International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 7, Number 2, pages 154–172, July 2017

Implementation and Evaluation of FPGA-based Annealing Processor for Ising Model by use of Resource Sharing

Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, and Masanao Yamaoka
Center for Exploratory Research, Hitachi, Ltd.,
1-280, Higashi-Koigakubo, Kokubunji-shi, Tokyo 185-8601, Japan

Received: February 24, 2017
Revised: April 26, 2017
Accepted: June 1, 2017
Communicated by Tomoaki Tsumura

Abstract

The non-von Neumann computer architecture has been widely studied to prepare us for the post-Moore era. The authors implemented this kind of architecture, which finds the lower energy state of the Ising model using circuit operations inspired by simulated annealing in SRAM-based integrated circuits. Our previous prototype was suited for the Ising model because of its simple and typical structure such as its three-dimensional lattice topology, but it could not be used in real world applications. A reconfigurable prototyping environment is needed to develop the architecture and to make it suitable for applications.

Here, we describe an FPGA-based prototyping environment to develop the annealing processor's architecture for the Ising model. We implemented the new architecture using a prototyping environment. The new architecture performs approximated simulated annealing for the Ising model, and it supports a highly complex topology. It consists of units having fully-connected multiple spins. Multiple units are placed in a two-dimensional lattice topology, and neighboring units are connected to perform interactions between spins. The number of logic elements was reduced by sharing the operator among multiple spins within the unit. Furthermore, a pseudo-random number generator, which produces random pulse sequences for annealing, is also shared among all the units. As a result, the number of logic elements was reduced to less than 1/10, and the solution accuracy became comparable to that of a conventional computer's simulated annealing.

Keywords: FPGA, Ising model, annealing, sharing, random pulse

1 Introduction

Improvements to the performance of conventional computers have mainly been achieved through semiconductor scaling; however, scaling is reaching its limit [15]. As we approach the end of Moore's law, many non-von Neumann architectures have been proposed. Some architectures rely on emerging devices such as superconductors to enable the quantum superposition [8], while others are still based on silicon integrated circuits [3, 12]. We previously proposed an SRAM-based integrated circuit to perform a lower-energy-state search of the Ising model [7, 18–21]. Our architecture aims to solve combinatorial optimization problems to operate social infrastructure efficiently.

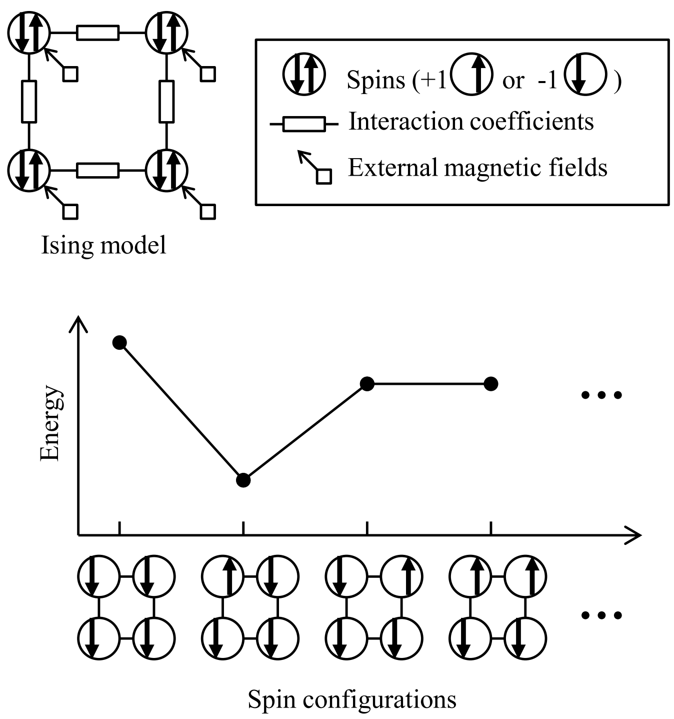

Figure 1: Ising model and its energy landscape

The Ising model can express the behavior of magnetic spins, and it consists of n binary spins $\{\sigma_1, \ldots, \sigma_n\}$, interactions $\{J_{ij}\}$ between spins, and external magnetic fields $\{h_i\}$ as shown in Figure 1 [4]. The Ising model's energy is defined as

$$H(\sigma_1, \cdots, \sigma_n) = -\sum_{i<j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i. \tag{1}$$

The energy varies with the spin configurations, and it can be visualized as an energy landscape, as shown in Figure 1. The combinatorial optimization problems can be mapped into the Ising model [10]. The Ising models' energy corresponds to the objective function of combinatorial optimization problems. This means that a spin configuration having the lowest energy represents the global optimum solution of the original problem.

Specific purpose computers, which solve the Ising model, have been proposed. Some computers are based on natural phenomena such as the quantum superposition [8] or optical parametric oscillation [17]. An FPGA-based prototype has been proposed to implement new architectures for solving it [6]. In addition, several FPGA-based specific purpose computers have been proposed to accelerate Monte Carlo simulations of the Ising model [2,5,9]. Their objective is to research physics further, not to solve optimization problems.

Searching the lowest-energy state of the Ising model is essentially equivalent to the weighted maximum cut problem in graph theory. The maximum cut problem is an archetype of the non-deterministic polynomial time hard (NP-hard) problem, as is a lowest-energy state search of the Ising model [1]. Finding the global optimum solution to the NP-hard problem generally requires exponential time. Therefore, a near-optimum solution is used in practical use cases.

Our previous prototype had a structure that was too simple to be used practically, and it could not be applied to real applications. A more complex structure is needed through developing applications. The requirement for the topology and dynamic range varies according to the application. A well-suited architecture needs to be developed for each application. Therefore, we developed an FPGA-based programmable prototyping environment to emulate the various configurations of our architecture.

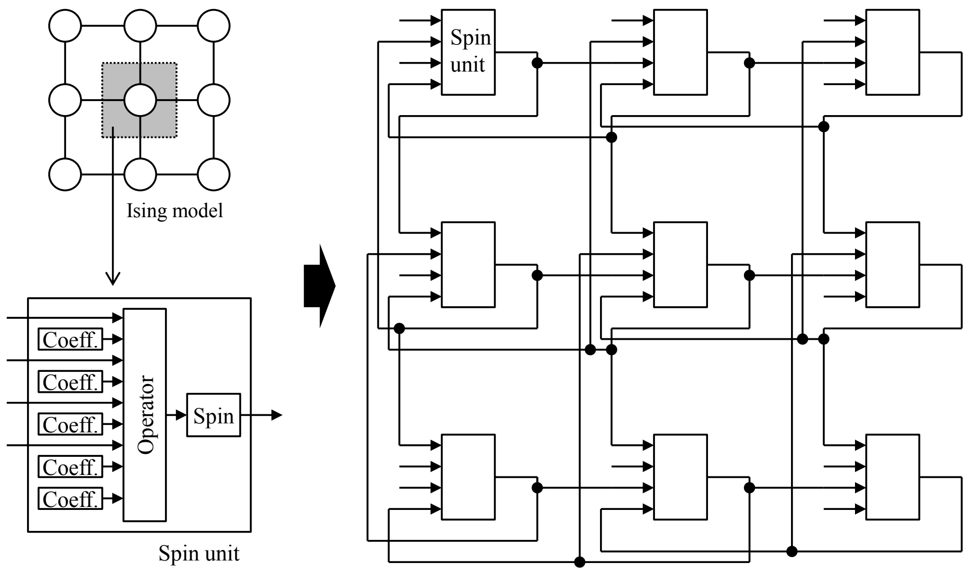

Figure 2: Basic idea of our architecture

This paper is organized as follows. Section 2 describes our architecture's basic concept. Section 3 presents a operator circuit to determine the next state of spin. Section 4 describes the spin unit's architecture to reduce the amount of resource usage by sharing the operator among multiple spins. Section 5 also describes a technique to reduce resource consumption on a pseudo-random number generator. Section 6 shows the implementation results and performance evaluation. We describe related studies in Section 7 and conclude this paper in Section 8.

