A Design Framework for Invertible Logic

Naoya Onizawa, *Member, IEEE*, Kaito Nishino, Sean C. Smithson, *Student Member, IEEE*, Brett H. Meyer, *Senior Member, IEEE*, Warren J. Gross, *Senior Member, IEEE*, Hitoshi Yamagata, Hiroyuki Fujita, and Takahiro Hanyu, *Senior Member, IEEE*

***Abstract*—Invertible logic using a probabilistic magnetoresistive device model has been recently presented that can compute functions in bidirectional ways and solve several problems quickly, such as factorization and combinational optimization. In this article, we present a design framework for invertible logic circuits. Our approach makes use of linear programming to create a Hamiltonian library with the minimum number of nodes for small invertible-logic functions. In addition, as the device model is approximated based on stochastic computing in synthesizable SystemVerilog, a faster simulation using the compiled SystemC binary is realized than a conventional SPICE-level simulation and is verified using field-programmable gate array (FPGA) as prototyping. Using our design framework, several invertible-logic circuits are designed and emulated (verified) in SystemC, exhibiting five order-of-magnitude faster simulation than conventional work.**

***keywords*—Field-programmable gate array (FPGA), Hamiltonian, stochastic computing, SystemVerilog model.**

I. Introduction

INVERTIBLE logic has been recently presented for providing a capability of forward and backward operations [1] as opposed to typical binary logic for the forward operation. It is designed based on underlying Boltzmann machines [2] and probabilistic magnetoresistive device models (p-bits) [3] whose input and output signals are represented by random bit streams. The bidirectional computing capability is realized by reducing the network energies of the machines with noise control (e.g., a multiplier could be used as a factorizer in the backward mode). Hence, several challenging problems could be quickly solved, such as integer factorization (e.g., cryptography problems [4]), combinatorial optimization (e.g., wireless sensor networks [5]), and machine learning (e.g., training neural networks [6]).

Manuscript received October 8, 2019; revised January 30, 2020; accepted June 4, 2020. Date of publication June 22, 2020; date of current version March 19, 2021. This work was supported in part by Brainware LSI Project of MEXT, Japan, JST PRESTO under Grant JPMJPR18M5, and in part by Canon Medical Systems Corporation. This article was recommended by Associate Editor J. Cortadella. *(Corresponding author: Naoya Onizawa.)*

Naoya Onizawa, Kaito Nishino, and Takahiro Hanyu are with the Research Institute of Electrical Communication, Tohoku University, Sendai 980-8577, Japan (e-mail: naoya.onizawa.a7@tohoku.ac.jp; hanyu@riec.tohoku.ac.jp).

Sean C. Smithson, Brett H. Meyer, and Warren J. Gross are with the Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 0E9, Canada (e-mail: sean.smithson@mail.mcgill.ca; brett.meyer@mcgill.ca; warren.gross@mcgill.ca).

Hitoshi Yamagata and Hiroyuki Fujita are with the Advanced Research Laboratory, Canon Medical Systems Corporation, Otawara 324-8550, Japan (e-mail: hitoshi.yamagata@medical.canon; hiroyuki12.fujita@medical.canon).

Digital Object Identifier 10.1109/TCAD.2020.3003906

However, there are several issues for designing large-scale invertible logic circuits. The functions that operate in bidirectional modes are defined by the Boltzmann machine configurations (Hamiltonians). A method of generating Hamiltonian is limited to small function blocks, such as Boolean logic [7], [8]. Another issue is the simulation speed due to the complicated device model described at the transistor level. In [9], the model is emulated in software using a microcontroller, however, the simulation speed per sample is not fast (e.g., 100–300 ms).

In this article, a design framework for large-scale invertible logic is presented in order to tackle the two main issues: 1) network configurations (Hamiltonians) and 2) simulation speed. For the small network configurations, a Hamiltonian library is created based on linear programming (LP), which provides the minimum number of nodes in Hamiltonians for basic functions, including adders and nonlinear functions. In addition, Hamiltonians of large functions (e.g., multiplication) can be constructed by adding those of small function blocks. For faster simulations, the probabilistic device model is approximated using stochastic computing [10]–[12] in synthesizable SystemVerilog. Stochastic computing that uses random bit streams realizes area-efficient computation blocks (e.g., multiplication and tanh function) and has been recently used for several applications, such as low-density parity-check decoders [13]–[16], image processing [17], [18], and deep neural networks [19]. As invertible logic may operate as serial computing, stochastic computing efficiently approximates the device model. Therefore, invertible logic can be emulated (verified) in the compiled SystemC environment and verified in the prototyping hardware (FPGA). Using our design framework, two noise-control methods are introduced and discussed in terms of convergence speed.

Our contributions are summarized: 1) the first design framework for invertible logic from specification to simulation; 2) Hamiltonian design using LP with the minimum number of nodes; and 3) five order-of-magnitude faster simulation than conventional works. The remainder of this article is as follows. Section II reviews invertible logic with related works and discusses the current issues of invertible logic design. Section III describes an overview of the proposed design framework for invertible logic. Section IV introduces a creation of Hamiltonian library using LP and a method of designing large-scale invertible logic. Section V models the probabilistic device model (p-bits) using stochastic computing for fast simulation. Section VI introduces two noise-control optimization methods for fast convergence of invertible logic.

