FPGA Implementation of a Chaotic Boltzmann Machine Annealer

Kanta Yoshioka*, Yuichi Katori †, Yuichiro Tanaka ‡, Osamu Nomura *‡, Takashi Morie *‡ and Hakaru Tamukoh*‡
*Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, Japan
†The School of Systems Information Science, Future University Hakodate, Japan
‡Research Center for Neuromorphic AI Hardware, Kyushu Institute of Technology, Japan

***Abstract*—Ising machines are attracting attention for their ability to solve large-scale combinatorial optimization problems because these problems are difficult to solve. To accelerate the computing of Ising machines, implementation of Ising machines with digital circuits such as simulated annealing (SA) machines is in progress. However, these Ising machines on digital circuits require random number generators, which are implemented with large circuit resources. This work focuses on chaotic Boltzmann machines (CBMs), which imitate the stochastic behavior of Boltzmann machines (BMs) with deterministic chaotic dynamics. CBMs are one of the models that work as chaotic simulated annealing (CSA) machines within Ising machines. Therefore, we can implement the Ising machines without random number generators by using CBMs. In conventional work, CSA machines using CBMs (CBM-CSAs) are implemented with some hardware-oriented algorithms, but the CBM-CSA circuit is not optimized for these hardware-oriented algorithms. In the conventional CBM-CSA circuit, memory circuits are implemented separately, which prevents making the CBM-CSA from larger, and neuron circuits require the reset of accumulated values, which causes the increase in the calculation time. To solve these problems, we implement only one large memory circuit to make the CBM-CSA larger and improve the neuron circuits to allow dynamic changes of inputs to arithmetic circuits to inhibit the increase in the calculation time. As a result, we implement a CBM-CSA with 4096 nodes on an FPGA (Alveo U250), and the CBM-CSA can control 16-bit width weights and run at 100MHz. We evaluate the implemented CBM-CSA by solving K_{4000}, max-cut problem, which is one of the combinatorial optimization problems. The best solution of CBM-CSA is comparable to that of the SA on the central processing unit (CPU). Moreover, the CBM-CSA is approximately 600 times as fast as the SA on the CPU and approximately twice as fast as the conventional Ising machine on an FPGA based on the improvements in this work. Furthermore, this work implements one of the highest-performance Ising machines on a single FPGA.**

***Index Terms*—Ising machine, field programmable gate array, chaotic simulated annealing**

I. Introduction

With the end of Moore's law, the performance of conventional computers is reaching its limits, and the development of next-generation computers with different operating principles from conventional computers such as central processing units (CPUs) and graphics processing units (GPUs) is progressing. One of the next-generation computers is annealing quantum computers [1], which are one of the Ising machines that are attracting attention for their ability to solve various combinatorial optimization problems with high speed [2]. However, quantum annealing machines require large-scale facilities because they operate at extremely low temperatures [3]. Therefore, implementations of Ising machines on digital circuits such as field programmable gate arrays (FPGAs) [4] [5] or application-specific integrated circuits (ASICs) [6] are progressing, and the commercial use of these machines is being widely adopted. Currently, some different types of Ising machines are proposed, such as simulated annealing (SA) machines, chaotic simulated annealing (CSA) machines, simulated bifurcation (SB) machines, and simulated quantum annealing (SQA) machines [7] [8].

This work focuses on chaotic Boltzmann machines (CBMs) [9], which are neural network models that are equivalent to the Ising model. CBMs behave deterministically by imitating the stochastic behavior of Boltzmann machines (BMs) by chaotic dynamics and are one of the models, which work as CSA machines. The optimization process of CSA does not require random numbers, while that of SA requires random numbers. Therefore, CSA machines are expected to be implemented with smaller circuit resources than SA machines because CSA machines can be implemented without random number generators.

In a conventional work [10], CSA machines using CBMs (CBM-CSAs) circuits are implemented with some hardware-oriented algorithms, such as the simplification of the exponential function and the differential multiply-accumulation [10] [11]. However, the conventional CBM-CSA is small-scale, and the conventional circuit does not take full advantage of the hardware-oriented algorithms.

We focus on memory circuits and neuron circuits, which are not optimized for hardware-oriented algorithms. In the conventional CBM-CSA, memory circuits are implemented separately for each neuron. Thereafter, the number of CBM neurons that can be implemented on an FPGA does not depend solely on the total amount of memories but also on the number of physical memory elements on an FPGA. To solve this problem, we implement one large memory circuit whose limitation depends solely on the total amount of memories. In the conventional CBM-CSA, its calculation is time-consuming because we have to reset accumulated values, which are used in the process of the differential multiply-accumulation, in neuron circuits. Therefore, we improve neuron circuits to allow dynamic changes of inputs to prevent the increase in

calculation time and take full advantage. As a result of improvements, we implemented one of the highest-performance Ising machines.

II. Related works

A. Chaotic Simulated Annealing

SA is one of the optimization processes of the Ising machines and uses the Ising model in physics as a principle to solve combinatorial optimization problems [2]. Combinatorial optimization problems are mapped as the weights between nodes of the Ising model. Then, a network state with the smallest energy (good solution) for the combinatorial optimization problem is searched. When searching network states, if network states with decreasing energy are only accepted, network states are trapped in local minimum solutions like Hopfield neural networks (HNNs). Therefore, changes in network states in the direction of increasing energy in stochastic manners are necessary. Random numbers are required for such stochastic behavior.