2 Overview of Annealing Processor

Our architecture uses an electronic circuit mimicking the Ising model shown in Figure 2. The spin and accompanying coefficients are grouped into a spin unit. Each spin unit has a memory cell array to represent the spin and coefficients. The next state of spin is determined by a digital or analog operator. The spin units are connected together according to the topology of the desired Ising model.

This processor is used as an accelerator for the combinatorial optimization problem. Coefficient values representing the optimization problem are written in memory cells to solve a problem using the processor. Thus, the interaction between spin units is executed repeatedly by updating the spin value. The operator takes neighboring spin values and corresponding coefficients for determining the next state of spin to minimize the energy locally. An update means that the latest output of the operator is stored in the memory cell representing the spin, and then the output can be observed using neighboring spin units.

Conflict with the update, meaning that the spin unit uses the spin value updated simultaneously, must be avoided. To prevent this conflict from occurring, the updating sequence is controlled during interactions. In the case of Figure 2, the spin units can be grouped in a checkerboard pattern, and the groups are updated sequentially.

The interaction, which is achieved by updating the spins, acts as the steepest descent method for reaching the local optimum state. This behavior can find a state that has energy lower than that of the current state. However, improving the solution is impossible once the state falls into a local optimum. This means it transits to the same state forever. Therefore, probabilistic behavior is added to the operator to enable escaping from the local optimum. Pseudo-random number generators are needed to achieve probabilistic behavior.

In our previous prototype, the topology, coefficient width, and operator were too simple, and it was not applicable to real world optimization problems. The topology of the supported Ising model

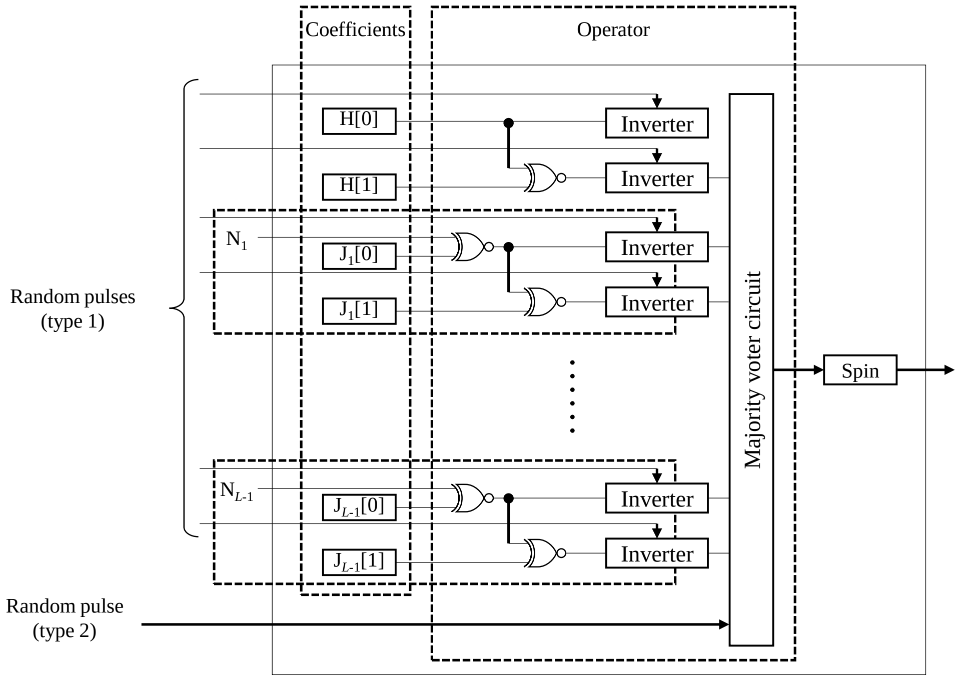

Figure 3: Circuit diagram of the operator

in our previous prototype was a three-dimensional lattice, and the dynamic range of the interaction coefficient was limited to 2 bits. The operator also needed to be improved to increase the solution's accuracy.

3 Glauber Dynamics Operator

This section describes operator's circuit, which determines the next spin state. The circuit diagram is shown in Figure 3. The proposed operator acts in accordance with Glauber dynamics (also known as the heat bath algorithm). The Glauber dynamics plays an important role in the optimization. The details of the theoretical background were published in reference [13].

The operator takes the value of adjacent spins shown as N_k in Figure 3. The value of spin $\{-1, +1\}$ is represented as that of binary digit $\{0, 1\}$. Memory cells $H[0]$ and $H[1]$ represent an external magnetic field coefficient $h_i \in \{\pm 1, 0\}$. They are defined as

$$H[0] = \begin{cases} 1 & (h_i \geq 0) \\ 0 & (h_i < 0) \end{cases}, \quad H[1] = |h_i| . \tag{2}$$

Similarly, memory cells $J_k[0]$ and $J_k[1]$ represent the interaction coefficient between the spin and its k-th adjacent spin and are defined in the same way. Each H_k and J_k has 2-bit width, and the index shows the bit field of the memory cells.

The acceptance rate of the Glauber dynamics is expressed as

$$P_{\text{Glauber}}(\Delta H) = \frac{1}{2}\left\{1 + \tanh\left(-\frac{\Delta H}{2T}\right)\right\}, \tag{3}$$

where ΔH is the energy change upon a spin flip, and T is the temperature. Random pulse sequences are controlled to satisfy Equation (3) to achieve the Glauber dynamics using the circuits shown in Figure 3.

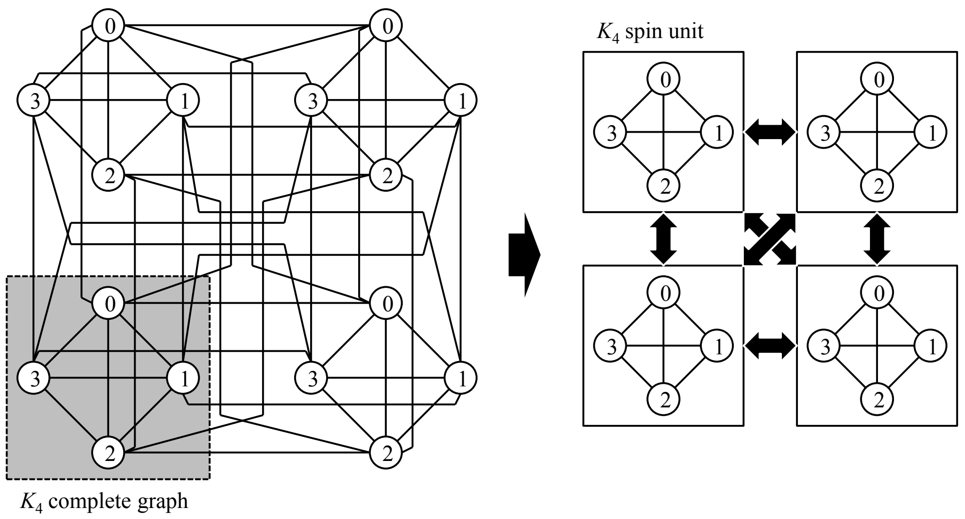

Figure 4: Topology of Ising model and corresponding hardware organization

A random pulse sequence that takes value "1" with probability p_{rnd} is injected to each random pulse signal of type 1. Moreover, a signal of type 2 receives a random string that takes value "1" with a probability of 50%. Each has independent randomness. Inverters probabilistically output the signals flipped by the random pulse signals. Let X be a random variable that takes the value "1" with probability of p_{rnd}, and let value "0" with probability $1 - p_{\mathrm{rnd}}$ and Y be the number of the majority voter circuit's inputs with value "1." The central limit theorem suggests that variable Y will be approximately normally distributed. By using a simple approximation for hyperbolic tangent [14, 16], we obtain the relationship between probability p_{rnd} and temperature T as

$$p_{\mathrm{rnd}} = \frac{1}{2}\left(1 - \frac{1}{\sqrt{1 + \frac{4T^2}{\pi L}}}\right), \tag{4}$$

where L is the number of spin coefficients. Temperature T determines the probability of accepting the state transition. At temperature zero, it behaves as the steepest descent method. Usually, temperature T is scheduled as

$$T_{i+1} = \beta T_i, \tag{5}$$

where β is cooling coefficient ($0 < \beta < 1$), and where T_i is the temperature in the i-th time step. The temperature exponentially decreases from given initial temperature T_0.

