheduler
: Decide the timing of annealing.

controller
: Control the calculation in the network.

temperature generator
: Calculate temperature values for the network.

In the network circuit, all neuron circuits and synapse circuits are connected to the host PC via multiplexers and demultiplexers and are accessed directly by the PC.

TABLE 1. Implementation Environment of Numerical Simulation.

environment	.NET Core 2.0.3
OS	Debian GNU/Linux (64-bit)
CPU	Intel Core i7 (4790k)
RAM	16 GB

TABLE 2. Experimental Conditions of Numerical Simulation.

initial temperature	$500,000$
attenuation factor	0.95
dt	2^{-12}

TABLE 3. Maximum Cutting Problems in Biq Mac Library.

# of nodes	Problem	# of nodes	Problem
100	ising2.5-100_5555	150	ising2.5-150_5555
	ising2.5-100_6666		ising2.5-150_6666
	ising2.5-100_7777		ising2.5-150_7777
	ising3.0-100_5555		ising3.0-150_5555
	ising3.0-100_6666		ising3.0-150_6666
	ising3.0-100_7777		ising3.0-150_7777
200	ising2.5-200_5555	250	ising2.5-250_5555
	ising2.5-200_6666		ising2.5-250_6666
	ising2.5-200_7777		ising2.5-250_7777
	ising3.0-200_5555		ising3.0-250_5555
	ising3.0-200_6666		ising3.0-250_6666
	ising3.0-200_7777		ising3.0-250_7777
300	ising2.5-300_5555		
	ising2.5-300_6666		
	ising2.5-300_7777		
	ising3.0-300_5555		
	ising3.0-300_6666		
	ising3.0-300_7777		

IV. RESULTS

A. NUMERICAL SIMULATION

Numerical simulations were conducted to evaluate the hardware-oriented algorithm. For the simulations, conventional CBMs (which have algorithms with exponential operations and floating-point numbers) and the proposed CBMs (which are implemented with shift operations and fixed-point numbers) are implemented as software in the environment shown in Table 1. The Euler method was used to update the internal states and outputs of the neurons of CBMs. Subsequently, the parameters shown in Table 2 were used, and the annealing was conducted every $N/128$ of time in the Euler method because CBM is a neural network that runs in continuous time, as illustrated in (2).

The maximum-cut problem can be solved by CBMs whose synaptic weights and biases are set as follows:

$$b_i = \sum d_{ij} \tag{13}$$

$$w_{ij} = -2d_{ij}, \tag{14}$$

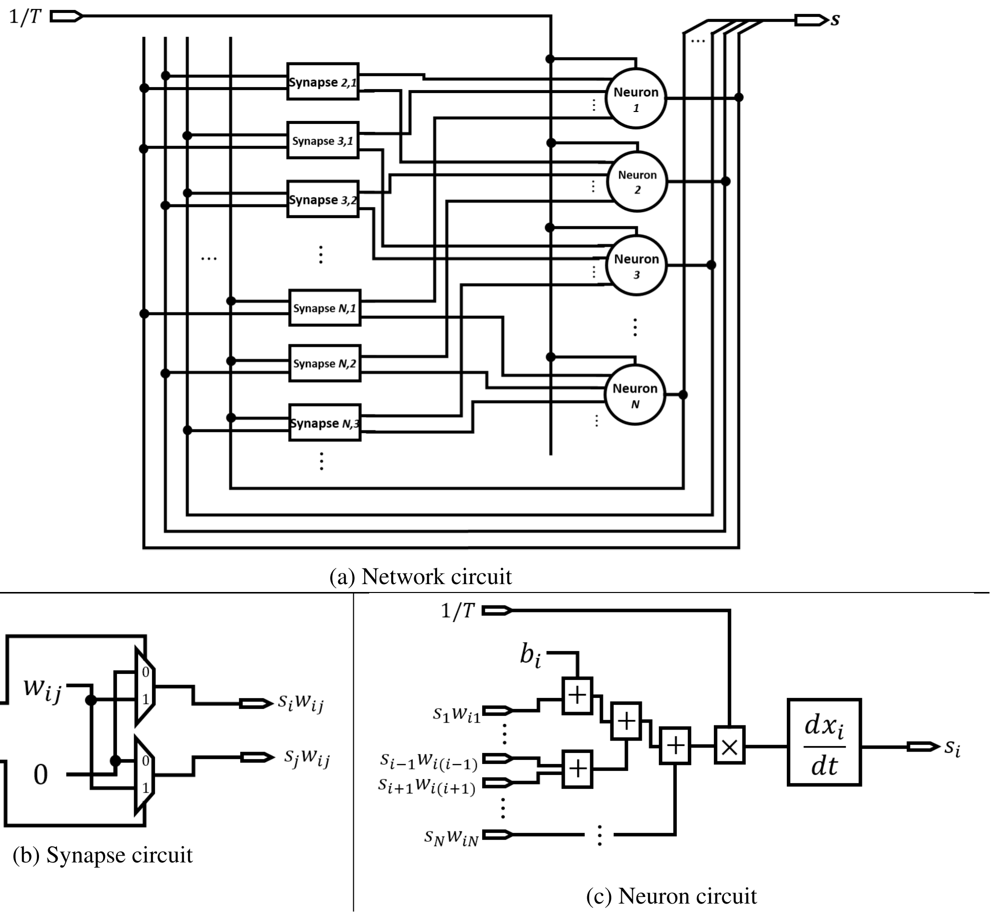


FIGURE 10. Conventional network circuit.

where w_{ij}, b_i, and d_{ij} refer to the weight value between ith and jth neurons, the bias value of ith neuron of the CBMs, and the edge weight between ith node and jth node of the problems, respectively [16]. Temperature, T, is decreased gradually from the initial value to a sufficiently low value when running CBMs. Solutions of maximum-cut problems are obtained as the combination of outputs of neurons in a CBM network when T becomes sufficiently low.

The computing ability of CBMs implemented with the proposed method has been measured with the maximum cut problem. Synaptic weights and biases were set according to (13). CBMs were run while the temperature decreases from the initial value to a sufficiently low value. Error rates were calculated as follows:

$$Error = 1 - \frac{E_{min}}{E_{opt}}, \tag{15}$$

where E_{min} and E_{opt} are defined as the minimum energy value acquired from CBMs and the optimized energy value of problems, respectively. In the numerical simulation, CBMs were applied ten times for each of the six different max-cut problems in Biq Mac Library [21], shown in Table 3, which has 100, 150, 200, 250, and 300 nodes with different initial values. The average and minimum error rates for each problem size were calculated.