CSA is an optimization process based on deterministic chaotic dynamics [8], unlike SA which operates in stochastic manners. There are mainly two significant differences between SA and CSA. The first is that SA is based on stochastic Monte Carlo methods, while CSA is based on deterministic chaotic dynamics. The second is that the convergence process of SA is based on a control of thermal fluctuations, while that of CSA is based on a control of bifurcation structures.

Because of the character of CSA, CSA machines can be implemented without random number generators which require large circuit resources. For example, Yamamoto implemented Ising machines and reported that approximately 11% of the total circuit resources were used for the random number generators in [6]. Therefore, CSA machines can be implemented with smaller circuit resources than Ising machines that require random number generators.

B. Chaotic Boltzmann Machines

One of the most famous models that work as SA machines is BMs. BMs are neural network models that operate stochastically by using random numbers. BMs use an input z_i to the i-th node as in Eq. (1), and the output of the i-th node will be one with probability $p(s_i = 1|z_i)$ as represented in Eq. (2). N, w_{ij}, s_i, and b_i in Eq. (1) are the number of nodes, weights between the i-th node and the j-th node, the output states of the i-th node, and the bias value of the i-th node, respectively. T in Eq. (2) is a temperature parameter, which represents how acceptably the energy is changed in the direction of increase when performing as SA machines. If $T \simeq 0$ without noise effects, only the change of states in the direction of decreasing the energy is accepted, hence it works like HNNs.

$$z_i = \sum_{j \neq i}^{N} w_{ij} s_j + b_i, \tag{1}$$

$$p(s_i = 1|z_i) = \frac{1}{1 + \exp(-z_i/T)}. \tag{2}$$

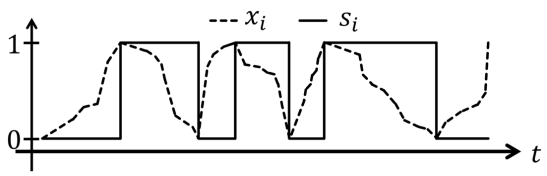

Fig. 1. Temporal change of the internal state and the output of i th neuron ©Ichiro Kawashima, 2020 (Licensed under CC BY 4.0) https://creativecommons.org/licenses/by/4.0/

One of the most famous models that work as CSA machines is CBMs. CBMs are neural networks that behave deterministically by imitating the stochastic behavior of BMs by the chaotic dynamics [9]. CBMs are constructed by neurons that have the internal state x_i and the output state s_i, respectively, and run in continuous time. The i-th internal state x_i is updated using Eq. (3) and the i-th output state $s_i \in \{0, 1\}$ deterministically changes when only when x_i reaches 0 or 1 in Eq. (4).

$$\frac{dx_i}{dt} = (1 - 2s_i)(1 + \exp\frac{(1 - 2s_i)z_i}{T}), \tag{3}$$

$$s_i = \begin{cases} 1 \ (x_i = 1), \\ 0 \ (x_i = 0). \end{cases} \tag{4}$$

If we focus on one node in CBMs, the internal state x_i and the output state s_i change because of these dynamics, as shown in Fig. 1. The internal state x_i of the i-th node change chaotically in the range of [0,1], and the output state s_i also changes according to x_i.

Eqs. (2) and (3) indicate that the parameter T of BMs controls thermal fluctuations, while that of CBMs controls bifurcation structures. Therefore, BMs are one of the models that work as SA machines and CBMs are one of the models that work as CSA machines.

C. Hardware-oriented algorithms

The conventional CBM-CSA implementation [10] uses some hardware-oriented algorithms to design high-performance and efficient circuits.

1) Simplification of exp functions: The exponential operation in Eq. (3) is replaced with a shift operation, as shown in Fig. 2. To implement functions such as sin, cos, and exp into digital circuits, the table approximation and the coordinate rotation digital computer (CORDIC) [12] methods are often used. These methods can perform highly accurate approximate calculations by storing the results of calculations as a table in advance. However, they use large circuit resources. Therefore, Kawashima [10] simplified the exponential function by using the shift operation as shown in Eq. (5) with the floor function $\lfloor x \rfloor$ and the ceiling function$\lceil x \rceil$, as shown in Fig. 2. Therefore, the calculation accuracy declines, but the approximate exponential functions in digital circuits are implemented with simple shift operations and small circuit resources.

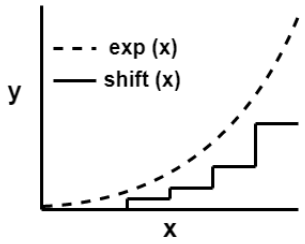

Fig. 2. Exponential and shift operations

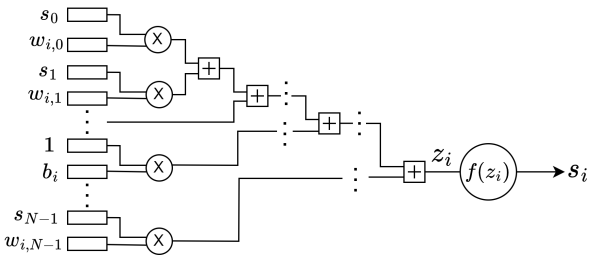

Fig. 3. All-parallel implementation of s_i calculation circuit

$$shift(x) = \begin{cases} 2^{\lfloor x \rfloor}(x \geq 0), \\ 2^{\lceil x \rceil}(x < 0). \end{cases} \tag{5}$$

2) Differential multiply-accumulation: A differential multiply-accumulation is a scheme for time-domain neural processing, which reduces hardware resources utilization of multiply-accumulation with a small increase in computational time [10] [11].