Equation (4) depends only on temperature T and the number of coefficients L. L is determined using the topology of the Ising model, and temperature T is updated every time step during the annealing. p_{rnd} can be shared among all the spins.

4 Shared Spin Unit

The left side of Figure 4 shows the desired complex topology. It consists of a local tight connection and global sparse connection. The local tight connection can be represented as a complete graph (e.g., K_4 complete graph in the figure). These complete graphs are placed in a grid pattern. The vertices in the same position in each of the neighborhood complete graphs are connected. The neighborhood is defined as being in eight directions in the plane: north, south, east, west, north-east, north-west, south-east, and south-west.

The complete graph is implemented in the aforementioned topology as a single spin unit, containing multiple spins. The spin units are placed in a grid pattern, and the units are connected

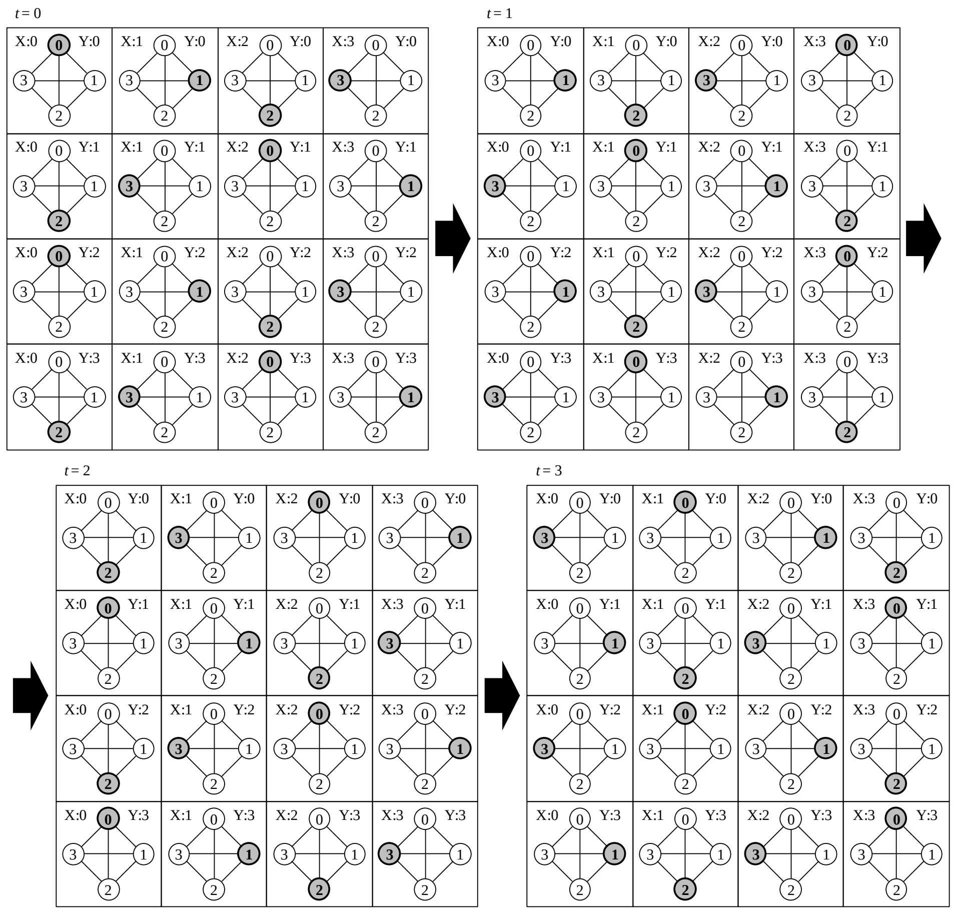

Figure 5: Updating sequence for spin units

with the neighborhood shown on the right side of Figure 4. Basically, each spin should be updated sequentially. Spins that are not directly connected may be updated simultaneously, and their result is the same as that of a sequential update. If the topology is a lattice like that shown in Figure 2, the spins that may be updated simultaneously can be organized into two groups. In the case of Figure 4, they can be organized into four groups. No spins may be updated simultaneously in a spin unit because the topology within them is a complete graph. However, the spins in different spin units can be updated simultaneously, as shown in Figure 5.

A circuit diagram of spin units is shown in Figure 6. According to the aforementioned updating rule, only one operator is sufficient in a spin unit because only one spin is updated at a time. The spin unit has two memories (spin memory and coefficient memory) representing multiple spins, and one operator is shared among multiple spins. The operator determines the next state of one spin at a time, and it is used to determine all spins repeatedly by spending multiple cycles.

The spin unit's temporal behavior is shown as a timing chart in Figure 7. The counter acts as a simple controller to decide the updating sequence to maintain the updating rule. The value of the counter is used as the address of memories indicating the spin that is to be updated at the current cycle. When an interaction begins, the value of the counters is initialized as the appropriate number shown in $t = 0$ of Figure 5. During the interaction, the counter is incremented when the spin is updated. If the number of spins in the spin unit is an exponent of 2, the counter can wrap around naturally to continue control.

Spin memory, which holds the state of multiple spins, is implemented as an array of flip-flops because all spins must be readable at all times to provide the spin value for neighboring spin units. In contrast, coefficient memory is sufficient with 1 R/W memory. During the interaction, only one