First, two kinds of CBMs that use 64 bits of floating-point numbers and 24 bits of fixed-point numbers were applied to the max-cut problems for verifying the computing ability of CBMs implemented with fixed-point numbers. Fig. 13a and 13b show the results. A significant difference was not identified between the use of floating-point numbers and fixed-point numbers for any number of nodes of problems, although some variations depended on the number of nodes.

Second, the computing ability of CBMs implemented with the shift operation was verified by applying CBMs implemented with the exponential operation and the shift operation to the max-cut problems. Figs. 14a and 14b show the result. These figures indicate that the computational ability of CBMs implemented with the shift operation is slightly inferior to CBMs implemented with the exponential operation; however, the difference in computational ability is only a few points. Consequently, the replacement of the exponential operation with the shift operation does not significantly impact the computational ability of CBMs.

Finally, CBMs whose bit widths of fixed-point numbers are 8, 12, 16, 20, and 24 were applied to the max-cut problems to evaluate the correlation between the computational ability

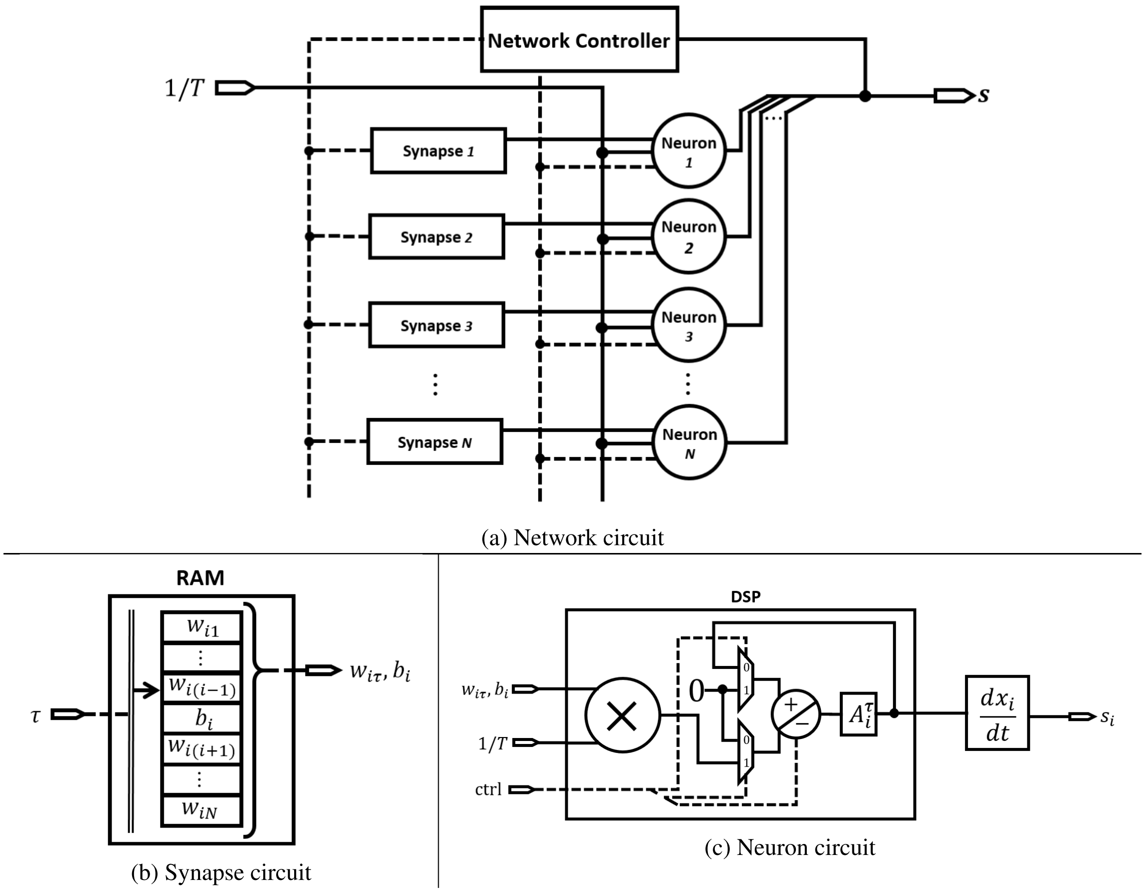


FIGURE 11. Proposed network circuit.

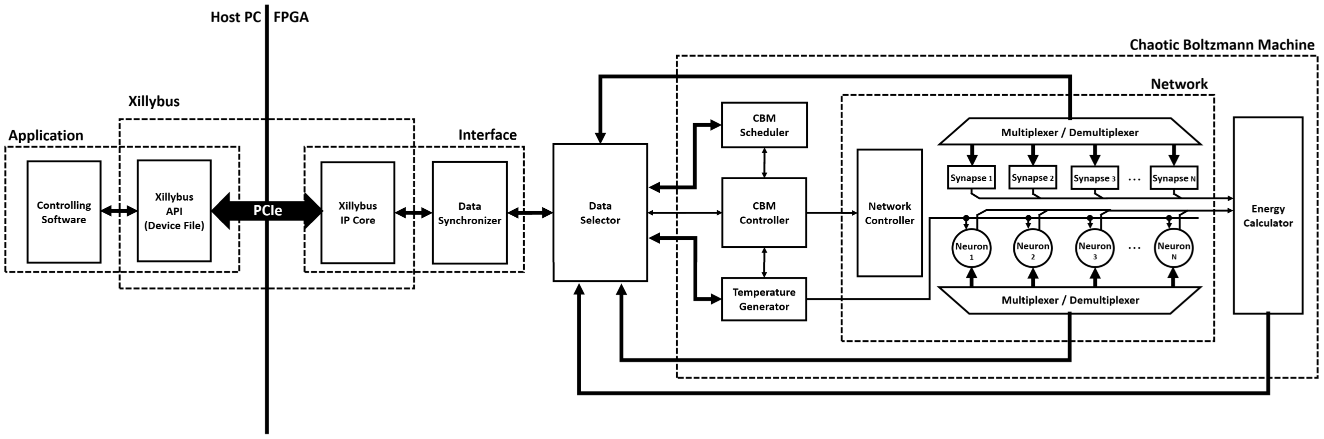


FIGURE 12. Implemented system.

of CBMs and the bit width of fixed-point numbers. In the experiment, $2^{\text{bit width of fixed-point numbers}/2}$ is used as the step size of the time of the Euler method because the stem size is limited by the bit width of the fixed-point numbers. Figs. 15a and 15b illustrate the results, which demonstrate that the error rate increases as the bit width decreases. The results indicate that the error rate could be maintained at approximately 10% if the bit width of fixed-point numbers is more than 16 bits.