One of the approaches to speeding up the multiply-accumulation operation is the calculation of z_i in Eq. (1), by implementing all multiply-accumulation operators in parallel in Fig. 3.

However, the number of multiply-accumulation operators increases in proportion to the square of the number of nodes implemented in this way because CBMs are all-to-all connected networks. Additionally, the weight data must be stored in distributed random access memories (RAMs) not block RAMs (BRAMs) despite the number of distributed RAMs in an FPGA is limited because all of the weights data is required to be accessed at the same time to compute the multiply-accumulate operations in parallel. Therefore, implementing all multiply-accumulate operations in parallel is difficult.

A technique that is often used is the time-division of the operations. A circuit that introduces a time-division technique is shown in Fig. 4. In the simple time-division calculations, the input value of the i-th neuron z_i is defined as Eqs. (6) and (7) where $\tau \in [0, N-1]$ and A_i^τ represent an iterator of the time-division and an accumulated value at time τ, respectively. The circuit in Fig. 4 holds the accumulated value in a register and adds the input values to it.

$$z_i = A_i^{N-1}, \tag{6}$$

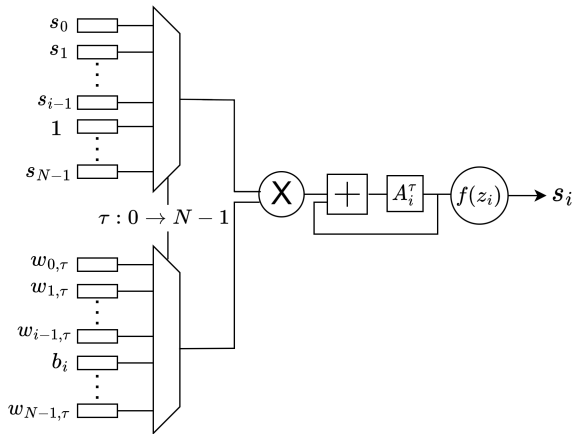

Fig. 4. Time-division implementation of s_i calculation circuit

$$A_i^\tau = \begin{cases} A_i^{\tau-1} + b_i & (\tau = i) \\ A_i^{\tau-1} + s_\tau w_{i,\tau} & (\tau \neq i) \end{cases}, \ A_i^{-1} = 0. \tag{7}$$

By using the simple time-division method, only N multiply-accumulation operators are implemented as N digital signal processors (DSPs) in an FPGA, and z_i in Eq. (1) is calculated by using them N times. Therefore, the number of multiply-accumulation operators increases in proportion to the number of nodes N, and the weights can be stored in BRAMs, whereas in the aforementioned all-parallel implementation, all weights data are stored in distributed RAMs.

However, for circuits that introduce the simple time-division of the operations, the calculation time increases significantly. For example, for $N = 1000$, the multiply-accumulation operators must be used 1000 times to calculate z_i in Eq. (1), and the calculation time increases in proportion to the number of nodes.

Because of these disadvantages, the conventional CBM-CSA introduces a differential multiply-accumulation method [10] [11], which is an improved hardware-oriented algorithm of the conventional simple time-division method described above. Furthermore, the increase in calculation time by time-division calculations can be reduced by focusing on the change of each neuron in a time-domain neural network. The calculation is described by the following Eqs. (8), (9) and (10), where t, z_i^t, and d_j^t represent time, the input of the i-th neuron at time t, and the difference between the former output of the j-th neuron s_j^{t-1} and the present output for s_j^t, respectively.

$$z_i^0 = b_i + \sum_{j \neq i}^{N} s_j^0 w_{ij}, \tag{8}$$

$$z_i^t = z_i^{t-1} + \sum_{\substack{j \neq i \\ s_j^{t-1} \neq s_j^t}} d_j^t w_{ij}, \tag{9}$$

$$d_j = \begin{cases} -1 & (s_j^{t-1} = 1, s_j^t = 0), \\ 1 & (s_j^{t-1} = 0, s_j^t = 1). \end{cases} \tag{10}$$

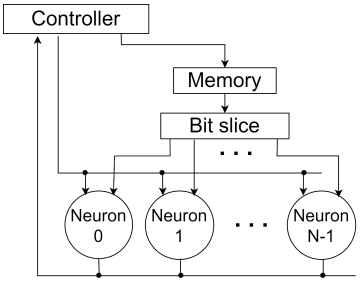

Fig. 5. Overall of CBM circuit

When $t = 0$, the calculation time is equal to the conventional simple time-division calculations to accurately calculate z_i^0. However, after the second time of the calculations ($t > 0$), the operation is performed according to Eqs. (9) and (10) for only the neurons changed. Therefore, the calculation time can be significantly reduced compared to the conventional time division calculation. In [10], Kawashima reported that by introducing the differential multiply-accumulation to CBM-CSAs, the calculation time of a CBM-CSA with 300 nodes is approximately one-twentieth of that of the conventional simple time-division calculations.