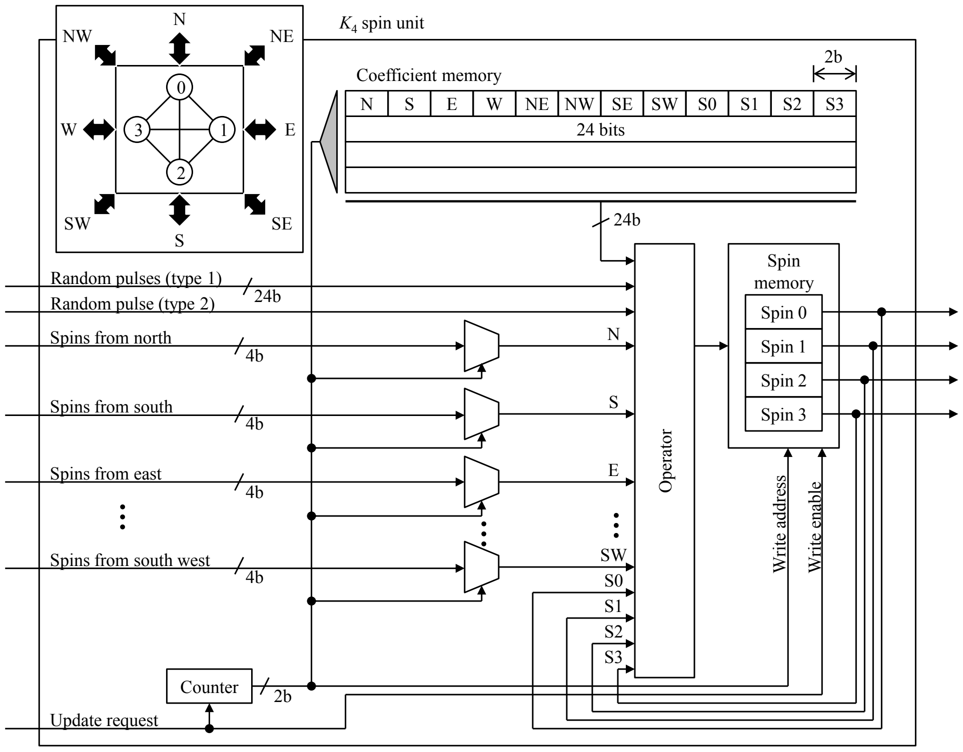

Figure 6: Circuit diagram of spin unit

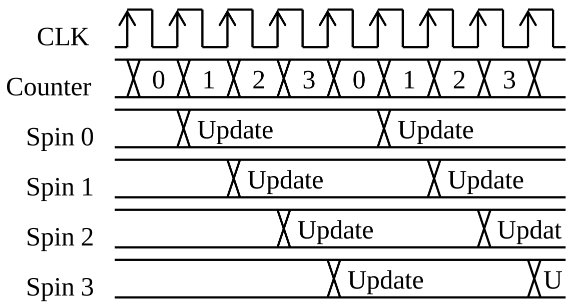

Figure 7: Timing chart of spin unit

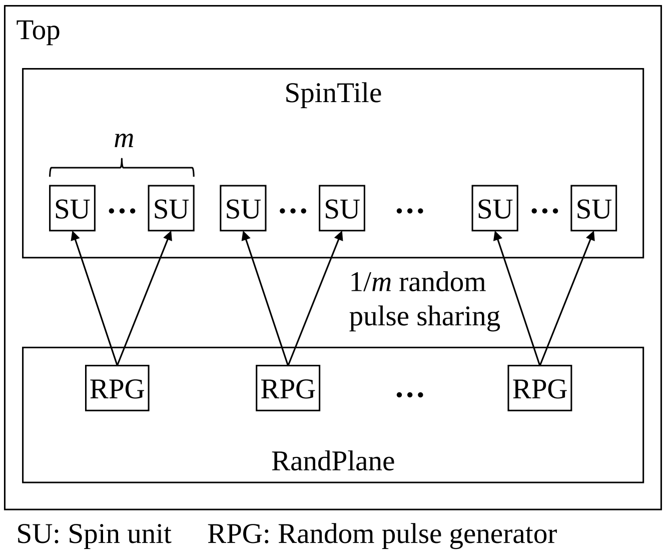

Figure 8: Concept of random pulse sharing

line of the coefficient memory is read out in each cycle. Writing to the coefficient memory is done during the initialization phase before starting the interaction.

The operator needs the neighboring spin values and random pulses to determine the next spin state. To generate random pulses according to p_{rnd}, one pseudo-random number generator and one comparator are needed to generate one random pulse sequence. Thus, one spin unit requires many pseudo-random number generators.

5 Random Pulse Sharing

5.1 Concept of random pulse sharing

We were concerned that the pseudo-random number generators would become dominant in resource consumption and that they would cap the scalability of our architecture. In our previous prototype, we added simple hardware for each spin unit instead of one-by-one random pulse generators [7]. The hardware mixes the few random pulse sequences to generate new random pulse sequences for each spin unit. The processor has only two random pulse generators as a source for mixing the pulses.

In this study, we created a simple method for sharing the random pulse generators among multiple spin units, as shown in Figure 8. The multiple spin units, which are shown on the right side of Figure 4, are organized as a SpinTile in Figure 8. All spin units in the SpinTile have input wires for random pulses. RandPlane is a collection of random pulse generators that transmit a bunch of random pulse sequences. Figure 9 shows the variation in the random pulse sharing in 256 spin units (16×16 spin units). The number below each configuration shows the ratio between the spin units and generator. Basically, the random pulse generator is placed one-by-one along with the spin unit (ratio 1). Ratio $1/m$ means that m spin units share random pulse generators.

According to the architecture's updating rules, a couple of spins, which are connected directly, are not updated at a time. In other words, the same pulse is not used among directly connected spins simultaneously even if all spin units share one random pulse generator. We think this is why multiple units can share the random pulse sequence. However, this method has not been validated mathematically, and it remains as future work. This paper presents an experimental evaluation of our method using an FPGA-based prototype.

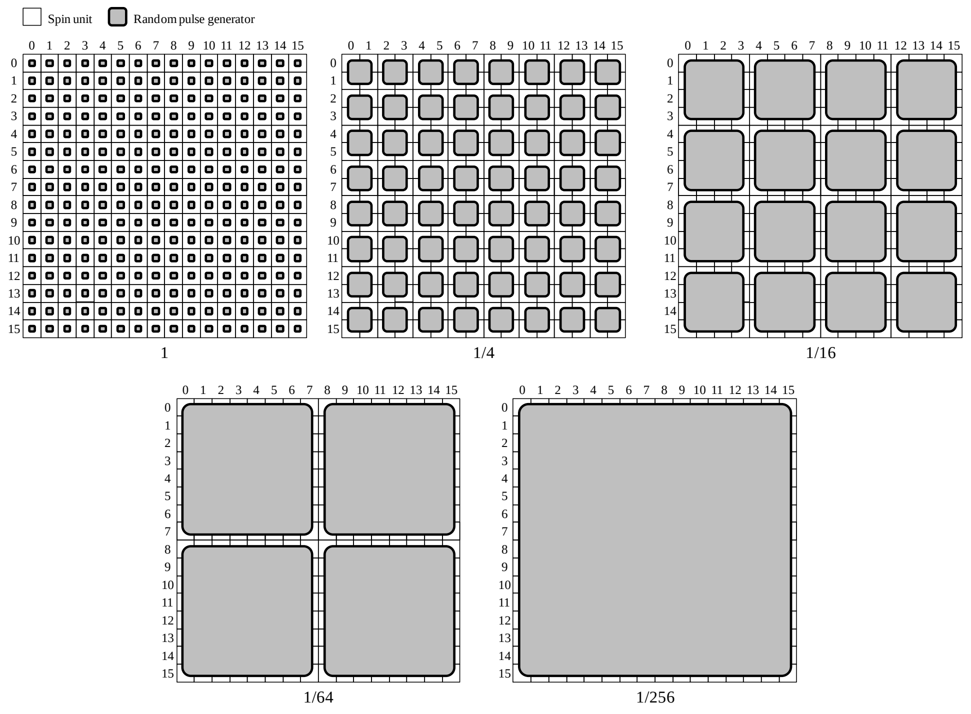

Figure 9: Variation in random pulse sharing

5.2 Random pulse generator

In this design, a random pulse generator (shown as a gray box in Figure 9) contains 25 pseudo-random number generators and 25 comparators, as shown in Figure 10. Each pseudo-random number generator is xorshift32 [11]. The comparator analyzes the 32-bit threshold and its random number. The comparator's output is "1" when the random number is greater than the threshold. Otherwise, it is "0." As a result, a random pulse sequence is generated, and it takes value "1" with a probability according to the threshold.

The bit width of the pseudo-random number generator and comparator determines the resolution of the temperature. The relationship between the resolution of temperature and the quality of the solution should be studied. In this prototype, 32-bit resolution is used to express accurately the behavior at low temperature. The processor has one common scheduler for all random pulse generators. It calculates $p_{\rm rnd}$ and converts the value into the threshold. Both $p_{\rm rnd}$ and the threshold are updated continuously during the interaction.

5.3 Long wiring for random pulse sharing

Long wires might be needed to share the random number generator among spin units, and they may cause the operating frequency to decrease. Making the pipeline by adding flip-flops into the long wires can maintain the operating frequency. The scheduler works in advance in accordance with the number of stages in the pipeline. A tree structure is used to spread the random pulses widely to the entire SpinTile, as shown in Figure 11. In our implementation discussed later, the pipeline is not needed in 10 MHz operation. However, we use a 6-depth pipeline for 100 MHz operation to satisfy the timing constraint.