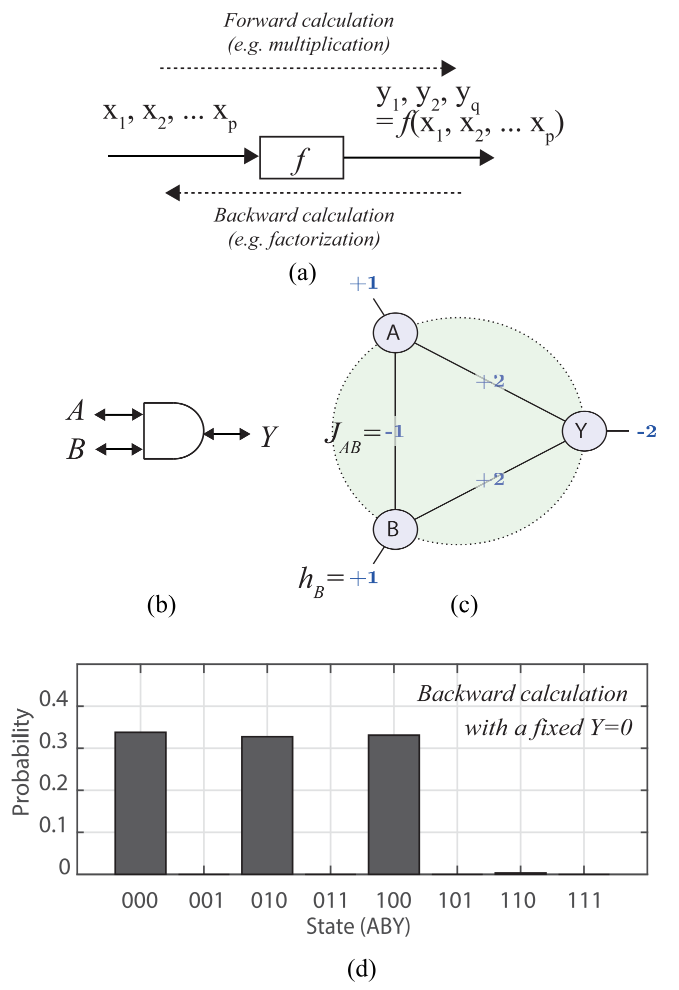


Fig. 1. Invertible logic: (a) concept, (b) invertible AND, (c) Hamiltonian of invertible AND, and (d) state probabilities when Y is fixed to 0.

Section VII evaluates the proposed design framework with the conventional work in several aspects, such as Hamiltonian and simulation speed. Section VIII concludes this article.

II. Preliminary

A. Invertible Logic

Fig. 1(a) shows a concept of invertible logic realized using Boltzmann machine and probabilistic bits (p-bits) [1]. Invertible logic circuits operate at forward and/or backward modes, where functions are embedded using Hamiltonian with inputs ($x_i \in \{0, 1\}$ $(1 \leq i \leq p)$) and outputs ($y_i \in \{0, 1\}$ $(1 \leq i \leq q)$). Note that the 2's complement format is used to represent data in invertible logic throughout this article. For example, an invertible multiplier exhibits a capability of multiplication with fixed inputs (forward mode) and factorization with fixed outputs (backward mode). If partial inputs and outputs are fixed, the invertible multiplier operates as division.

Fig. 1(c) shows a Hamiltonian of a two-input AND corresponding to the gate shown in Fig. 1(b). There are three nodes, where weight values (J) between nodes and bias values at nodes are given by

$$\mathrm{h}_{\mathrm{AND}} = \begin{bmatrix} +1 & +1 & -2 \end{bmatrix} \tag{1a}$$

$$\mathrm{J}_{\mathrm{AND}} = \begin{bmatrix} 0 & -1 & +2 \\ -1 & 0 & +2 \\ +2 & +2 & 0 \end{bmatrix} \tag{1b}$$

where the first two rows correspond to A and B and the last row corresponds to Y. Hamiltonians of simple logic gates can be obtained using ground-state spin logic [7], [8]. With given h and J, each node (p-bit) probabilistically generates an output ($m_i \in \{-1, 1\}$ $(1 \leq i \leq l)$), where l is the number of nodes. m_i is given by the following equations:

$$m_i(t+\tau) = \mathrm{sgn}(\mathrm{rnd}(-1, +1) + \tanh(\mathrm{I_i}(\mathrm{t}+\tau))) \tag{2a}$$

$$I_i(t+\tau) = I_0\left(h_i + \sum_j J_{ij} m_j(t)\right) \tag{2b}$$

where $rnd(-1, +1)$ is a uniformly distributed random (real) number between -1 and $+1$, sgn is the sign function (with binary $+1$ or -1 outputs), and I_0 is a scaling factor (an inverse *pseudo-temperature*). As m_i is represented in bipolar format, "$m_i = +1$" and "$m_i = -1$" correspond to logic values of "1" and "0," respectively.

Energies (H) of invertible logic circuits are given by

$$H = -\sum_i h_i m_i - \sum_{i<j} J_{ij} m_i m_j. \tag{3}$$

By controlling noise levels using several parameters, such as I_0, H ideally decreases to the global minimum, leading to desired inputs and/or outputs. Fig. 1(d) shows an example of the two-input invertible AND in the backward mode. With fixing the output (Y) to "0" ("$m_y = -1$"), there are three valid states ("ABY") of ("000," "010," "100"). In this simulation, the three valid states are obtained with almost the same probability of 33%.

B. Related Works

Table I summarizes comparisons of logic family characteristics. Unlike conventional Boolean logic that realizes only forward operations, invertible logic can realize bidirectional (forward/backward) operations. The number of inputs and outputs are flexible, while computation is deterministic or probabilistic in conventional and invertible logic, respectively. Invertible logic is designed using a probabilistic device model and can be implemented using a magnetoresistive device [1].