III. Proposed Circuit design

The overall view of a circuit designed in this work is shown in Fig. 5. There are four major circuits inside the CBM-CSA: a controller, a memory, a bit slice, and a neuron. The controller is a state machine that controls the operations of the CBM-CSA. The controller determines temperature parameter changes and controls calculations of the iterator τ in Fig. 4. The memory stores the weights between neurons. The bit slice divides a large bit string into small bit strings. The neuron is a circuit that actually performs the operations. It updates the internal state x_i by the calculation Eq. (3) and returns the respective output state s_i to the controller. In this way, the current output state is compared with the past output state in the controller, and we can use the differential multiply-accumulation.

A. Memory circuits

In the conventional CBM-CSA [10], the memory circuits are shown in Fig. 6. Memory circuits are created separately for each neuron. Therefore, the number of CBM-CSA neurons that could be implemented on an FPGA does not depend solely on the total amount of memories but also on the number of physical memory elements on an FPGA.

This work improves and implements only one large memory circuit, as shown in Fig. 7. It has an address from 0 to $N-1$, and the data at each index has all the weights from that index to the other neurons together. In addition, the memory usage is reduced by storing bias b_i in the $w_{i,i}$ location because CBMs have no self-weights $w_{i,i} = 0$, as shown in Fig. 7. Then, the obtained weights are divided by the bit slice and sent to individual neurons.

Creating one large memory circuit, rather than implementing memory circuits for each neuron individually is efficient

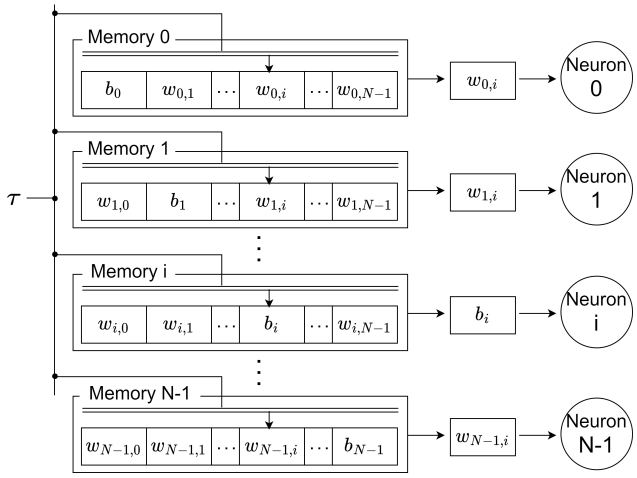

Fig. 6. Conventional memory circuit

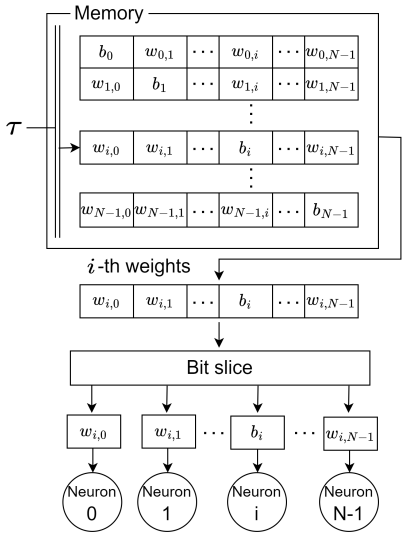

Fig. 7. Proposed memory circuit

for the following reasons. If memory circuits are installed in each neuron, at least N physical memory elements are required. Therefore, the number of nodes that can be implemented is greatly affected by the number of memory elements, and not by the total amount of memory in an FPGA. Furthermore, conventional memory circuits are more wasteful than proposed memory circuits. Xilinx FPGAs include BRAMs and ultra-RAMs (URAMs), with capacities of 36 kb and 288 kb, respectively. If memory circuits are implemented separately similar to conventional memory circuits, only half capacities of a RAM are actually often used.

B. Neuron circuits

The conventional CBM-CSAs [10] have a problem with a large increase in calculation time when the temperature is changed. Fig. 8 shows the increase in the calculation time when the temperature is changed every ten epochs, and the graph shape of the conventional CBM-CSA looks similar to

a stair. This significant increase in calculation time is caused by the reset of accumulated value in the conventional neuron circuits.

Fig. 9 shows the conventional neuron circuit. In the circuit, the DSP operation mode is multiply-accumulate (MACC). The two inputs to the DSP are A and B, respectively, the DSP's internal register is C, and the output is Z. In the MACC mode, $Z = A * B + C$ and C is updated by Z. In the conventional CBM-CSAs, A and B are $w_{i,\tau}$ or b_i and $1/T$, respectively. The internal register C hold $C_i^\tau = \sum_\tau w_{i,\tau}/T + b_i/T$ and the differential multiply-accumulation is realized by using the $ctrl$ signal to decide whether to add or subtract inputs to C_i^τ. Furthermore, the conventional CBM-CSAs require calculating Eq. (8) to reset the internal register C at the time of temperature changes, so the calculation time becomes large.

This work proposes a novel neuron circuit to remove the reset that increases the calculation time. We add a register rZi to hold the previous input z_i and the $Stage$ signal that controls the operation, as shown in Fig. 10. These changes in the neuron circuit enable dynamically changing inputs to a DSP and remove the reset. The calculation time of the proposed CBM-CSAs increases purely depending on the number of nodes that might have changed. Therefore, the increase in the calculation time is inhibited to a smooth increase like Fig. 8.