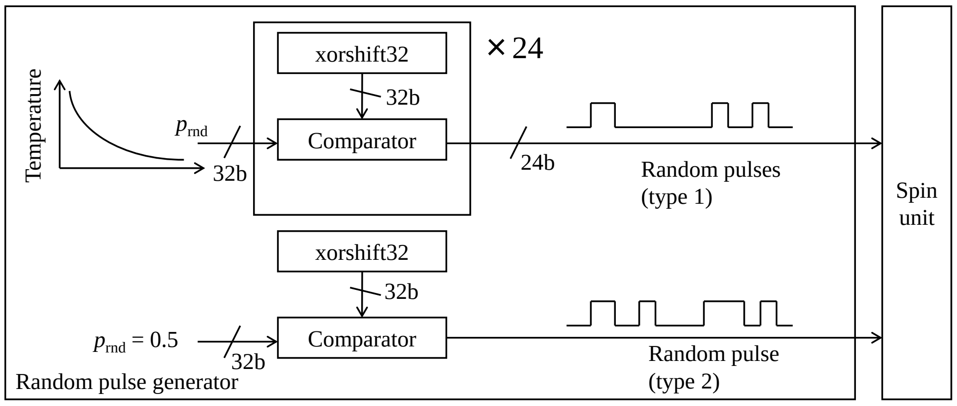

Figure 10: Block diagram of random pulse generator

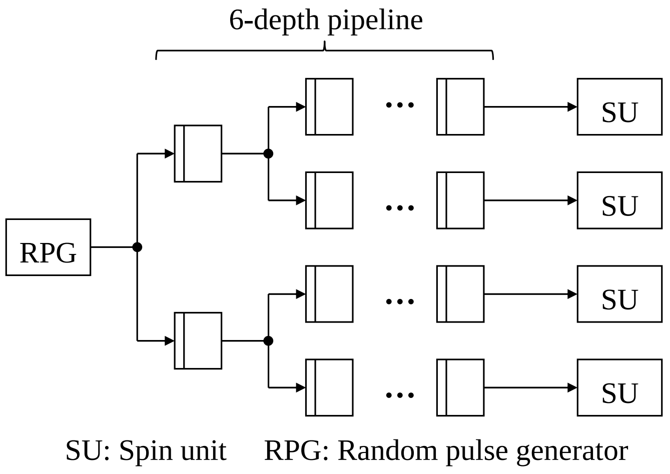

Figure 11: Block diagram of long wiring of random pulse

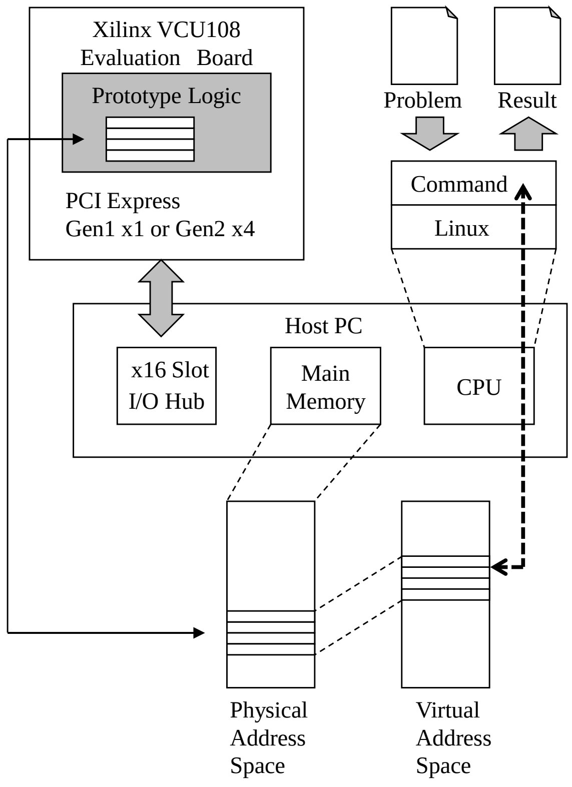

Figure 12: Block diagram of prototyping environment

6 Implementation and Evaluation

6.1 Prototyping Environment

The block diagram of our prototyping environment is shown in Figure 12, and a photograph of it is shown in Figure 13. We implemented a prototyping environment consisting of a Xilinx VCU108 evaluation board (FPGA: Virtex UltraScale XCVU095) and a PC used as the FPGA board's host. Our architecture was implemented in Bluespec SystemVerilog, and Vivado Design Suite 2016.1 was used to synthesize the RTL generated by Bluespec. The implementation of our architecture runs at 100 MHz. All spin units update a spin for each clock cycle. Therefore, all spins are updated once every four cycles.

All storage in the FPGA, representing the spins, coefficients, and annealing parameters, was accessed using the PCI Express interface. A command, running on the host PC, wrote the coefficients and parameters into the registers to store the problem to be solved, and the command read the spins from the registers to load the solution. In an earlier stage of this prototype's development, the data between the PC and FPGA were transferred using a memory-mapped I/O. For this reason, the effective performance of the data transfer was significantly lower (Write to FPGA: 9 MB/s, Read from FPGA: 2 MB/s) than the peak performance (250 MB/s @ PCI Express Gen1 x1). We implemented the DMA transfer to enhance the data transfer performance, as shown in Figure 14 and 15. The DMA version of our implementation uses PCI Express Gen2 x4 (Peak performance: 2 GB/s). Each plot shows the average bandwidth for 10,000 trials of DMA transfer. For both DMA write and read, the bandwidth saturated with a transfer size of 256 KB or more. The efficiency was about 70% (write) and 60% (read) in saturated area.

The design flow of our prototyping environment is shown in Figure 16. We implemented the parameter-independent design of our architecture using Bluespec System Verilog. The number of spins in the spin unit, number of spin units in the X direction and Y direction on the SpinTile, and

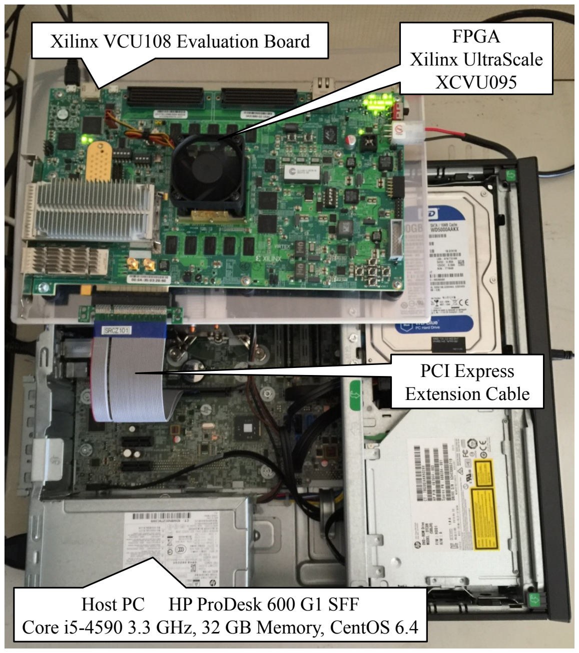
Xilinx VCU108 Evaluation Board
FPGA
Xilinx UltraScale
XCVU095
PCI Express
Extension Cable
Host PC HP ProDesk 600 G1 SFF
Core i5-4590 3.3 GHz, 32 GB Memory, CentOS 6.4

Figure 13: Photograph of prototyping environment

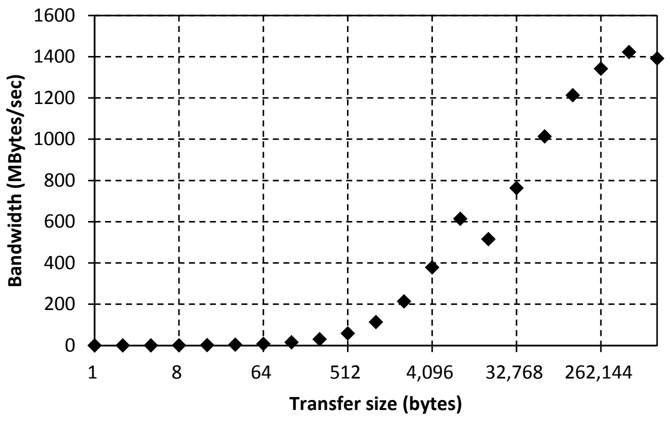

Figure 14: Data transfer performance of prototyping environment (DMA Write)