Reversible logic circuits are constructed of special gates (such as controlled NOT (CNOT) or Toffoli gates) having a direct one-to-one mapping of inputs to outputs [20]. While reversible logic gates allow for circuits to be built which are bidirectional, they must be designed differently and do not include standard gates (such as AND or OR gates) and require different design methods, such as binary decision diagrams (BDDs) [21]. While both reversible and invertible logic circuits reconstruct inputs from a given output value, they differ at fundamental levels. Unlike invertible logic, the number of inputs is equal to the number of outputs, which could require additional outputs/inputs, such as even a simple AND gate in reversible logic [22]. For physical realization, gates of reversible logic used in quantum circuits can be converted to standard binary logic that can be in turn realized in standard CMOS.

TABLE I
COMPARISONS OF LOGIC-FAMILY CHARACTERISTICS

Logic family	Operation	Function representation	I/O conditions	Physical realization
Conventional	Forward	Boolean algebra	n-input / m-output	CMOS circuit
Reversible	Forward / Backward	BDD, matrix representation	n-input / n-output	Quantum circuit
Invertible	Forward / Backward	Hamiltonian	n-input / m-output	Probabilistic circuit

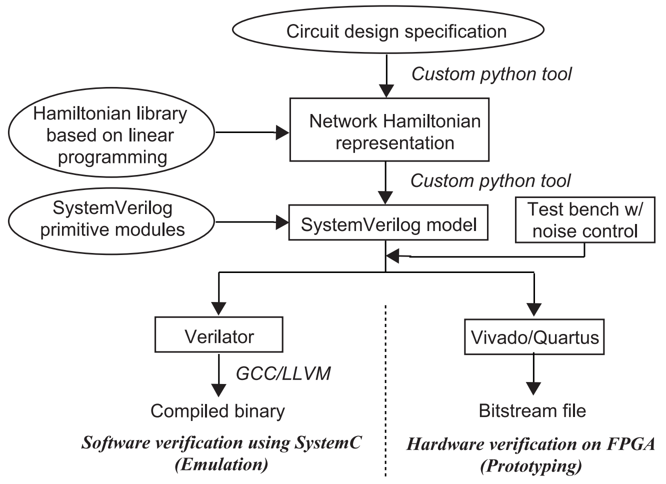

Circuit design specification
Custom python tool
Hamiltonian library based on linear programming
Network Hamiltonian representation
Custom python tool
SystemVerilog primitive modules
SystemVerilog model
Test bench w/ noise control
Verilator
GCC/LLVM
Compiled binary
Software verification using SystemC (Emulation)
Vivado/Quartus
Bitstream file
Hardware verification on FPGA (Prototyping)

Fig. 2. Proposed design framework for invertible logic.

C. Design Issues With Invertible Logic

There are two main issues for large-scale invertible-logic circuits. The first issue is a Hamiltonian design method. Different from reversible logic, large-scale functions (e.g., multiplication) are represented using a corresponding Hamiltonian as it is designed by adding small Hamiltonians based on ground-state spin logic [7], [8]. However, a variety of Hamiltonians is limited to small functions, such as AND. In addition, there is no specific design method of creating Hamiltonians corresponding to other functions. The second issue is simulation speed. Small invertible logic circuits have been designed and simulated at the transistor level [1] and in a microcontroller (software) [9], which takes 100–300 ms for a cycle of operation. For designing large-scale invertible logic, slow simulation could be a critical issue. Especially, as a control of noise effect, I_0, in Eq. (2) is required to converge to a valid state (minimum energy), a parameter search of the noise effect is required. A fast simulator allows designers to find a good noise parameter quickly. In this article, these two issues are mainly tackled using the proposed design framework for large-scale invertible circuits, such as the design methodology of large variety of Hamiltonians, fast simulation environment, and noise-control optimization.

III. DESIGN FRAMEWORK

Fig. 2 shows the proposed design framework for invertible logic. Let us explain the framework from the beginning.

1) A circuit design specification is defined, such as desired functions and input/output bit widths.
2) A whole network Hamiltonian corresponding to the function is generated based on a Hamiltonian library. The Hamiltonian library is preliminarily created using LP described in Section IV, where Hamiltonians of small invertible logic circuits are included in the library, such as logic functions and adders. The whole Hamiltonian is obtained by adding the small Hamiltonians using our custom Python tool.
3) The whole Hamiltonian is converted to the corresponding SystemVerilog model using SystemVerilog primitive modules. The primitive modules are preliminarily designed using stochastic computing [10] described in Section V, where stochastic computing approximates the probabilistic device model. The SystemVerilog model generated using our custom Python tool is synthesizable using commercial EDA tools, such as the synopsys design compiler.
4) A test bench is created with noise control of parameters, such as I_0. In invertible logic, the convergence speed could be significantly changed due to the noise control including hyper parameters, where a selection of optimum parameters can reach the global minimum energy. Two noise-control methods are introduced in Section VI.
5) The invertible logic circuit using the SystemVerilog model is verified (emulated) using Verilator [23] that is faster than SPICE simulations and interpreted Verilog simulations, where Verilator compiles SystemC test benches and the SystemVerilog models. Using the fast simulation environment, hyper parameters for fast convergence to the global minimum can be optimized (noise-control optimization) described in Section VI. In addition, the SystemVerilog model can be verified using field-programmable gate array (FPGA) boards for quite large invertible circuits as prototyping through commercial FPGA design tools, such as Xilinx Vivado.

IV. HAMILTONIAN DESIGN

A. Hamiltonian Library of Small Invertible Logic Using Linear Programming

Hamiltonians of small functions blocks, such as logic gates, are obtained using LP. Fig. 3 illustrates an example of Hamiltonian design of an invertible AND ($Y = A \wedge B$). There are total eight states that are divided into valid and invalid states based on the AND function.