Details of the proposed neuron circuits are described below. The CBM-CSAs operation can be roughly divided into three stages: Stage0, Stage1, and Stage2, which are the calculation of z_i, the calculation of z_i/T, and the update of the internal state by $x_i + dx_i/dt$, respectively. In Stage0 ($Stage = 0$), the weights or bias and 1 are fed to A and B, respectively, and the differential multiply-accumulation is implemented by a DSP that operates in MACC mode. The internal register C hold $C_i'^\tau = \sum_\tau w_{i,\tau} + b_i$ in the proposed neuron circuits. In Stage1 ($Stage = 1$), z_i that is obtained by the differential multiply-accumulation and stored in the register rZi and temperature parameter $1/T$ are fed to A and B, respectively, and the operation of z_i/T was performed by setting $C = 0$. In Stage2 ($Stage = 2$), the internal state x_i is updated by the adder that is constructed by LUTs in the latter of the DSP. Meanwhile, we feed rZi and 1 to A and B, respectively, and initialize C to z_i. As shown above, inputs are changed dynamically by adding rZi and the $Stage$ signal.

IV. Implementation

A. Implemented systems

We implemented a system as shown in Fig. 11. We controlled a CBM-CSA implemented on an FPGA by the host program running on a CPU. The CBM-CSA on an FPGA includes the proposed circuits described above. The host sends data such as weights and some parameters through PCI-Express (PCIe) to the CBM-CSA on the FPGA and receives the calculated results from the FPGA. We use an open-source library named PyQUBO for generating the weights [13] [14]. Furthermore, we can automatically generate weights that can be mapped to the Ising model from an energy function E such as representing a combinatorial optimization problem

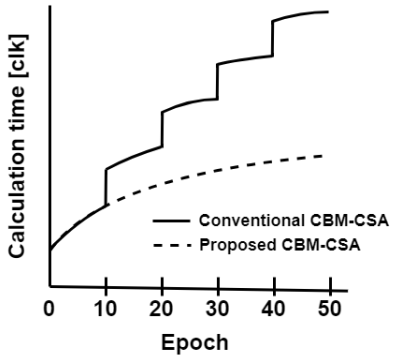

Fig. 8. Change of calculation time (Conventional circuit vs Proposed circuit)

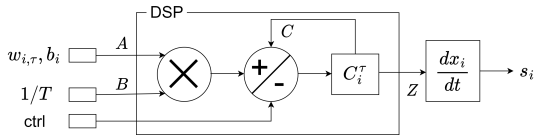

Fig. 9. Conventional neuron circuit

with PyQUBO. In addition, we use PYNQ, a library provided by AMD Xilinx, for communication between the FPGA and the CPU [15]. The software versions and a target board are listed in Table I. Our target board is Alveo U250 which is one of the high-end FPGAs provided by AMD Xilinx.

B. Implementation results

As a result of the implementation using the environment as shown in Table I, we implemented a CBM-CSA with 4096 nodes on an FPGA board. The CBM-CSA controls 16-bit width weights and bias and runs at 100MHz. The circuit utilization including all circuits such as communication with the CPU is shown in Table II, where LUT, LUTRAM, FF, are look up table, LUT used as memory, and flip-flop, respectively.

V. Experiments

We experimented with solving max-cut problems to verify the performance of the implemented CBM-CSA. The max-cut problem is one of the most famous combinatorial optimization problems. Furthermore, it is to divide the nodes of a weighted graph ($w_{ij} = w_{ji}$ is the weight between the i-th node and the j-th node) into two groups such that the result maximizes the total weight of the edges. The total weight of the edges is called the cut value $Cut(\mathbf{s})$ and is represented in Eq. (11).

$$Cut(\mathbf{s}) = \sum_{i=0}^{N-1} \sum_{j=i}^{N-1} w_{i,j}(2s_i s_j - s_i - s_j). \qquad (11)$$

K_{4000} is one of the max-cut problems for a complete graph of 4000 nodes and is the benchmark problem used in [4]. We created the K_{4000} max-cut problem with a pseudo-random number algorithm called the Mersenne Twister [16] with $seed = 1$ as the same as [4].

In addition, we made and solved the max-cut problem with weights between nodes in the range of $[-100, 100]$,

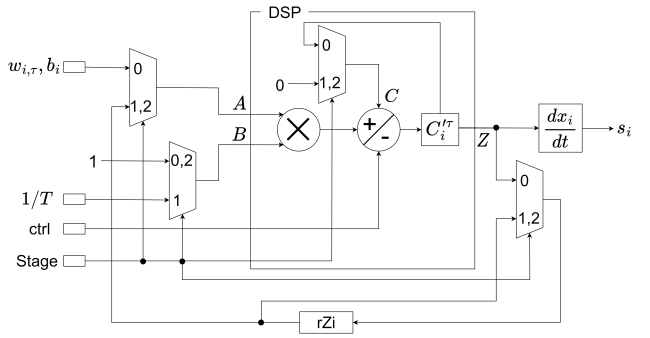

Fig. 10. Proposed neuron circuit

Fig. 11. Overall of the CBM-CSA system

$R[-100, 100]_{4000}$ because the implemented CBM-CSA can control 16-bit width weights and we wanted to test abilities to control multi-bit weights, although there were not such problems. There are some max-cut problems such as Biq Mac [17], G-set [18] and K_{4000}, but those problems are small or have only $\{-1, 1\}$ weights; therefore, we created $R[-100, 100]_{4000}$. We created $R[-100, 100]_{4000}$ whose weights were defined as: $w_{ij} = w_{ji} = r_i$, where r_i is generated number by Mersenne Twister with $seed = 1$ in the range of $[-100, 100]$.