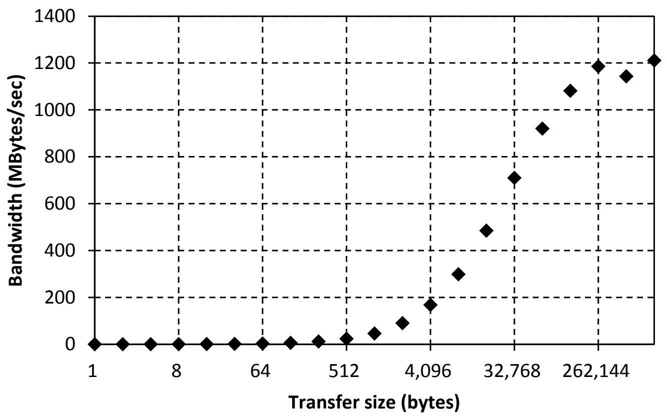

Figure 15: Data transfer performance of prototyping environment (DMA Read)

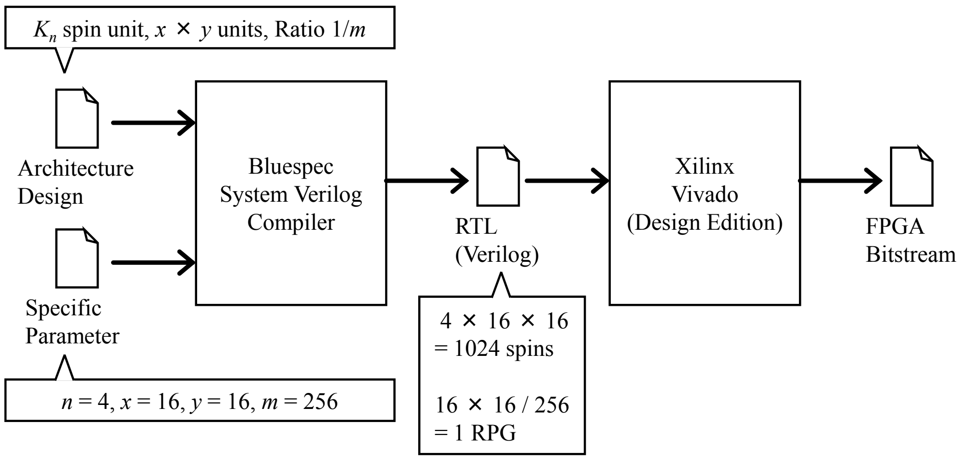
K_n spin unit, $x \times y$ units, Ratio $1/m$
Architecture Design
Specific Parameter
$n = 4, x = 16, y = 16, m = 256$
Bluespec System Verilog Compiler
RTL (Verilog)
$4 \times 16 \times 16$ = 1024 spins
$16 \times 16 / 256$ = 1 RPG
Xilinx Vivado (Design Edition)
FPGA Bitstream

Figure 16: Design flow of prototyping environment

ratio between the spin units and random pulse generators can be set as parameters written in the separated file. The design space can be explored automatically by modifying the parameter file and by recompiling.

6.2 Evaluation of 1024-Spin Implementation

The resource usage of our architecture is shown in Table 1 and Figure 17. The implemented design supports 1024 spins ($16 \times 16 \times 4$) and 2-bit coefficients ($-1, 0, +1$). The look-up table (LUT) is the dominant component in our design's resource usage. For example, the consumption of flip-flops is only 22.10% at maximum. We evaluated the resource consumption by changing the sharing ratio of the random pulse generator. The table heading indicates the ratio between the spin units and random pulse generators (e.g., $1/m$ means that m spin units share one random pulse generator). The top hierarchy shows the total resource consumption, including those for PCI Express. The core hierarchy indicates our architecture's resource consumption, and it contains SpinTile and RandPlane. SpinTile aggregates the spin units, and it contains 256 ($= 16 \times 16$) spin units in every case. Thus, the resource consumption of SpinTile was almost the same in every case. RandPlane contains the random pulse generators and its controller. It also contains the wires to distribute the random pulses to multiple spin units if the random pulse generator is shared. The implementation results showed that the LUT consumption can be reduced dramatically by using random pulse sharing because RandPlane dominated the resource consumption. Although almost all resources of the FPGA were consumed at a ratio of 1, even more spins (4096 spins) could be put into the same FPGA at a ratio of 1/256.

The number of LUTs in the RandPlane could be linearly approximated with the number of random pulse generators as $1673 \times (256/m) + 10,664$. The LUT consumption for one random pulse generators could be estimated as being about 1673 LUTs. Thus, more than 10,000 LUTs were used for the controller in the RandPlane. It has a function to calculate a threshold based on a scheduled temperature and a function to manage parameters for calculating temperatures. In addition, it has a state machine that manages the progress of the schedule and is responsible for part of the overall control of the Ising chip. The circuit scale is large because Equation (4) includes the reciprocal square root. Block RAM can be used as a table of thresholds to reduce the controller's resource consumption. In the current implementation, block RAM (1 MB) was implemented as a communication buffer with the host. Even if the values of spins and coefficients are stored, most of the block RAM area remains. Therefore, a threshold is stored there. Instead of calculating the threshold, the controller uses a previously prepared threshold. Because of these improvements, we anticipate that the logical amount consumed in controllers can be reduced.

The solution accuracy of these shown implementations and reference code are shown in Table 2.

Table 1: Number of LUT consumptions in 1024-spin implementation of proposed architecture

	1	1/4	1/16	1/64	1/256
Available	537,600 (100%)	537,600 (100%)	537,600 (100%)	537,600 (100%)	537,600 (100%)
Top	468,183 (87.09%)	146,506 (27.25%)	66,435 (12.36%)	46,447 (8.64%)	41,618 (7.74%)
Core	463,101 (86.14%)	141,424 (26.31%)	60,806 (11.31%)	40,818 (7.59%)	36,064 (6.71%)
SpinTile	22,947 (4.27%)	22,942 (4.27%)	22,952 (4.27%)	22,490 (4.18%)	22,942 (4.27%)
SpinUnit	91 (0.02%)	91 (0.02%)	91 (0.02%)	91 (0.02%)	91 (0.02%)
RandPlane	438,982 (81.66%)	117,408 (21.84%)	37,353 (6.95%)	17,390 (3.23%)	12,612 (2.35%)

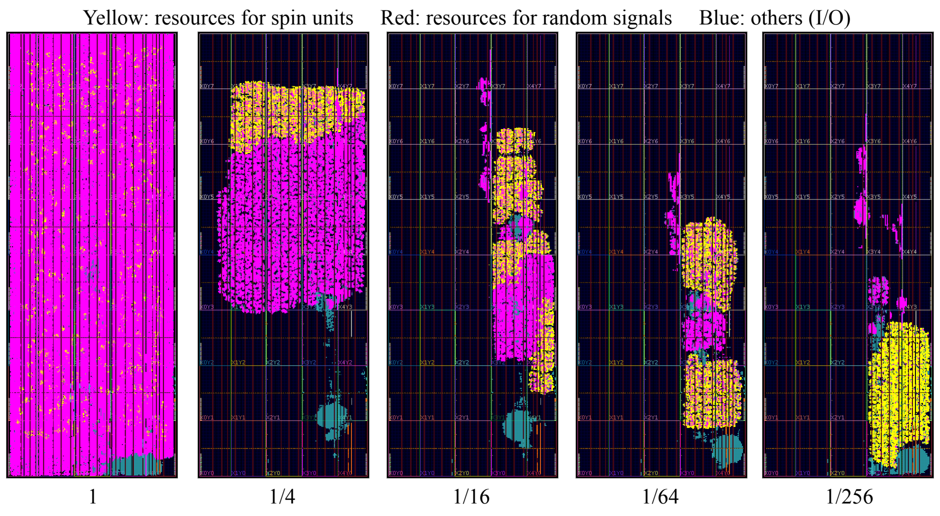
Yellow: resources for spin units
Red: resources for random signals
Blue: others (I/O)
1
1/4
1/16
1/64
1/256

Figure 17: FPGA implementation results of our architecture (1024 spins)