Let us explain a procedure of generating a Hamiltonian using the invertible AND. The inputs ($x_i \in \{0, 1\}$ $(1 \leq i \leq p)$) and the outputs ($y_i \in \{0, 1\}$ $(1 \leq i \leq q)$) are defined shown in Fig. 1(a). First, logical values are converted to bipolar format as m_i. Second, an energy of each state (E_k $(1 \leq k \leq (l+l(l-1)/2))$) is defined based on Eq. (3), where l is a summation of input and output bit widths. In this case, l is 3 and the maximum k is 6. In invertible logic, the energies of the

A	B	Y	State
0	0	0	Valid
0	0	1	Invalid
0	1	0	Valid
0	1	1	Invalid
1	0	0	Valid
1	0	1	Invalid
1	1	0	Invalid
1	1	1	Valid

m_A	m_B	m_Y	H (Energy)
-1	-1	-1	**E0 = Emin**
-1	-1	1	E1 ≥ Emin + d
-1	1	-1	**E2 = Emin**
-1	1	1	E3 ≥ Emin + d
1	-1	-1	**E4 = Emin**
1	-1	1	E5 ≥ Emin + d
1	1	-1	E6 ≥ Emin + d
1	1	1	**E7 = Emin**

Fig. 3. Example of Hamiltonians design of an invertible AND ($Y = A \wedge B$) using linear programming (LP).

```
import pulp

problem = pulp.LpProblem('and', pulp.LpMinimize)
# Definition of variables
E = pulp.LpVariable('E',-3,-1,'Continuous')
h = pulp.LpVariable.dicts('h',([0],[0,1,2]),-2,2,'Integer'
  )
j = pulp.LpVariable.dicts('j',([0],[0,1,2]),-2,2,'Integer'
  )

for r in range(3):      # Object function
        H_hp = H_hp + arr[0][r]*h[0][r]
for r in range(2):
        ccc = r+1
        for c in range(ccc,3):
                H_jp = H_jp + arr[0][r]*arr[0][c]*j[0][0]
problem += (-1)*H_hp-H_jp-E

for i in range(0,col):  # Constraint conditions
        if IN_A*IN_B == OUT_C:
                ///
                problem += (-1)*H_h-H_j-E == 0
        else:
                ///
                problem += (-1)*H_h-H_j-E-1 >= 0

status = problem.solve()        # Solving this LP
```

Listing 1. Part of the Python Code of LP With PuLP for the Invertible-AND Hamiltonian

valid states must be equal to the minimum ($E_{\min}$) while that of the invalid states are larger than $E_{\min}$ described as following:

$$E_k = \begin{cases} -\sum_i h_i m_i - \sum_{i<j} J_{ij} m_i m_j = E_{\min} \\ \quad (f(x_1 \cdots x_p) = (y_1 \cdots y_q)) \\ -\sum_i h_i m_i - \sum_{i<j} J_{ij} m_i m_j \geq E_{\min} + d \\ \quad (\text{otherwise}) \end{cases} \tag{4}$$

where d is the energy difference between $E_{\min}$ and the second minimum energy. Third, the objective function is maximizing d using LP in order to obtain h_i and J_{ij} as follows:

$$\text{maximize} \quad d \tag{5}$$
$$\text{subject to} \quad Eq.\ (4) \tag{6}$$

where m_i and m_j are constants and h_i, J_{ij}, E, and d are variables.

Hamiltonians are obtained using LP with PuLP [24]. Listing 1 shows a part of python code of LP for the invertible-AND Hamiltonian. Using this method, Hamiltonians of small functions blocks are obtained in Table II. The number of nodes is the summation of input and output bit widths. These numbers are the minimum value because there is not auxiliary bit (node). Note that the auxiliary bits are extra bits except the input and the output bits [see Fig. 5 (a)].

In addition to the logic functions and the adders, Hamiltonians of several unique functions, such as bitcount function and rectified linear unit (ReLU) function can be obtained using LP. The reason of creating these Hamiltonians is that these functions are often used for machine learning as building blocks in neural networks [25]–[27]. Both functions are activation functions of neural networks, where the bitcount function is used in binary neural networks. By using these building blocks, invertible logic could be applied for machine learning, especially training neural networks using the bidirectional operations of invertible logic [6].

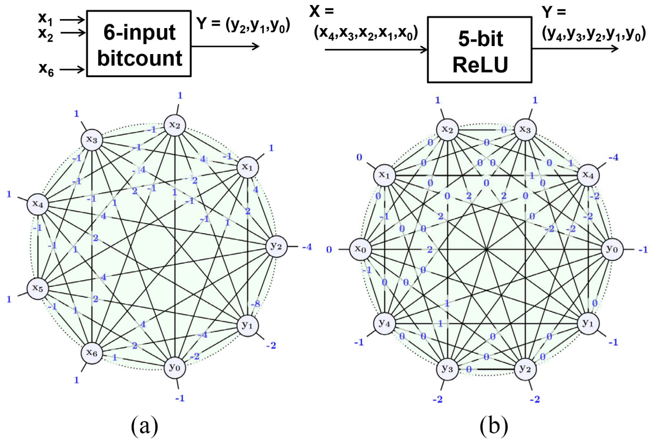

x_1
x_2
x_6
6-input bitcount
Y = (y_2,y_1,y_0)
X = (x_4,x_3,x_2,x_1,x_0)
5-bit ReLU
Y = (y_4,y_3,y_2,y_1,y_0)
(a)
(b)

Fig. 4. Examples of Hamiltonian that could be used for machine learning: (a) bitcount function with six inputs of $(x_1, x_2, \ldots, x_6)$ and a 3-b unsigned output of $Y = (y_2, y_1, y_0)$ and (b) 5-b ReLU function with a 5-b signed input of $X = (x_4, x_3, x_2, x_1, x_0)$ and a 5-b signed output of $Y = (y_4, y_3, y_2, y_1, y_0)$ in 2's complement format.