The hyperparameters of these experiments are shown in Table III. In Table III, dt and the number of epochs refers to the number of time divisions of the CBM operation in Eq. (3) and the number of times calculate Eq. (3) and update x_i per one temperature, respectively.

The results of ten times averages of both our experiments are shown in Tables IV and V. In this experiment, the calculation time refers to the time required to reach the HNN energy, because it is difficult to search for the globally optimal solution on the scale of problems. This definition of calculation time is the same as in the conventional work [4]. Averages of ten times of the solution obtained by HNN run for long iterations were -88,405 (K_{4000}) and -5,119,074 ($R[-100, 100]_{4000}$).

Additionally, in Tables IV and V, Optimized CPU-SA and Unoptimized CPU-SA are the results of K_{4000} and $R[-100, 100]_{4000}$ solved by SA run on a CPU. Optimized CPU-SA is the result in [4], which is the result of optimizing the cache usage and calculation method for the CPU. The result of Optimized CPU-SA is not in Table V because $R_{4000}[-100, 100]$ is the original max-cut problem. Unoptimized CPU-SA is the result of [19], which we run on AMD Ryzen Treadripper PRO 3995WX.

Regarding the result of K_{4000} in Table IV, the best solution of this work was comparable to that of Unoptimized CPU-SA. This work was approximately twice as fast as the conventional SB machine on an FPGA (FPGA-SB) [4]. The median result of experiments is shown in Fig. 12. Furthermore, the number of epochs and calculation time were 185 and 0.110 [ms], respectively. In addition, the best solution was obtained at 448 epochs and 0.175 [ms].

Regarding the result of $R[-100, 100]_{4000}$ in Table V, the best solution of the implemented CBM-CSA was comparable to that of Unoptimized CPU-SA similar to the result of K_{4000}. Furthermore, it was dramatically faster than Unoptimized CPU-SA as well.

TABLE I
IMPLEMENTATION ENVIRONMENT

Target board	Alveo U250
Vitis, Vivado, VitisHLS	v2021.2
Python	v3.8.10
PyQUBO [13] [14]	v1.2.0
PYNQ [15]	v2.7.0

TABLE II
RESOURCE UTILIZATION

Elements	Used	Available	Utilization[%]
LUT	1,114,049	1,726,216	64.54
LUTRAM	20,479	790,200	2.59
FF	719,766	3,456,000	20.83
BRAM	655	2,688	24.37
URAM	911	1,280	71.17
DSP	4,116	12,288	33.50

VI. DISCUSSION

A. Implementations

The implementation result is compared with the conventional implementations on an FPGA, as shown in Table VI.

Regarding SA and SQA, when compared to sparsely connected Ising machines such as chimera graphs and king graphs, the number of nodes is equal to or more. However, when solving actual problems, to represent dense connections with sparse connections, graph embedding must be performed, which greatly reduces the number of nodes that can actually be used. For example, it is difficult to solve K_{4000}, an all-connected optimization problem by using [5] or [20].

Regarding other algorithms of Ising machines, the number of nodes for SQA is 32768, which is very large compared with other conventional Ising machines. This is caused by the repeated time-division of operations to reduce the number of arithmetic units and to store the weights in external RAMs. Such repeated time-division and storage of weights in external RAMs can be implemented, and we can implement approximately 100000 nodes CBM-CSAs by using these. However, using these techniques cause a significant increase in calculation time, and from the results of [4] and [21], this SQA machine could be slower than the GPU.

Moreover, the proposed circuit allows for further scaling of CBM-CSAs. The proposed circuit requires at least one DSP for each neuron. Therefore, we can implement a CBM-CSA with more than 10000 nodes by reducing the bit width of the weights and the internal state x_i to reduce the use of LUTs, BRAMs, and URAMs.

TABLE III
PARAMETERS FOR THIS WORK IN THE EXPERIMENTS

	K_{4000}	$R[-100, 100]_{4000}$
dt	1024	1024
Initial temperature	300	1000
Temperature rate	0.90	0.95
The number of epochs	15	20
The number of temperatures	30	50

TABLE IV
EXPERIMENTAL RESULTS FOR K_{4000}

	This work	FPGA-SB [4]	Optimized CPU-SA [4]	Unoptimized CPU-SA [19]
Calculation time				
Best [ms]	0.105	0.185	63.5	8320
Avg [ms]	0.110	0.211	66.0	8320
(vs. this work)	(x1)	(x2)	(x600)	(x75636)
Best solution				
Best	-93102	-	-	-97736
Avg	-92015	-	-	-97736
(vs. Unoptimized CPU-SA)	(x0.94)	(-)	(-)	(x1.00)

B. Experiments

The best solution of the implemented CBM-CSA was comparable to that of Unoptimized CPU-SA. This is caused by the simplification of the exp function to the shift function as the hardware-oriented algorithm and the implementation of CBM-CSAs to operate in discrete time by using the Euler method, although CBMs run in continuous time in the model. However, the performance degradation was insignificantly and Kawashima reported that CBM-CSA obtained solutions comparable to the optimal solution of Biq Mac [17] in [10], so we conclude that the effects of the hardware-oriented algorithm and the discretization were very insignificant. Additionally, the worsening rate of the best solution of $R[-100, 100]_{4000}$ was similar to that of K_{4000}, indicating that the bit width of the weights that this work can handle is wide and has large width of the representation. Therefore, the implemented CBM-CSAs can solve not only simple combinatorial optimization problems such as the existence of a relationship or not but also complex problems such as traveling salesman problems (TSPs).