Table 2: Solution accuracy in 1024-spin implementation of our architecture

	1	1/4	1/16	1/64	1/256
Problem 1	-0.10%	0.26%	-0.05%	-0.05%	0.05%
Problem 2	1.26%	0.96%	1.26%	1.16%	1.06%
Problem 3	-1.26%	-0.96%	-1.37%	-1.21%	-1.21%
Problem 4	0.00%	0.05%	0.10%	-0.15%	-0.15%
Problem 5	-0.57%	-0.26%	-0.41%	-0.52%	-0.41%
Problem 6	2.48%	2.22%	2.01%	2.27%	2.27%
Problem 7	1.78%	2.35%	2.04%	2.09%	1.78%
Problem 8	0.49%	0.34%	0.44%	0.28%	0.44%
Problem 9	2.87%	2.71%	2.56%	2.66%	2.87%
Problem 10	1.34%	1.34%	1.55%	1.50%	1.65%

Ten benchmark problems, which were generated, were used for an evaluation. The category of the problems was a ground-state search of the Ising model. These problems were essentially equivalent to the weighted maximum cut problem in graph theory. The randomly generated problems had the topology shown in the left side of Figure 4. The coefficients were randomly selected from three possible coefficients $(-1, 0, +1)$. In this experiment, the prototype solved each benchmark problem ten times, and the best solution was chosen. Because annealing relies on randomness, the quality of the solution varied in each trial. The quality was measured by comparing the solution from the reference code. We define the solution accuracy as a metric for comparison by

$$R(\boldsymbol{s}) = \frac{H(\boldsymbol{s})}{H(\boldsymbol{s}_{\mathrm{ref}})} - 1 \tag{6}$$

where $\boldsymbol{s}$ is the solution derived from the method for comparison, and $\boldsymbol{s}_{\mathrm{ref}}$ is the solution to the same problem derived from the reference code. The reference code was a straightforward software-based implementation of simulated annealing; it also solved each problem ten times, and the best solution was chosen. The best solution of the reference code was used as reference energy. The quality of the solution could be measured using an energy function defined as Equation (1). $H(\boldsymbol{s})$ is the energy from our prototype implementation, and $H(\boldsymbol{s}_{\mathrm{ref}})$ is the energy from the reference code. Lower energy shows that the state of the Ising model corresponds to a better solution of the optimization problem. Therefore, both our implementation and reference code used the minimum values of the energy. $R(\boldsymbol{s})$ means the increases or decreases relative to the reference energy, and it is listed in Table 2. The large $R(\boldsymbol{s})$ value showed that our implementation's solution was better than the reference code's one.

In both our implementation and reference code, all spins were updated 250,000 times during an execution. Our implementation spent approximately 10 ms (1,000,000 clock cycles).

As a result, the solution accuracy of the proposed architecture was almost the same among the variations of the random pulse sharing. Thus, random pulse sharing could reduce resource consumption without degrading the solution accuracy. Additionally, the solution accuracy is comparable to the reference simulated annealing code. This result experimentally demonstrated that our architecture has the potential to be a special-purpose processor for annealing the Ising model.

6.3 Evaluation of 4096-Spin Implementation

Without random pulse sharing, as shown in Table 1, the limitation of integration was 1024 spins with 87.09% LUT consumption. LUT consumption could be reduced to less than 1/10 using it. Therefore, 10-times higher integration was expected using our method. Next, a 4096-spin prototype was implemented and evaluated to show the scalability of random pulse sharing.

The resource usage of the 4096-spin implementation is shown in Table 3 and Figure 18. The implemented design supports 4096 spins ($32 \times 32 \times 4$) and 2-bit coefficients $(-1, 0, +1)$. We tried

Table 3: Number of LUT consumptions in 4096-spin implementation of our architecture

	1/256 (did not meet the timing)	1/1024
Available	537600 (100%)	537600 (100%)
Top	154352 (28.71%)	149393 (27.79%)
Core	141967 (26.41%)	136998 (25.48%)
SpinTile	122942 (22.87%)	122965 (22.87%)
SpinUnit	120 (0.02%)	119 (0.02%)
RandPlane	17130 (3.19%)	12152 (2.26%)

it using a design with low resource usage that had only one random pulse generator placed. In the design, one random pulse generator was placed for every 1024 spin units (ratio of 1/1024). The implementation with a 1/1024 ratio met the timing constraints and ran correctly. The solution accuracy is shown in Table 4 and discussed later.

A more complex design, in which the ratio between the number of spin units and random pulse generations was increased to 1/256, was also implemented. The implementation with ratio 1/256 had sufficiently low LUT consumption; however, the timing constraints could not be satisfied in the I/O wiring. We estimated the timing violation was caused by some long wirings occurring between the block RAMs and spin units (worst negative slack: −0.213ns, total negative slack: −0.283ns). Block RAMs were used as the buffer of data transfer. This is not a critical problem of our architecture; hence, we will continue to improve the detailed design to achieve a more complex one.

Table 4 shows the solution accuracy of the 4096-spin implementation with the ratio 1/1024. The method of evaluation was the same as that with 1024-spin implementation. We prepared ten randomly generated problems for benchmarking, with the results shown in Table 4. The number of repetitions to solve the problem was set to 1, 2, 5, 10, 20, 50, and 100 for each problem to clarify the relationship between the number of repetitions and the quality of the solution. Each column in the first line of Table 4 shows the number of repetitions. The best solution among multiple repetitions was chosen from the results generated by the prototype. For example, 20 means the described solution accuracy was the best solution among 20 trials. Each item in Table 4 is a value calculated by Equation (6). In all items, the energy of the reference code $H(\boldsymbol{s}_{\text{ref}})$ was the best solution among 100 trials of the reference code. However, the energy of our implementation $H(\boldsymbol{s})$ was the best solution among i trials, where i is the number of repetitions.

Table 5 shows the worst solution accuracy with 100 randomly generated problems. Even in the worst case shown in Table 5, the solution accuracy improved as the number of repetitions of the trials increased. Under the same temperature scheduling, the hardware implementation could perform the annealing even when the number of spins increased because the hardware naturally exploits the parallelism in the Ising model.

7 Related Work

FPGA-based specific purpose computers for the Ising model have been proposed. These computers can be categorized into two groups based on their objective. One is for a Monte Carlo simulation of the spin system [2, 5, 9]. The other is for a ground-state search of the Ising model [6].

Gilman et al. [5] and Lin et al. [9] implemented pipelined architectures on a single FPGA that

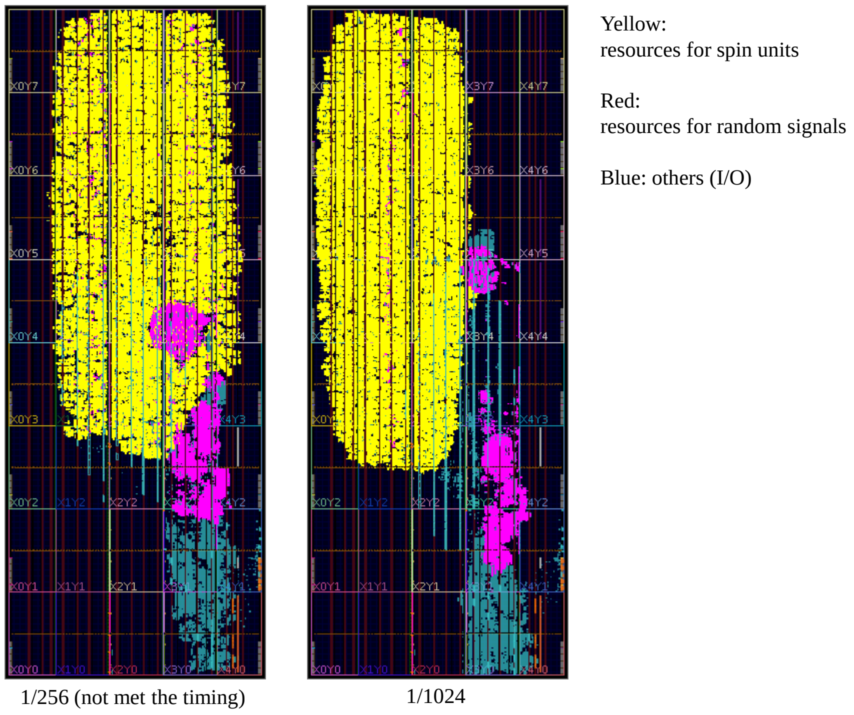

Figure 18: FPGA implementation results of our architecture (4096 spins)