TABLE II
HAMILTONIAN LIBRARY GENERATED USING LP

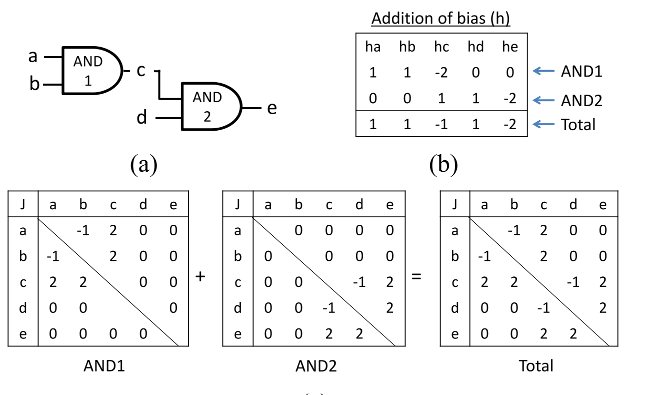
Function	Minimum number of nodes
AND, OR, NAND, NOR	3
XOR, XNOR, HA	4
FA	5
n-bit adder	$(3n+1)$
n-input bitcount	$n + \lceil \log_2(n+1) \rceil$
n-bit ReLU	$2n$

Fig. 4(a) shows a Hamiltonian example of a 6-input bitcount function with 6 inputs of $(x_1, x_2, \ldots, x_6)$ and a 3-b unsigned output of $Y = (y_2, y_1, y_0)$ in 2's complement format. The invertible bitcount circuit can realize $Y = \sum_{i=1}^{6} x_i$ in forward and backward modes. Fig. 4(b) shows a Hamiltonian example of a 5-b ReLU function with a 5-b signed input of $X = (x_4, x_3, x_2, x_1, x_0)$ and a 5-b signed output of $Y = (y_4, y_3, y_2, y_1, y_0)$, where the function of ReLU is defined by $Y = \max(0, X)$.

B. Hamiltonian Construction for Large Invertible Logic

Hamiltonians of large and/or complicated functions, such as multiplication, cannot be directly generated using LP because of linear separability problems. Hence, auxiliary bits are required to create such Hamiltonians. The whole Hamiltonian can be created by adding small Hamiltonians as follows:

$$h = \sum_k h_k \tag{7}$$



Addition of bias (h)

ha	hb	hc	hd	he	
1	1	-2	0	0	← AND1
0	0	1	1	-2	← AND2
1	1	-1	1	-2	← Total

AND1

J	a	b	c	d	e
a		-1	2	0	0
b	-1		2	0	0
c	2	2		0	0
d	0	0			0
e	0	0	0	0	

+

AND2

J	a	b	c	d	e
a		0	0	0	0
b	0		0	0	0
c	0	0		-1	2
d	0	0	-1		2
e	0	0	2	2	

=

Total

J	a	b	c	d	e
a		-1	2	0	0
b	-1		2	0	0
c	2	2		-1	2
d	0	0	-1		2
e	0	0	2	2	

Fig. 5. Hamiltonian design example of a three-input invertible AND by adding two Hamiltonians of two-input AND gates: (a) block diagram using two 2-input AND gates, (b) of h, and (c) addition of J.

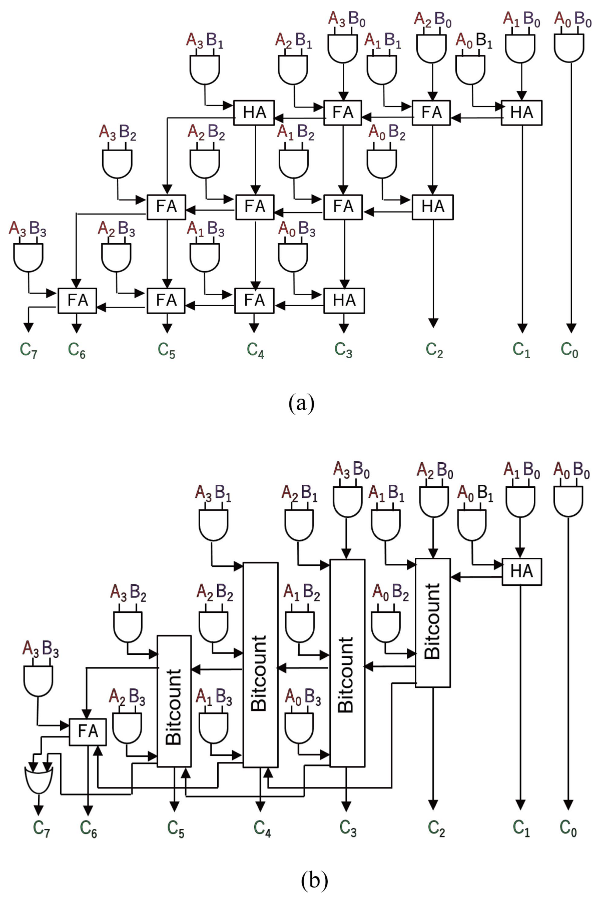


Fig. 6. 4×4-bit unsigned multiplier architecture for constructing Hamiltonians: (a) adder-based structure and (b) bitcount-based structure (proposed) that decreases the number of vertical interconnections (auxiliary bits).

$$J = \sum_k J_k \tag{8}$$

where h_k and J_k represent a Hamiltonian corresponding to a small circuit, such as AND, HA, and FA.