The results of the numerical simulation revealed that the hardware-oriented algorithm impacted the computing ability of CBMs. However, a few points of difference in the error rate between the original algorithm and the hardware-oriented algorithm remained. The sufficiently

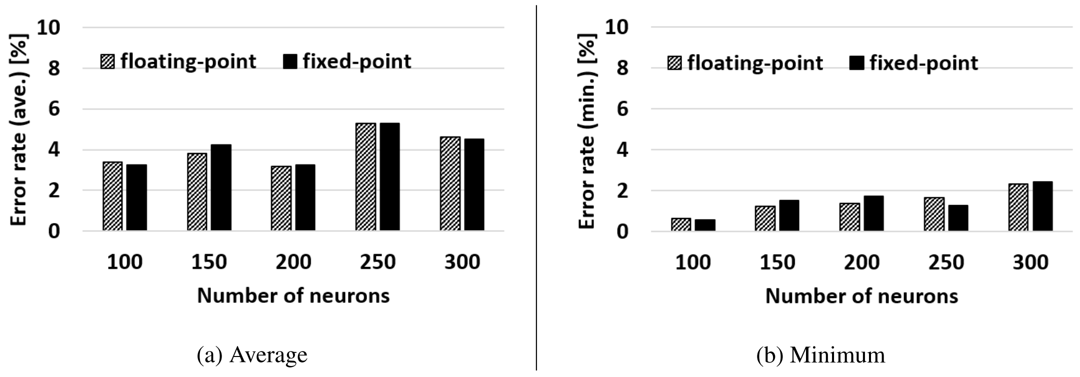


FIGURE 13. Error rates of CBM using floating-point numbers and fixed-point numbers.

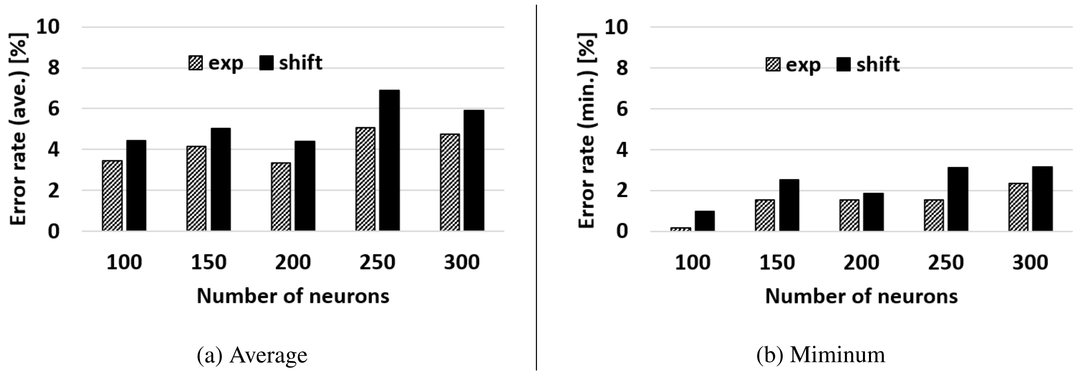


FIGURE 14. Error rates of CBM using exponential operation and shift operation.

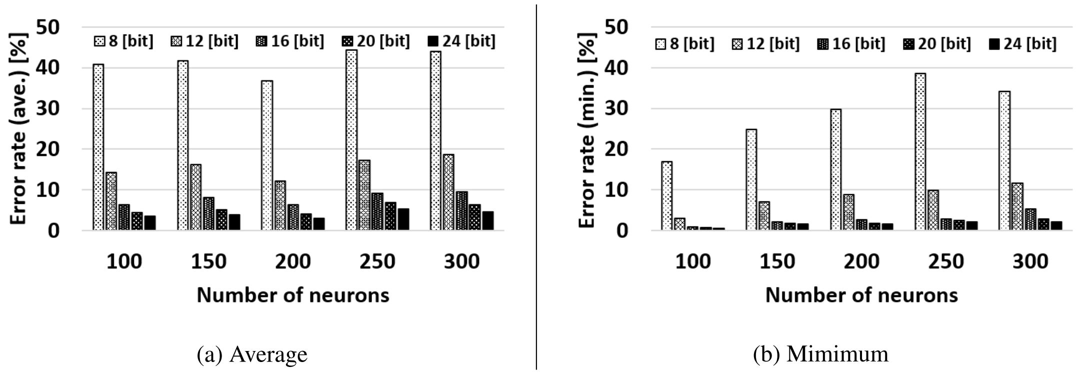


FIGURE 15. Error rates of CBMs using fixed-point numbers with various bit precision.

TABLE 4. Synthesis Environment for Neuron Circuits.

Environment	Vivado 2017.2
FPGA board	Xilinx Zynq UltraScale+ MPSoC ZCU102
Target device	Zynq UltraScale XCZU9EG-2FFVB1156 FPGA

small difference indicates the feasibility of the proposed hardware-oriented algorithm.

B. LOGICAL SYNTHESIS

The resource utilization of the circuits designed with conventional algorithms and hardware-oriented algorithms were compared to verify the resource utilization of a neuron circuit. The neuron circuits of the network, whose number of neurons are 4, 8, 16, 32, and 64 were designed and synthesized using logical synthesis. The number of LUTs, FFs, DSPs, and RAMs used in the designed circuit was measured from the results of the synthesis. The synthesis environment is shown in Table 4. The bit width of the synaptic weight, bit width of the fixed-point number of the neuron circuit, and step size of the time were set to 16, 16, and 2^{-8}, respectively.

Fig. 16 shows the results. In the neuron circuit that used the hardware-oriented algorithm, the number of resources used for LUT and FF is reduced for any number of neurons. According to the results, the resources can be reduced to