Regarding the calculation time of K_{4000}, this work was approximately twice as fast as the conventional work [4], because of the introduction of the differential multiply-accumulation. Furthermore, we significantly reduce the calculation time of the time division calculation by using the differential multiply-accumulation. For example, we estimated the calculation time by using parameters of K_{4000}. From Fig. 12, the total epochs to reach HNN energy was 185 which means that temperatures were changed 11 times from Table III. Furthermore, the estimated calculation times of a CBM-CSA without differential multiply-accumulation (Without DMACC CBM-CSA) and the conventional CBM-CSA (Conventional CBM-CSA) [10] were 7.578 [ms] and 0.561 [ms], respectively as shown in Fig. 12. From these results of estimation, the CBM-CSA implemented in this work is 69 times and five times as fast as Without DMACC CBM-CSA and Conventional CBM-CSA, respectively. In addition, by introducing the improvements of this work, CBM-CSA becomes faster than the conventional Ising machine, FPGA-SB [4].

TABLE V
EXPERIMENTAL RESULTS FOR $R[-100, 100]_{4000}$

	This work	Unoptimized CPU-SA [19]
Calculation time		
Best [ms]	0.100	8520
Avg [ms]	0.105	8520
(vs. this work)	(x1)	(x81143)
Best solution		
Best	-5464497	-5662692
Avg	-5426268	-5662692
(vs. Unoptimized CPU-SA)	(x0.96)	(x1.00)

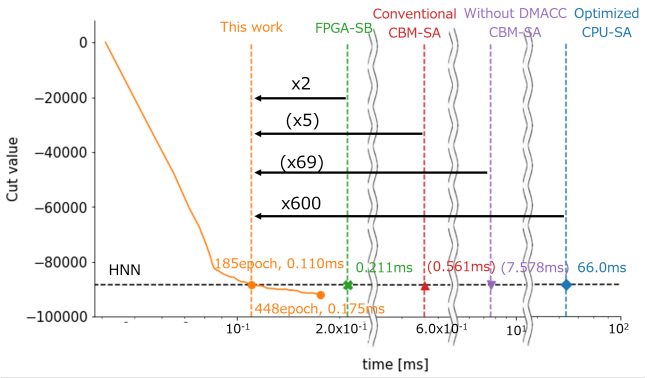

Fig. 12. Change of K_{4000} cut value
( ) means the estimated calculation time

Although the calculation time when introducing the differential multiply-accumulation depends on the problem to be solved and the number of node flips that have occurred, the hardware-oriented algorithm was very powerful and useful because the calculation time was greatly reduced, despite the calculation accuracy has not changed with the introduction of the hardware-oriented algorithm. Furthermore, as the number of nodes increased, the effect of the differential multiply-accumulation is expected to increase. Therefore, by introducing the proposed architecture when implementing a large-scale time-domain neural network such as spiking neural networks [23] and chaotic neural networks [24], we expect a significant reduction in calculation time without any side effects.

Finally, we conclude that this work implements one of the highest-performance Ising machines on a single FPGA board based on the results of the FPGA implementation and experiments and the above discussion. In addition, CBMs are not only used to solve combinatorial optimization problems but also for reservoir computing [25] [26]. The speedup and scale-up of CBMs in this work will contribute to the development of reservoir computing using CBMs.

VII. CONCLUSION

This work improved the conventional CBM-CSA [10] to make it larger and faster. Memory circuits in the conventional CBM-CSA are implemented separately for each neuron. However, this work implemented only one large memory circuit.

TABLE VI
IMPLEMENTED ISING MACHINES ON A SINGLE FPGA BOARD

	This work	Kawashima [10]	Tsukamoto [22]	Yoshimura [5]	Tatsumura [4]	Waidyasooriya [21]	Okuyama [20]
Algorithm	CSA		SA		SB	SQA	
Topology	complete graph			chimera graph	complete graph		king graph
Size	4,096	300	1,024	4,096	4,096	32,768	9,216
Precision	16	16	16	1	1	32*	8
Target board	Alveo U250	Virtex-6 FPGA ML605	Arria 10 GX 1150	Virtex-7	Arria 10 GX 1150	Arria 10 10AX	Virtex UltraScale XCVU905

Precision : default is a fixed point, * is a floating point

Therefore, the implementable scale of CBM-CSA only depends on the total amount of memories, and we can implement a larger CBM-CSA than the conventional CBM-CSA. In addition, we improved the conventional neuron circuits required to reset accumulated values to change temperatures. The reset of the accumulated value causes a significant increase in the calculation time. Furthermore, we enabled dynamic changes of inputs to the neuron circuits and inhibited the increase in the calculation time.