Fig. 5 shows an example of Hamiltonians of a three-input AND logic. The Hamiltonian is obtained by adding two Hamiltonians of the two-input AND logic. In this case, there are an additional connection (c) that becomes an auxiliary bit. If the Hamiltonian is directly created from the three-input AND logic, the auxiliary bit could be removed, leading to the minimum number of nodes. When the number of nodes is increased due to the auxiliary bits, the hardware of invertible logic could be larger and the convergence speed could be slower.

For designing Hamiltonians of large invertible logic, a circuit architecture is a important factor that can affect the performance of invertible logic. Fig. 6(a) shows a 4×4-bit unsigned multiplier architecture based on a simple adder-based structure. This design includes (2 × 4) inputs, 8 outputs and 32 internal connections. The Hamiltonian is obtained by adding that of AND and FA generated using LP. The number of nodes in the Hamiltonian is 48. The number of internal connections (auxiliary bits) is exponentially increased when the input bit width is increased because of horizontal and vertical internal connections. Note that a well-known Wallace-tree structure for fast multiplier design in conventional logic [28] causes a larger number of internal signals (nodes) than the adder-based structure.

In order to obtain smaller number of nodes, the proposed multiplier is designed using the bitcount circuits as shown in Fig. 6(b). As there is no internal connection in the bitcount circuit, the vertical internal connections can be eliminated, leading to a smaller number of nodes. In case of the 4×4-bit multiplier, the number of internal connections (auxiliary bits) decreases to 26 and hence the total number of nodes decreases to 42. The reduction method is much more effective in larger multipliers.

Fig. 7 compares the number of nodes in invertible multipliers (factorizers). The number of nodes is exponentially increased in the conventional adder-based multiplier because horizontal and vertical internal connections (auxiliary bits) are required. As the proposed bitcount-based structure eliminates the vertical internal connections, the number of nodes is almost linearly increased, leading to significant reductions in the number of nodes. As a result, the number of nodes in the 4×4-bit and the 12×12-bit multipliers are reduced by 80.6% and 89.1%, respectively. The detailed evaluation is described in Section VII-E.

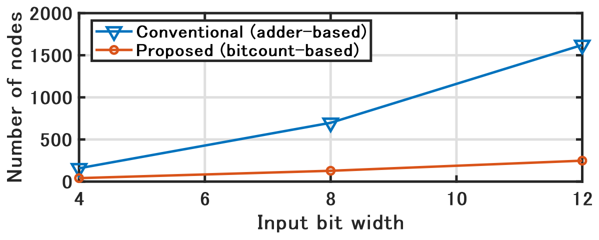
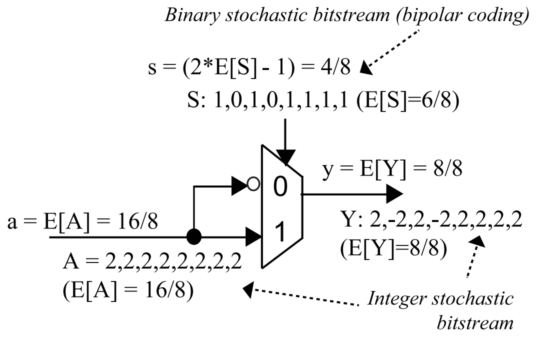


Fig. 7. Number of nodes in invertible multipliers (factorizers). The proposed structure realizes almost a linear growth of nodes in proportion to the input bit width while the exponential growth occurs in the conventional structure.

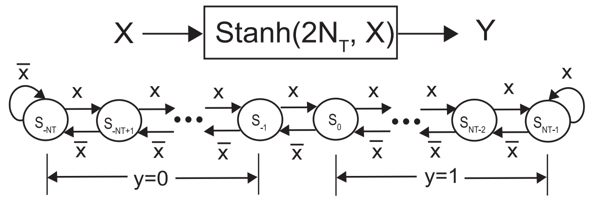
V. SystemVerilog Model Using Stochastic Computing

A. Binary and Integral Stochastic Computing

In invertible logic, p-bits operate based on Eq. (2) with bias values (h) and weight values (J) of Hamiltonians. In



(a)



(b)

Fig. 8. Binary and integral stochastic computing in bipolar coding: (a) multiplier of an integer stochastic bitstream and a binary stochastic bitstream ($y = a * s$) and (b) stochastic tanh function realized using a saturated updown counter.

order to realize faster simulations than conventional works at the transistor level [1] and Microcontroller (software) [9], a SystemVerilog model corresponding to Eq. (2) is created. The SystemVerilog model is designed based on binary and integral stochastic computing [10], [19].

In stochastic computing, data values are represented by frequencies of "1" in bit streams. Let us denote by $S \in \{0, 1\}$ a random bit streams. A real number, $s \in [-1 : 1]$, is represented by $(2 * \mathrm{E}[S] - 1)$ in binary stochastic computing in bipolar format, where $\mathrm{E}[S]$ denotes the expected value of the random variable, S. In contrast, in case of integral stochastic computing, one or more bit streams are concurrently used to represent data values in larger ranges than that of binary stochastic computing. Let us denote by $X \in \{-r, -(r-1), \ldots, r\}$ a random bit streams, where $r \in \{1, 2, \ldots\}$. A real number, $x \in [-r : r]$, is represented by $\mathrm{E}[X]$ in signed format, where $\mathrm{E}[X]$ denotes the expected value of the random variable X.