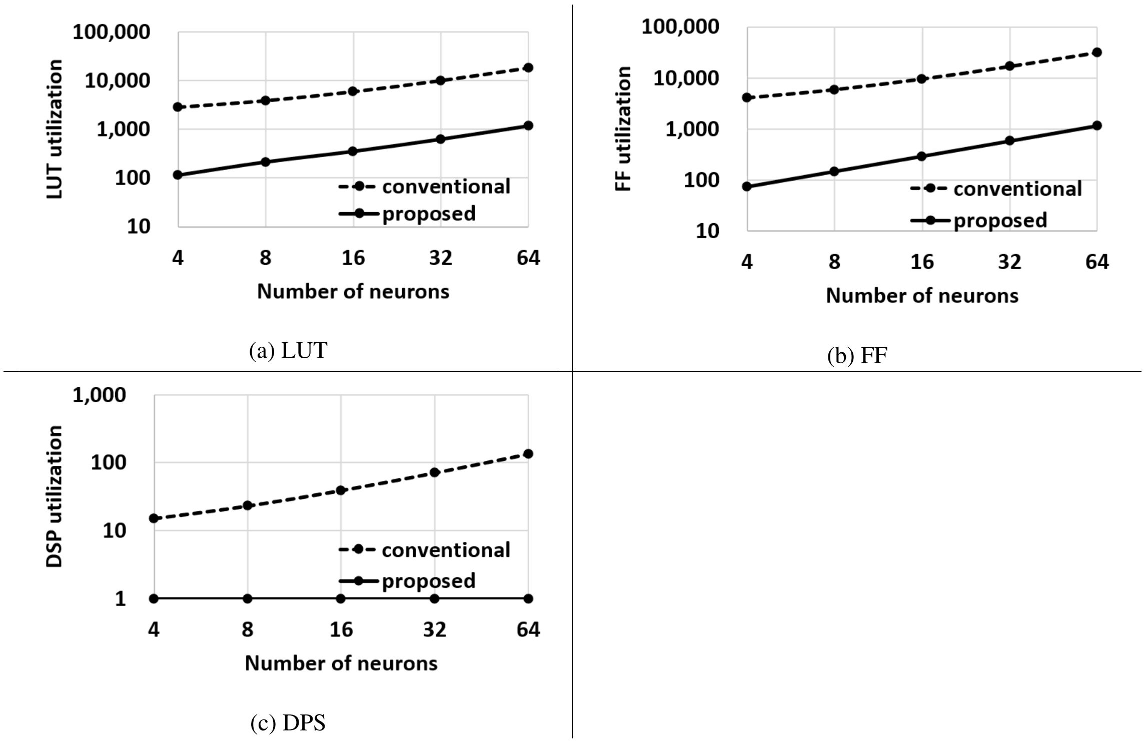


FIGURE 16. Number of resources used in neuron circuit.

TABLE 5. Synthesis Environment for Network Circuits.

synthesis environment	ISE Design Suite 14.7
FPGA board	Virtex-6 FPGA ML605
target device	Virtex-6 XC6VLX240T-1FFG1156 FPGA

TABLE 6. Synthesis Environment for Network Circuits.

synthesis environment	Vivado 2019.2
FPGA board	Virtex UltraScale+ VCU1525 Acceleration Development Board
target device	Virtex UltraScale+ XCVU9P-L2FSGD2104E FPGA

nearly 1/30, and the amount of DSP consumed in the neuron circuit designed using the conventional algorithm can be reduced. Note that block RAMs are not used in the designs of the conventional and the proposed methods.

The resources of the network circuit designed with conventional multiply-accumulate operations and differential multiply-accumulate operations were compared to validate the differential multiply-accumulate operation. The network circuits with 4, 8, 16, 32, and 64 neurons were designed and synthesized by logical synthesis. The number of LUTs, FFs, DSPs, and RAMs used in the designed circuit was measured from the synthesis results. The logical synthesis environment is shown in Table 5. The neuron circuit was designed using the hardware-oriented algorithm.

Fig. 17 shows the results for this step. In the network circuits using differential multiply-accumulate operations, the increase in the number of resources used for LUTs and FFs, for the number of neurons, was smaller than in the network circuits designed using conventional multiply-accumulate operation. Moreover, in the network circuit with 64 neurons, the number of resources used for LUT and FF was reduced to approximately 1/8 and approximately 1/70, respectively. Note that the numbers of DSPs on the conventional network and the proposed network were the same. Block RAMs were not used in the conventional network.

The proposed network was synthesized for an accelerator card. The scale of the system, which can be implemented into an accelerator card, was measured. Fig. 6 shows the synthesis environment for the card. The experiment's condition was the same as in the previous experiment.

The results are shown in Table 7 and Fig. 18. The results indicate that resource utilization increases polynomially as the number of neurons increases, and the usage of block RAMs dominates other hardware resources, such as FF, LUT, and DSP in the device. As a result of the synthesis, 2, 048 neurons in the network could be implemented in the accelerator card device.

The results of the logical synthesis show the efficiency of the hardware-oriented algorithm and the differential multiply-accumulation. Therefore, these methods reduce the hardware resource utilization drastically.

C. FPGA IMPLEMENTATION

A CBM circuit was implemented in the environment shown in Table 8. The implemented circuit is shown in Fig. 12. The

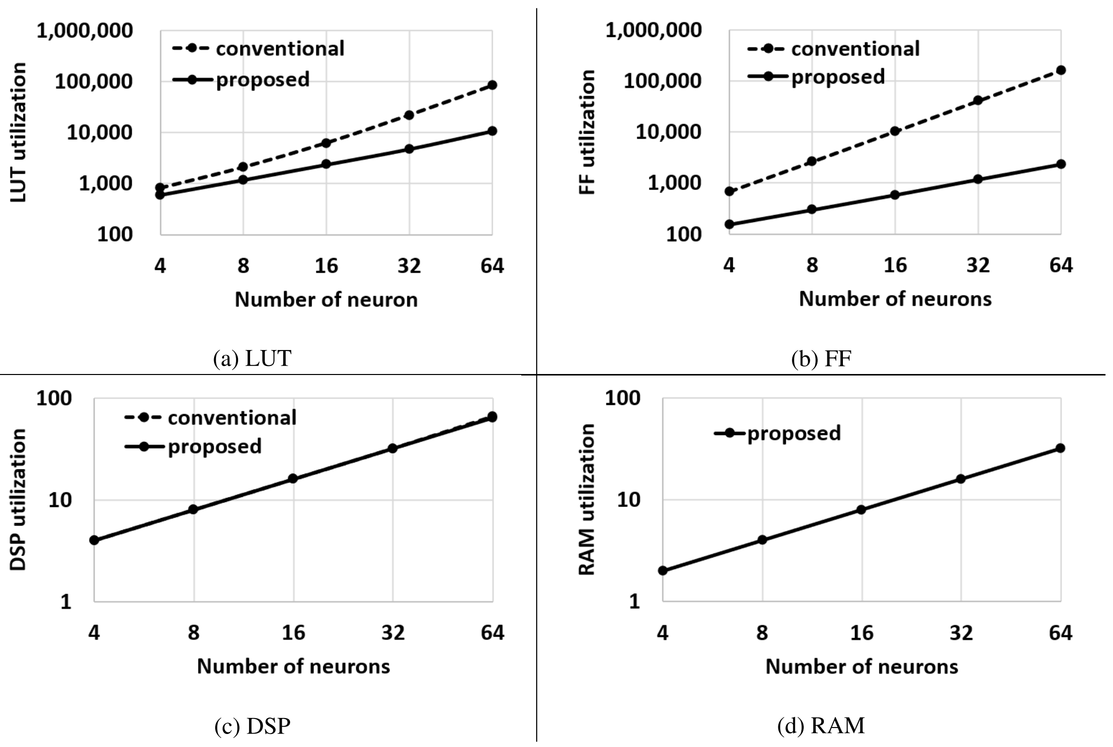


FIGURE 17. Resources used in the network circuit.