We implemented a CBM-CSA with 4096 nodes on an FPGA. The CBM-CSA can control 16-bit width weights and runs at 100MHz. We evaluate the proposed CBM-CSA by solving max-cut problems. The best solution of the CBM-CSA is comparable to that of the CPU-SA, but the calculation time is approximately one-fifth of the conventional CBM-CSA. Moreover, with the improvements of this work, the CBM-CSAs become 600 times as fast as a CPU-SA, and twice as fast as the conventional Ising machine on an FPGA [4]. Then, we conclude that one of the highest-performance Ising machines is implemented for a single FPGA board in this work.

ACKNOWLEDGEMENT

This work is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and JSPS KAKENHI Grant Number 20H04258 and 23H03468.

REFERENCES

[1] G. Rosenberg, P. Haghnegahdar, P. Goddard, P. Carr, K. Wu, and M. L. De Prado, "Solving the optimal trading trajectory problem using a quantum annealer," in *Proceedings of the 8th Workshop on High Performance Computational Finance*, 2015, pp. 1–7.

[2] T. Kadowaki and H. Nishimori, "Quantum annealing in the transverse ising model," *Physical Review E*, vol. 58, no. 5, p. 5355, 1998.

[3] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk *et al.*, "Quantum annealing with manufactured spins," *Nature*, vol. 473, no. 7346, pp. 194–198, 2011.

[4] K. Tatsumura, A. R. Dixon, and H. Goto, "FPGA-based simulated bifurcation machine," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019, pp. 59–66.

[5] 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," *International Journal of Networking and Computing*, vol. 7, no. 2, pp. 154–172, 2017.

[6] K. Yamamoto, K. Ando, N. Mertig, T. Takemoto, M. Yamaoka, H. Teramoto, A. Sakai, S. Takamaeda-Yamazaki, and M. Motomura, "7.3 statica: A 512-spin 0.25 m-weight full-digital annealing processor with a near-memory all-spin-updates-at-once architecture for combinatorial optimization with complete spin-spin interactions," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 138–140.

[7] T. Zhang, Q. Tao, B. Liu, and J. Han, "A review of simulation algorithms of classical ising machines for combinatorial optimization," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 1877–1881.

[8] L. Chen and K. Aihara, "Chaotic simulated annealing by a neural network model with transient chaos," *Neural networks*, vol. 8, no. 6, pp. 915–930, 1995.

[9] H. Suzuki, J.-i. Imura, Y. Horio, and K. Aihara, "Chaotic boltzmann machines," *Scientific reports*, vol. 3, no. 1, pp. 1–5, 2013.

[10] I. Kawashima, T. Morie, and H. Tamukoh, "FPGA implementation of hardware-oriented chaotic boltzmann machines," *IEEE Access*, vol. 8, pp. 204 360–204 377, 2020.

[11] I. Kawashima, Y. Katori, T. Morie, and H. Tamukoh, "An area-efficient multiply-accumulation architecture and implementations for time-domain neural processing," in *2021 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2021, pp. 1–4.

[12] J. E. Volder, "The cordic trigonometric computing technique," *IRE Transactions on electronic computers*, no. 3, pp. 330–334, 1959.

[13] M. Zaman, K. Tanahashi, and S. Tanaka, "Pyqubo: Python library for qubo creation," *IEEE Transactions on Computers*, 2021.

[14] K. Tanahashi, S. Takayanagi, T. Motohashi, and S. Tanaka, "Application of ising machines and a software development for ising machines," *Journal of the Physical Society of Japan*, vol. 88, no. 6, p. 061010, 2019.

[15] A. Xilinx, "PYNQ," https://github.com/Xilinx/PYNQ (accessed Mar.16,2022), 2018.

[16] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[17] A. Wiegele, "Biq mac library—a collection of max-cut and quadratic 0-1 programming instances of medium size," *Technical report*, 2007.

[18] S. E. Karisch, "G-set maxcut problem," https://web.stanford.edu/~yyye/yyye/Gset/ (accessed Dec.7,2022), 2003.

[19] D.-W. S. Inc., "dwave-neal," https://github.com/dwavesystems/dwave-neal (accessed Mar.16,2022), 2018.

[20] T. Okuyama, M. Hayashi, and M. Yamaoka, "An ising computer based on simulated quantum annealing by path integral monte carlo method," in *2017 IEEE international conference on rebooting computing (ICRC)*. IEEE, 2017, pp. 1–6.

[21] H. M. Waidyasooriya and M. Hariyama, "Highly-parallel FPGA accelerator for simulated quantum annealing," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, 2019.

[22] S. Tsukamoto, M. Takatsu, S. Matsubara, and H. Tamura, "An accelerator architecture for combinatorial optimization problems," *Fujitsu Science Technology Jounal*, vol. 53, no. 5, pp. 8–13, 2017.

[23] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[24] K. Aihara, T. Takabe, and M. Toyoda, "Chaotic neural networks," *Physics letters A*, vol. 144, no. 6-7, pp. 333–340, 1990.

[25] 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 *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–7.

[26] Y. Katori, H. Tamukoh, and T. Morie, "Reservoir computing based on dynamics of pseudo-billiard system in hypercube," in *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019, pp. 1–8.