Stochastic computing realizes several functions, such as addition, multiplication, and nonlinear functions (see detail in [12]). Fig. 8(a) shows a multiplier of an integer stochastic bitstream and a binary stochastic bitstream ($y = a*s$) designed using a two-input multiplexor. Fig. 8(b) shows a tanh function block (Stanh) using a finite state machine (FSM) in stochastic computing. The tanh function is approximated using Stanh as follows:

$$\mathrm{Stanh}(2 \cdot \mathrm{N_T}, \mathrm{x}) \approx \tanh(\mathrm{x} \cdot \mathrm{N_T}) \tag{9}$$

where $2 \cdot N_T$ is the total number of states of the FSM. The Stanh block is designed using a saturated updown counter in hardware.

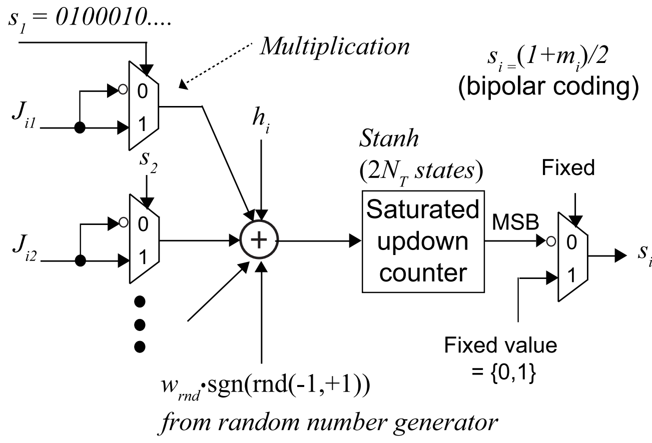


Fig. 9. Spin-gate circuit (SystemVerilog model) using stochastic computing, which corresponds to Eq. (10).

B. Spin-Gate Circuit for Modeling p-Bits

Fig. 9 shows a spin-gate circuit (SystemVerilog model) using binary and integral stochastic computing. This model approximates the original equation of Eq. (2) as follows:

$$m_i(t+\tau) \simeq \mathrm{sgn}(\tanh(\mathrm{I_i}(t+\tau) \cdot \mathrm{N_T})) \tag{10a}$$

$$I_i(t+\tau) \simeq \left(h_i + \sum_j J_{ij} m_j(t) + w_{rnd} \cdot \mathrm{sgn}(\mathrm{rnd}(-1, +1))\right) \tag{10b}$$

where I_0 of Eq. (2) corresponds to N_T. In addition, the weighted noise source with corresponding weight denoted as w_{rnd} is an additional parameter from Eq. (2). The weighted noise source is generated using a random number generator [29]. The model is designed as an extended version of [30] and [31], which can support controlling I_0 using N_T. The inputs and the output of the spin-gate circuits (m_i) are represented in binary stochastic computing in bipolar format as stochastic bit streams, $s_i = (1 + m_i)/2$. Instead, integral stochastic computing is exploited inside the spin-gate circuits in order to deal with integer values of h and J. As the model is fully designed using stochastic computing, it is synthesizable for standard digital CMOS circuits.

VI. Noise-Control Optimization

In invertible logic, it is important to control noise effects in order to reach the global minimum of energy (Hamiltonian). To converge node states to that at the global minimum, N_T and/or w_{rnd} of Eq. (10) can be controlled as noise optimization. In this article, w_{rnd} is selected to be controlled for two scenarios.

A. Grid Search on Monotonous Noise Reduction

To find the optimum control of w_{rnd}, a grid search is used as shown in Fig. 10. In the grid-search method, w_{rnd} is linearly decreased using four parameters as follows.

1) *RND_WEIGHT:* The maximum value of w_{rnd}.
2) *RND_STEP:* The amount of noise drops.
3) N_s: The number of noise drops defined by $2^{\mathit{RND\_DECAY}\text{-}1}$.
4) *T:* The number of cycles at the same w_{rnd}.

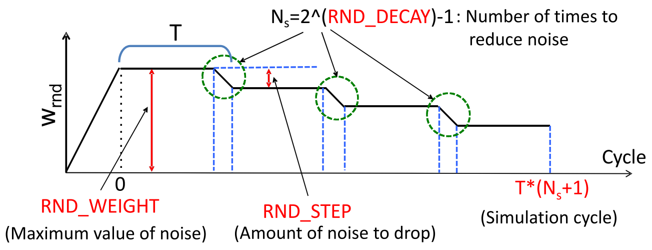


Fig. 10. MNR with grid search. The noise, w_{rnd}, is linearly decreased in order to converge energy to the global minimum.

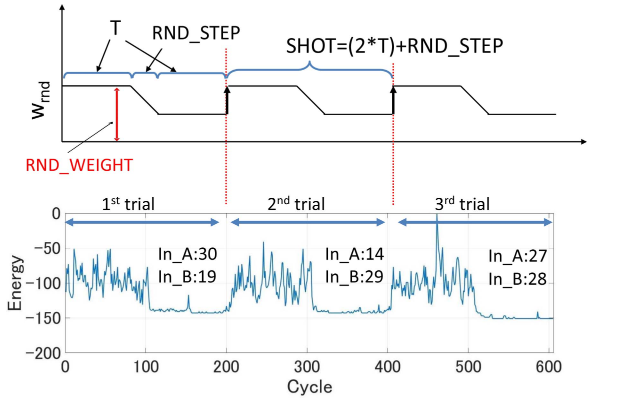


Fig. 11. Noise control based on TPR with an example of factorization of 756 ($in\_A \times in\_B$). A short pulsed noise is repeatedly applied to obtain the correct values. In this case, at the third trial, correct input values of 27 and 28 are obtained.