TABLE 7. Resources Used in the Network Circuits.

# of Neurons	32	64	128	256	512	1,024	2,048
# of LUTs	4,650 (0.39%)	8,866 (0.75%)	17,667 (1.49%)	34,298 (2.90%)	70,532 (5.97%)	141,751 (11.99%)	294,093 (24.88%)
# of FFs	2,539 (0.11%)	4,667 (0.20%)	9,239 (0.39%)	17,692 (0.75%)	37,906 (1.60%)	73,550 (3.11%)	156,360 (6.61%)
# of DSPs	32 (0.47%)	64 (0.94%)	131 (1.92%)	256 (3.74%)	512 (7.49%)	1,024 (14.97%)	2,048 (29.94%)
# of RAMs	16 (0.70%)	32 (1.50%)	64 (2.96%)	128 (5.90%)	256 (11.90%)	512 (23.70%)	2,048 (94.81%)

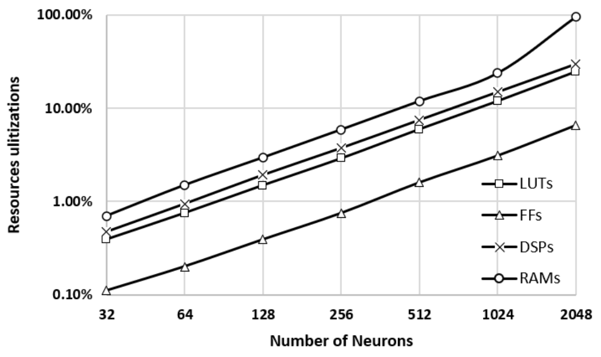


FIGURE 18. Resources used in the network circuits.

FPGA-implemented CBM was compared with the software-implemented CBM in terms of calculation ability and speed.

The computing ability of CBMs implemented into software and hardware was compared for the verification of FPGA-implemented CBM. The CBMs were applied to 100, 150, 200, 250, and 300 nodes of the maximum cut problems. The average error rate was measured as in numerical simulations (i.e., applying it ten times for each six different max-cut problems in Biq Mac Library). In this experiment, the inverted value of temperature was used instead

TABLE 8. Implementation Environment for CBM Circuits.

Synthesis environment	ISE Design Suite 14.7
FPGA board	Virtex-6 FPGA ML605
Target device	Virtex-6 XC6VLX240T-1FFG1156 FPGA

TABLE 9. Experimental Conditions for Verification of Computing Power of CBM.

The initial value of inverse temperature	1
The increasing rate of inverse temperature	$1/0.95$
Exit condition	$1/T > 150$
Time precision	2^{-8}

of temperature because the FPGA-implemented CBM needs to input the inverted value of temperature to the neuron circuit. In the experiment, annealing was performed every $N/128$ time period in the Euler method, using the values shown in Table 9. To align the conditions, the same synaptic weights and initial outputs of neurons as in the FPGA-implemented CBM were input to the software-implemented CBM. Note that the software-implemented CBM does not use the hardware-oriented algorithm to compare the calculation abilities of the original CBM and the FPGA-implemented CBM.

Fig. 19b shows the experimental results. The error rates of FPGA-implemented CBMs are found to be slightly higher than those of software-implemented CBMs. However, it was revealed that the change in error rate due to hardware implementation was less than two points. Moreover, the time change of the energy function of the software and FPGA-implemented CBM when applied to the problem "ising3.0-100_5555" is as shown in Fig. 19a and Fig. 19c, respectively. The time on the horizontal axis represents the Euler method's elapsed time, and a broken line indicates the optimum value of the applied problem. These figures indicate that the energy functions of all CBMs change similarly. As time passes, the value of the energy function approaches the optimum value as the temperature decreases.

The calculation speed of the software-implemented CBM and two kinds of FPGA-implemented CBM (i.e., a CBM with (8) as the multiply-accumulation and a CBM with (10) as the multiply-accumulation) were compared to evaluate the calculation speed of CBMs. The calculation time of FPGA-implemented CBM and the software-implemented CBM were measured. The average values of 60 iterations for each number of neurons were calculated.

Fig. 19d shows the experimental results of this step. The results show that the calculation speed was significantly improved by implementing a CBM to an FPGA, and the differential multiply-accumulation contributed to improving the calculation speed. The calculation speed was reduced to at least $1/1,300$, and the calculation speed was reduced to $1/6,500$ in the case of 300 neurons.

Therefore, the results of the FPGA implementation indicate the effectiveness of the FPGA-implemented CBM. The FPGA-implemented CBM can perform much faster calculations than the software-implemented CBM.

V. DISCUSSION

A. RESOURCES CONSUMPTION

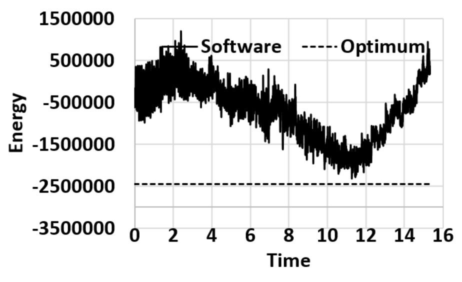
The results of this study demonstrated that our proposed method reduces resource consumption. By using the hardware-oriented algorithm, the consumption of LUTs and FFs was reduced to nearly $1/30$, as indicated in Fig. 16. The results imply that the calculation circuits on the neuron circuits are simplified using fixed-point numbers and the shift operations instead of floating-point numbers and the exponential operation; notably, the utilization of DSPs due to floating-point numbers' high bit precision and calculations of exponential bits is decreased. The multiply-accumulate operation also contributed to reducing resource consumption; the consumption of LUT and FF was reduced to approximately $1/8$ and $1/70$, respectively, in the network circuit with 64 neurons, as shown in Fig. 17. The results indicate that the increase in LUTs and registers in proportion to the square number of neurons in the network is replaced with the number of DSPs and RAMs in proportion to the number of neurons in our proposed approach. The maximum size of the network synthesized in this study was $2,048$, as shown in Fig. 18. According to this result, the number of block RAMs in the FPGA limits the size of the network due to the dominating utilization of RAMs. It may be possible to further increase the size of the network by reducing the bit precision and assigning multiple weight values in the same column of a RAM. Moreover, reduced LUT consumption indicates that more complicated neural networks can be implemented with differential multiply-accumulation.