Table 4: Solution accuracy in 4096-spin implementation of our architecture

	1	2	5	10	20	50	100
Problem 1	0.05%	0.05%	0.05%	0.05%	0.05%	0.08%	0.08%
Problem 2	-0.06%	-0.06%	-0.06%	-0.06%	-0.01%	0.14%	0.14%
Problem 3	-0.27%	-0.06%	0.10%	0.10%	0.15%	0.30%	0.42%
Problem 4	-0.60%	-0.44%	-0.28%	-0.19%	-0.19%	-0.19%	-0.10%
Problem 5	-0.65%	-0.39%	-0.39%	-0.19%	-0.19%	-0.19%	-0.19%
Problem 6	-0.35%	-0.35%	-0.19%	-0.19%	-0.19%	-0.06%	-0.06%
Problem 7	-0.19%	0.13%	0.13%	0.25%	0.29%	0.31%	0.40%
Problem 8	-0.20%	-0.20%	0.08%	0.08%	0.08%	0.08%	0.11%
Problem 9	-0.54%	-0.51%	0.00%	0.00%	0.00%	0.00%	0.09%
Problem 10	-0.44%	-0.44%	-0.44%	-0.29%	-0.19%	-0.04%	-0.04%

Table 5: Worst solution accuracy among 100 randomly generated problems in 4096-spin implementation of our architecture

	1	2	5	10	20	50	100
Worst	-0.92%	-0.60%	-0.53%	-0.50%	-0.42%	-0.42%	-0.35%

integrates 256^3 (three-dimensional lattice) and 1024^2 (two-dimensional lattice) spins, respectively. They used an external memory chip to store the value of spins. Coefficient values of the Ising model were assumed to be fixed. This is a critical difference between the simulation purpose and optimization purpose because this assumption is not applicable for solving optimization problems via the Ising model. For optimization, the original problem has to be represented using a combination of coefficient values, and it cannot be fixed.

Belletti et al. [2] developed Janus, which was designed for larger systems. The entire system consists of multiple compute cores and hosts. Each core has a 4×4 two-dimensional grid of processing elements and an I/O processor. This system is flexible in terms of the Ising model's topology and behavior of updating rules. It offers higher flexibility and scalability in exchange for requiring more hardware resources.

Gyoten et al. [6] implemented an FPGA-based specific processor that integrates 8×100 spins in a two-dimensional torus topology with coefficient $(+1, -1)$ for solving optimization problems. Their architecture is based on the unit (isingcell) to represent a single spin. The processor is a collection of many units connected together according to a given topology. This concept is the same as our architecture's; however, the main difference is that the unit contains multiple spins to share the computation resources.

8 Conclusion

In this paper, we presented a FPGA-based prototyping environment for an annealing processor and a new architecture for solving the Ising model. Compared with our previous prototype, both the solution accuracy and the complexity of the topology were improved. The new architecture is based on units having multiple spins, and the spins share one operator to reduce the resource consumption. Furthermore, a pseudo-random number generator, which generates random pulse sequences for annealing, is also shared among all the units. As a result, the number of logic elements was reduced to less than 1/10, and the solution accuracy became comparable to the simulated annealing running on a conventional computer.

However, a problem still remains regarding practical applications, and solving it is our future work. The dynamic range of the coefficient in our architecture is 2 bits wide, and it represents the value of +1, 0, and -1. This range is still insufficient for practical use. We estimated that practical applications need at least 4 or 5 bits. The specifications of the processor are still being studied, and co-design with applications is needed. We will continue working on a prototype to develop the architecture and to ensure its efficient implementation using our proposed environment.

References

[1] Francisco Barahona. On the computational complexity of Ising spin glass models. *Journal of Physics A: Mathematical and General*, 15(10):3241, 1982.

[2] F. Belletti, M. Cotallo, A. Cruz, L. A. Fernandez, A. Gordillo-Guerrero, M. Guidetti, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, A. Munoz-Sudupe, D. Navarro, G. Parisi, S. Perez-Gaviro, M. Rossi, J. J. Ruiz-Lorenzo, S. F. Schifano, D. Sciretti, A. Tarancon, R. Tripiccione, J. L. Velasco, D. Yllanes, and G. Zanier. Janus: An FPGA-based system for high-performance scientific computing. *Computing in Science Engineering*, 11(1):48–58, Jan 2009.

[3] Maciej Brodowicz and Thomas Sterling. A non von neumann continuum computer architecture for scalability beyond Moore's law. In *Proceedings of the ACM International Conference on Computing Frontiers*, CF '16, pages 335–338, New York, NY, USA, 2016. ACM.

[4] Stephen G. Brush. History of the Lenz-Ising model. *Rev. Mod. Phys.*, 39:883–893, Oct 1967.

[5] A Gilman, A Leist, and KA Hawick. 3D lattice Monte Carlo simulations on FPGAs. In *Proceedings of the International Conference on Computer Design (CDES)*. The Steering Committee

of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[6] Hidenori Gyoten, Masayuki Hiromoto, and Takashi Sato. An FPGA implementation of fast 2D-Ising-model solver for the Max-Cut problem. In *IEICE Tech. Rep. (VLD2015-133)*, volume 115, pages 125–130, Feburary 2016.

[7] Masato Hayashi, Masanao Yamaoka, Chihiro Yoshimura, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. Accelerator chip for ground-state searches of ising model with asynchronous random pulse distribution. *International Journal of Networking and Computing*, 6(2):195–211, 2016.

[8] MW Johnson, MHS Amin, S Gildert, T Lanting, F Hamze, N Dickson, R Harris, AJ Berkley, J Johansson, P Bunyk, et al. Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198, 2011.

[9] Y Lin, F Wang, X Zheng, H Gao, and L Zhang. Monte Carlo simulation of the Ising model on FPGA. *Journal of Computational Physics*, 237:224–234, 2013.

[10] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2(5), 2014.

[11] George Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003.

[12] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[13] T. Okuyama, C. Yoshimura, M. Hayashi, and M. Yamaoka. Computing architecture to perform approximated simulated annealing for Ising models. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Oct 2016.

[14] Jagdish K Patel and Campbell B Read. *Handbook of the normal distribution*, volume 150. CRC Press, 1996.

[15] T. Skotnicki, J. A. Hutchby, Tsu-Jae King, H. S. P. Wong, and F. Boeuf. The end of CMOS scaling: toward the introduction of new materials and structural changes to improve MOSFET performance. *IEEE Circuits and Devices Magazine*, 21(1):16–26, Jan 2005.

[16] K. D Tocher. *The Art of Simulation.* English Universities Press, 1967.

[17] Shoko Utsunomiya, Kenta Takata, and Yoshihisa Yamamoto. Mapping of Ising models onto injection-locked laser systems. *Optics express*, 19(19):18091–18108, 2011.

[18] Masanao Yamaoka, Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. 20k-spin Ising chip for combinational optimization problem with CMOS annealing. *ISSCC Dig. Tech. Papers*, pages 432–433, 2015.

[19] Masanao Yamaoka, Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing. *IEEE Journal of Solid-State Circuits*, 51:303–309, 2016.

[20] C. Yoshimura, M. Yamaoka, H. Aoki, and H. Mizuno. Spatial computing architecture using randomness of memory cell stability under voltage control. In *2013 European Conference on Circuit Theory and Design (ECCTD)*, pages 1–4, Sept 2013.

[21] Chihiro Yoshimura, Masanao Yamaoka, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, Kenichi Kawarabayashi, and Hiroyuki Mizuno. Uncertain behaviours of integrated circuits improve computational performance. *Scientific Reports*, 5, 2015.