B. CALCULATION SPEED

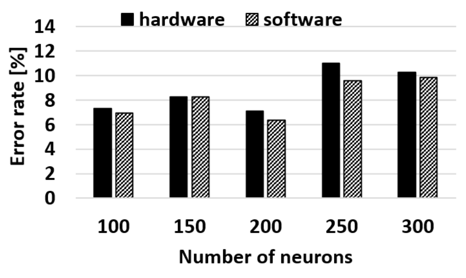
Fig. 20a demonstrates the calculation efficiency of the FPGA-implemented CBM (differential multiply-accumulation) compared with the software-implemented CBM. The results show that the calculation time of software implementation was reduced almost linearly with the number of neurons. The FPGA implementation with the differential multiply-accumulation could improve the calculation speed by at least $1,300$ times. In the case of 300 neurons, the calculation speed was $6,500$ times faster.

Fig. 20b demonstrates the calculation efficiency of FPGA-implemented CBMs (non-differential multiply-accumulation) and FPGA-implemented CBMs (differential multiply-accumulation). The figure indicates that the calculation time improves as the number of neurons in the network increases.

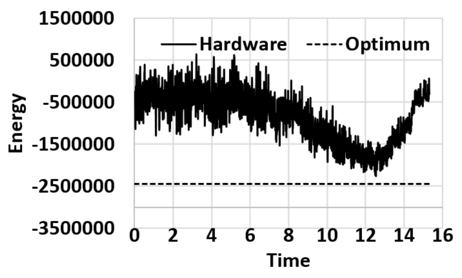
The calculation of the FPGA-implemented CBM comprises a multiply-accumulation part and neuron updating part. The differential multiply-accumulation reduces the calculation time of the multiply-accumulation part. In the non-differential multiply-accumulation, the multiply-accumulation part requires an increased calculation time that is proportional to the number of neurons. On the contrary, the differential multiply-accumulation calculation time depends on the number of neurons that change the output in the time step. Fig. 2 shows that one neuron changes its output twice (i.e., 0 to 1 and 1 to 0) in one period of the output of neurons if the input of the neuron does not change. The period can be roughly considered as $1 = (\alpha+\beta)$ on average regardless of the duration of the period depends on the input of the neuron. Hence, it could be considered that $2N/n_t$ of output changes occur in a time step on average when the period is divided into n_t time steps, where N refers to the number of neurons in the network ($n_t = 2^{12} = 4,096$ in Fig. 20b). It indicates that the differential-multiply accumulation requires $2N/n_t$ iterations for a single update of the network; meanwhile, the non-



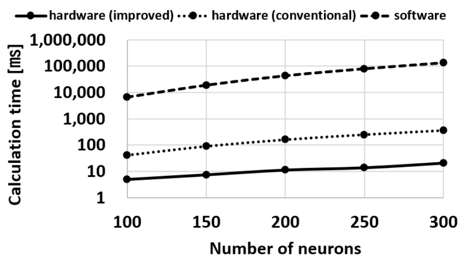
(a) Temporal change of the energy function of software-implemented CBM



(b) Error rates of software-implemented and FPGA-implemented CBMs

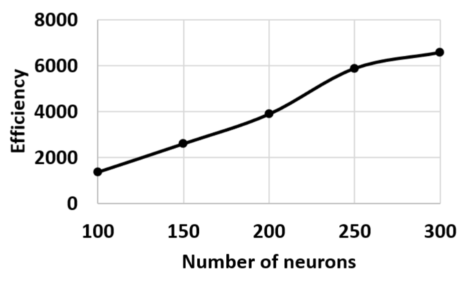


(c) Temporal change of the energy function of FPGA-implemented CBM

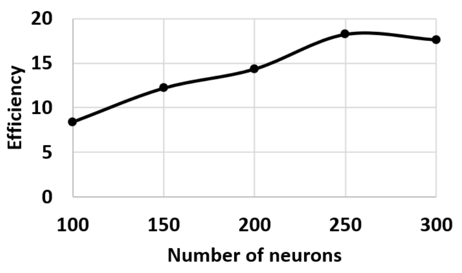


(d) Calculation times of the software-implemented CBM and FPGA-implemented CBMs (conventional: non-differential multiply-accumulation; improved: differential multiply-accumulation)

FIGURE 19. Results of the FPGA implementation.



(a) Software-implemented and FPGA-implemented CBM (differential multiply-accumulation)



(b) FPGA-implemented CBMs (non-differential multiply-accumulation) and FPGA-implemented CBMs (differential multiply-accumulation)

FIGURE 20. Efficiency of the calculation time.

differential multiply-accumulation requires N iterations for every update of the network.

C. RELATED WORK

CBMs are equivalent to Ising models; therefore, CBMs are also adaptable to combinatorial optimization problems. Many studies were related to Ising models [25], [26] and their hardware implementations [27]–[29] in recent years because many problems in our society can be expressed as combinatorial optimization problems. Table 10 compares the results of our implementation and related studies. The table shows that the number of edges in our study and the bit precision of edges is improved compared with other studies. The increased number of edges and bit precision implies fewer limitations on the mapping of combinatorial optimization problems; for instance, the traveling salesman problem (TSP) requires the square number of nodes to be equal to the number of cities in the network, and a sufficient amount of precision is required to express the constraints and the distance between cities in the problem. Note that the number of edges inside parentheses is calculated in this study, as shown in 11 (an edge between two nodes in graphs on the table is assumed to be bidirectional, and the grid of the lattice graph is assumed to be connected like a torus).

Implementations use three kinds of topologies: the complete graph, chimera graph [14], and lattice graph, as shown in Fig. 21. In a complete graph, every pair of nodes is connected by an edge; therefore, the complete graph includes

TABLE 10. **Comparison with Related Studies.**

	A. This work	B. Tsukamoto [22]	C. Yamamoto [23]	D. Yamamoto [24]	E. Yoshimura [14]	F. Gyoten [15]
Topology	Complete graph			Chimera graph [14]		Lattice graph
# of nodes	2,048	1,024	512	13,200	4,096	2,000
# of edges	2.10M	1.05M (524K)	0.25M (131K)	(71.2K)	(21.8K)	4.00K
Bit precision	16	16	5	2	2	2
Target device	Virtex-UltraScale+	Arria-10GX	CMOS (65nm)	Virtex-7	Virtex-UltraScale	Virtex-5

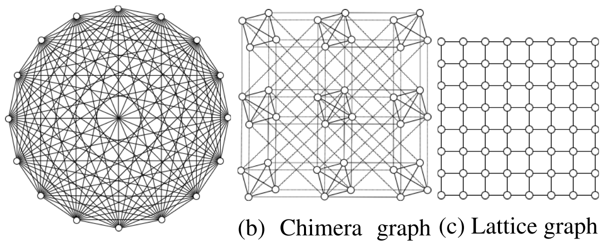

(a) Complete graph (b) Chimera graph [14] (c) Lattice graph

FIGURE 21. **Graph topologies.**

TABLE 11. **Graph Topologies.**

Topologies	Complete graph	Chimera graph [14]	Lattice graph
Size	K vertexes	$M \times N$ square grid of subgraphs	$S \times T$ square grid
# of nodes	K	$4MN$	ST
# of edges	$K(K-1)/2$	$22MN - 12M - 12N + 8$	$2ST - S - T$

other types of graphs, such as the chimera graph and lattice graph. The lattice graph is a graph whose nodes are arranged on intersections of a grid, and the nodes are connected to the adjacent four nodes. The chimera graph has four nodes of complete graphs as its subgraph, the subgraphs are arranged as the king's graph [30], and the nodes are connected to the adjacent eleven nodes. Table 11 indicates that the number of edges depending on the number of nodes in the complete graph is more than in the other graph topologies. However, the lattice graph and the chimera graph are often implemented into hardware because their connectivities are limited to the adjacent nodes; therefore, these topologies are easily accommodated into hardware that have a two-dimensional structure. In particular, the chimera graph can increase the number of connectivities for each node by limiting the connectivities to the adjacent nodes. The problem of increasing the number of edges needs to be solved to implement the complete graph topology into hardware. In our work, block RAMs in FPGAs were used to address this problem as well as a study by Yamamoto [23] that used static random access memories SRAMs. Nevertheless, it is still possible to represent a complete graph with chimera graphs by a minor embedding [31]; however, the number of edges needs to be more than the target complete graph, to represent the complete target graph by combining nodes of the original graph.

CBMs do not require random numbers in their calculation, whereas the Ising model needs to use random numbers so that it is not tricked into local optima. Random number generators require significant hardware resources; in the case of Yamamoto [23], random number generators consumed 11 percent of hardware resources in the implemented circuit. Yoshimura [14] proposed random pulse sharing to reduce the hardware resource utilization of random number generators; then, a random number generator was shared among multiple calculation units not updated at the same calculation step, nearby. However, hardware resources are still consumed by random number generators in these implementations [22], [24]. In contrast, Gyoten proposed shift-register-based spin flipper (SRSF) [15], [32]: an implementation method that enables implementation circuits of the Ising model to eliminate random number generators. In our work, random numbers generators are eradicated by the deterministic algorithm of CBMs.

To increase the number of combinatorial optimization problems that can be mapped to the network, a massive number of connectivities need to be stored in hardware such as an FPGA. Moreover, the bit precision requires to be sufficiently big to express the problems. Block RAMs in FPGAs may be used to achieve this owing to their overpowering capacity, compared to LUTs. However, values of connectivities stored in RAMs need to be accessed one by one, when referenced by the address signal. Our implementation addressed this limitation by using the time-division of the multiply-accumulation and suppressed the increasing calculation time by the differential multiply-accumulation. Additionally, the time-division enabled us to use DSPs on the multiply-accumulation. There is a similarity between this study and related research. Yamamoto [24], [33] proposed a time-division multiplexing architecture (TDM), which divided the network logically (called "phase") and updated spins phase by phase, then values of connectivities could be stored in RAMs and accessed one by one. However, the calculation time increased as the number of phases increased because of the time-division of the calculation. Yamamoto [23] could address the difficulty by the delta-driven simultaneous spin update (DDSS) that calculates updates inputs of spins by the difference from the prior inputs, as in our implementation.

The comparison shows CBMs' superiority because random numbers are not required in the hardware implementation. Moreover, our implementation's advantage is also revealed: the differential multiply-accumulation enables the utilization of block RAMs and DSPs in FPGAs. This advantage produced an FPGA implementation of the largest complete-graph network and the biggest bit precision compared with other studies.

VI. CONCLUSION

In this study, a CBM was implemented in an FPGA using the proposed hardware implementation methods (i.e., the hardware-oriented algorithms and the differential multiply-accumulation). The results of the numerical simulations indicated that the hardware-oriented algorithms were feasible. The results of the logical synthesis revealed that our proposal drastically reduced the consumption of hardware resources. Moreover, the results of the FPGA implementation show that the FPGA-implemented CBM can be applied to combinatorial optimization problems (such as the max-cut problems) and complete computations quicker than the software implementation.

The comparison of our study and previous research indicates the superiority of our implementation. The non-linear and chaotic dynamics of CBMs reduced hardware resource utilization. The network in our implementation has more edges than any other related implementation, which indicates that our implementation has fewer limitations in the mapping of combinatorial optimization problems.

The FPGA-implemented CBM proposed in this study could be applied to more practical combinatorial optimization problems in our future work. Additionally, the scale of the implementation could be expanded by improving the FPGA implementation of the differential-multiply accumulation.

ACKNOWLEDGMENT

The authors would like to thank Masatoshi Yamaguchi, Seiji Uenohara, and Hideyuki Suzuki for having a discussion and giving advice on this work. They would also like to thank Editage (www.editage.com) for English language editing.

REFERENCES

[1] G. E. Hinton, "Learning and relearning in Boltzmann machines," *Parallel Distrib. Process.*, vol. 1, pp. 282–317, Jan. 1986.

[2] M. Welling and G. E. Hinton, "A new learning algorithm for mean field Boltzmann machines," in *Proc. Int. Conf. Artif. Neural Netw.* Cham, Switzerland: Springer, 2002, pp. 351–357.

[3] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann machines," in *Proc. Artif. Intell. Statist. (AISTATS)*, 2009, pp. 448–455.

[4] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[5] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. for Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.

[6] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," 2012, *arXiv:1206.6392*. Accessed: Nov. 2, 2020. [Online]. Available: http://arxiv.org/abs/1206.6392

[7] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep Boltzmann machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2222–2230.

[8] H. Nishimori, *Statistical Physics of Spin Glasses and Information Processing: An Introduction*, no. 111. Oxford, U.K.: Oxford Univ. Press, 2001.

[9] A. Douglass, A. D. King, and J. Raymond, "Constructing SAT filters with a quantum annealer," in *Proc. Int. Conf. Theory Appl. Satisfiability Test.* Berlin, Germany: Springer, 2015, pp. 104–120.

[10] D. de Falco and D. Tamascelli, "An introduction to quantum annealing," *RAIRO-Theor. Inform. Appl.*, vol. 45, no. 1, pp. 99–116, 2011.

[11] S. Bravyi and M. Hastings, "On complexity of the quantum ising model," *Commun. Math. Phys.*, vol. 349, no. 1, pp. 1–45, Jan. 2017.

[12] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. Mcmahon, T. Umeki, K. Enbutsu, O. Tadanaga, H. Takenouchi, K. Aihara, K.-I. Kawarabayashi, K. Inoue, S. Utsunomiya, and H. Takesue, "A coherent ising machine for 2000-node optimization problems," *Science*, vol. 354, no. 6312, pp. 603–606, Nov. 2016.

[13] M. Yamaoka, "An ising computing to solve combinatorial optimization problems," in *Proc. 5th Berkeley Symp. Energy Efficient Electron. Syst. Steep Transistors Workshop (E3S)*, Oct. 2017, pp. 1–3.

[14] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, "Implementation and evaluation of FPGA-based annealing processor for ising model by use of resource sharing," *Int. J. Netw. Comput.*, vol. 7, no. 2, pp. 154–172, 2017.

[15] H. Gyoten, M. Hiromoto, and T. Sato, "Area efficient annealing processor for ising model without random number generator," *IEICE Trans. Inf. Syst.*, vol. 101, no. 2, pp. 314–323, 2018.

[16] H. Suzuki, J.-I. Imura, Y. Horio, and K. Aihara, "Chaotic Boltzmann machines," *Sci. Rep.*, vol. 3, no. 1, p. 1610, Dec. 2013.

[17] M. Yamaguchi, Y. Katori, D. Kamimura, H. Tamukoh, and T. Morie, "A chaotic Boltzmann machine working as a reservoir and its analog VLSI implementation," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–7.

[18] M. Yamaguchi, H. Tamukoh, H. Suzuki, and T. Morie, "A CMOS chaotic Boltzmann machine circuit and three-neuron network operation," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, May 2017, pp. 1218–1224.

[19] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[20] E. Billauer. (2017). *Xillybus Product Brief.* Nov. 2, 2020. [Online]. Available: http://xillybus.com/downloads/xillybus_product_brief.pdf

[21] A. Wiegele, "Biq Mac Library–A collection of Max-Cut and quadratic 0-1 programming instances of medium size," Dept. Math., Alpen-Adria-Universität, Klagenfurt, Austria, Tech. Rep., 2007. Accessed: Nov. 2, 2020. [Online]. Available: http://biqmac.uni-klu.ac.at/biqmaclib.pdf

[22] S. Tsukamoto, M. Takatsu, S. Matsubara, and H. Tamura, "An accelerator architecture for combinatorial optimization problems," *FUJITSU Sci. Tech. J*, vol. 53, no. 5, pp. 8–13, 2017.

[23] K. Yamamoto, K. Ando, N. Mertig, T. Takemoto, M. Yamaoka, H. Teramoto, A. Sakai, S. Takamaeda-Yamazaki, and M. Motomura, "STATICA: A 512-Spin 0.25M-weight full-digital annealing processor with a near-memory all-spin-updates-at-once architecture for combinatorial optimization with complete spin-spin interactions," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2020, pp. 138–140.

[24] K. Yamamoto, W. Huang, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, and M. Motomura, "A time-division multiplexing ising machine on FPGAs," in *Proc. 8th Int. Symp. Highly Efficient Accel. Reconfigurable Technol. (HEART)*, 2017, pp. 1–6.

[25] S. Kanamaru, D. Oku, M. Tawada, S. Tanaka, M. Hayashi, M. Yamaoka, M. Yanagisawa, and N. Togawa, "Efficient ising model mapping to solving slot placement problem," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Jan. 2019, pp. 1–6.

[26] K. Terada, D. Oku, S. Kanamaru, S. Tanaka, M. Hayashi, M. Yamaoka, M. Yanagisawa, and N. Togawa, "An ising model mapping to solve rectangle packing problem," in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2018, pp. 1–4.

[27] C. Yoshimura, M. Hayashi, T. Takemoto, and M. Yamaoka, "CMOS annealing machine: A domain-specific architecture for combinatorial optimization problem," in *Proc. 25th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2020, pp. 673–678.

[28] T. Takemoto, M. Hayashi, C. Yoshimura, and M. Yamaoka, "A 2×30k-spin multi-chip scalable CMOS annealing processor based on a processing-in-memory approach for solving large-scale combinatorial optimization problems," *IEEE J. Solid-State Circuits*, vol. 55, no. 1, pp. 145–156, Jan. 2020.

[29] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "A 20k-spin ising chip to solve combinatorial optimization problems with CMOS annealing," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 303–309, Jan. 2016.

[30] G. J. Chang, "Algorithmic aspects of domination in graphs," in *Handbook of Combinatorial Optimization*. Boston, MA, USA: Springer, 1998, pp. 1811–1877.

[31] M. Juenger, E. Lobe, P. Mutzel, G. Reinelt, F. Rendl, G. Rinaldi, and T. Stollenwerk, "Performance of a quantum annealer for ising ground state computations on chimera graphs," 2019, *arXiv:1904.11965*. Accessed: Nov. 2, 2020. [Online]. Available: http://arxiv.org/abs/1904.11965

[32] H. Gyoten, M. Hiromoto, and T. Sato, "Enhancing the solution quality of hardware ising-model solver via parallel tempering," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2018, pp. 1–8.

[33] K. Yamamoto, M. Ikebe, T. Asai, M. Motomura, and S. Takamaeda-Yamazaki, "FPGA-based annealing processor with time-division multiplexing," *IEICE Trans. Inf. Syst.*, vol. 102, no. 12, pp. 2295–2305, 2019.

ICHIRO KAWASHIMA (Student Member, IEEE) received the B.Eng. and M.Eng. degrees from the Kyushu Institute of Technology, in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree with the Graduate School of Life Science and Systems Engineering Department. He is a Student Member of IEICE.

TAKASHI MORIE (Member, IEEE) received the B.S. and M.S. degrees in physics from Osaka University, Osaka, Japan, in 1979 and 1981, respectively, and the Dr.Eng. degree from Hokkaido University, Sapporo, Japan, in 1996. From 1981 to 1997, he was a Research Staff Member with Nippon Telegraph and Telephone Corporation (NTT). From 1997 to 2002, he was an Associate Professor of the Department of Electrical Engineering, Hiroshima University, Higashihiroshima, Japan. Since 2002, he has been a Professor of the Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Kitakyushu, Japan. His research interests include the VLSI implementation of neural networks and new functional nanodevices.

HAKARU TAMUKOH (Member, IEEE) received the B.Eng. degree from Miyazaki University, Japan, in 2001, and the M.Eng. and Ph.D. degrees from the Kyushu Institute of Technology, Japan, in 2003 and 2006, respectively. From April 2006 to September 2007, he was a Postdoctoral Research Fellow of the 21st Century Center of Excellence Program, Kyushu Institute of Technology. From October 2007 to January 2013, he was an Assistant Professor with the Tokyo University of Agriculture and Technology. He is currently an Associate Professor with the Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology. His research interests include hardware/software complex systems, digital hardware design, neural networks, soft computing, and home service robots. He is a member of IEICE, SOFT, JNNS, JSAI, and RSJ.