These four parameters are swept in order to converge energy to the global minimum. The monotonic noise reduction (MNR) method derives from [32] that monotonically increases I_0. This method is simple, but it requires long simulation time to find good noise parameters. The detailed simulation results are summarized in Section VII-D.

B. Tuning Parameter Repeat With Pulsed Noise

In order to reduce the simulation time of finding good noise parameters, tuning parameter repeat (TPR) is introduced. In this method, a short pulsed noise is repeatedly applied as opposed to MNR based on grid search. Fig. 11 shows a noise control based on TPR with an example of factorization of 756 ($in\_A \times in\_B$). In TPR, w_{rnd} is decreased from large to small as a one shot. There are three parameters in TPR as follows.

1) *RND_WEIGHT:* The maximum value of w_{rnd}.
2) *RND_STEP:* The amount of noise drops.
3) *T:* Cycles at large or small noise.

Hence, a cycle of one shot is $(2 * T + RND\_STEP)$.

As invertible logic is probabilistic, the results (energies) can be different, if the same noise parameter is applied. This example shows a factorization of 756 using TPR with $RND\_WEIGHT = 6$, $RND\_STEP = 4$, and $T = 6$. The tuning parameters were determined using simulations of a small invertible factorizer and can be applied to larger invertible factorizers. In this example, at the first and second trials, the correct input values of *in_A* and *in_B* are not obtained. In contrast, at the third trial using the same noise parameters, the correct input values of 27 and 28 are obtained. The comparison results with the grid search are summarized in Section VII-D.

TABLE III
COMPARISONS OF NUMBER OF NODES IN HAMILTONIAN

	Conventional [33]	Proposed
AND	3	3
Full adder	14	5
32-bit adder	434	97

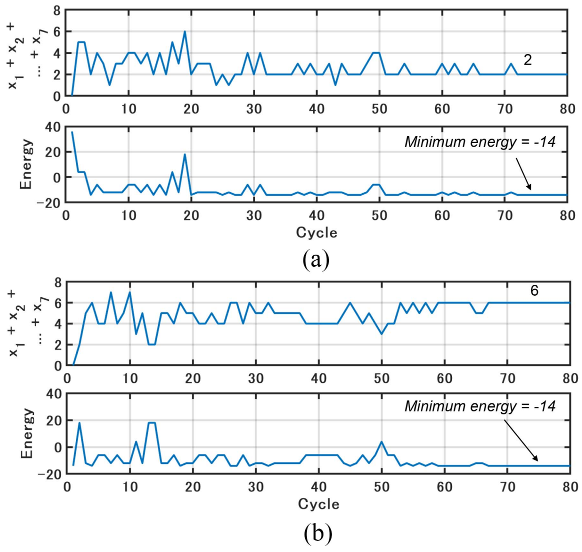


Fig. 12. Simulation results of a seven-input bitcount function in the backward mode with MNR. Given a fixed output of Y, seven inputs are correctly obtained at: (a) $Y = 2$ and (b) $Y = 6$.

VII. Evaluation

A. Comparisons of Hamiltonian

The Hamiltonian library is created using LP with PuLP [24] in AMD Opteron 6282 SE (2.6 GHz) used for all the simulations. Table III summarizes the number of nodes in Hamiltonians in comparison with a conventional work [33]. The conventional method is based on [1] that uses auxiliary bits and a handle bit to create Hamiltonians, causing a larger number of nodes. In contrast, the proposed method using LP generates the minimum number of nodes for the Hamiltonians of AND, FA, and 32-b adder. The number of nodes in FA and the 32-b adder are reduced by 64.3% and 77.7%, respectively, in comparison with the conventional method.

B. Simulation of Invertible Logic Circuits

Invertible logic circuits are simulated using our SystemVerilog model with the compiled SystemC binary in Verilator [23] and SystemC-2.3.2. Verilator is a fast Verilog-HDL simulator running on C++ and SystemC, which accepts synthesizable Verilog-HDL and SystemVerilog. Fig. 12 shows simulated waveforms of a seven-bit bitcount function in the backward mode. The output of Y is fixed in order to obtain correct seven inputs of $(x_1, x_2, \ldots, x_7)$ at $Y = 2$ and $Y = 6$ with a noise control of MNR, where $RND\_WEIGHT=16$, $RND\_STEP = 2$, $N_s = 7$, and $T = 10$ are used. When the

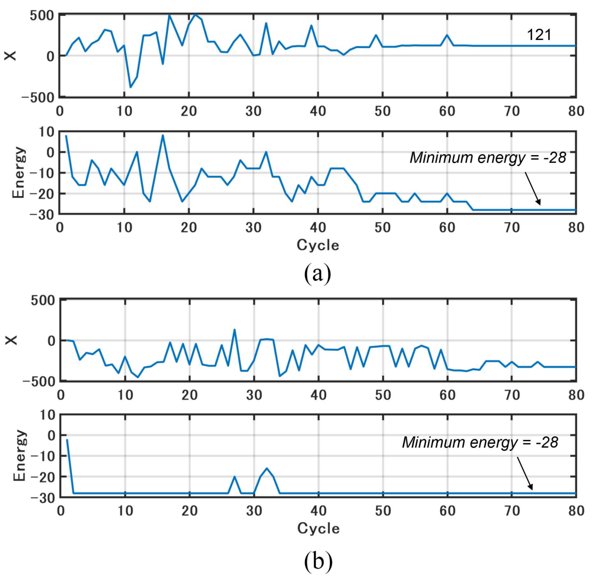


Fig. 13. Simulation results of a 10-b ReLU function in the backward mode with MNB. Given a fixed output of Y, 10-b inputs are correctly obtained at: (a) $Y = 121$ and (b) $Y = 0$.

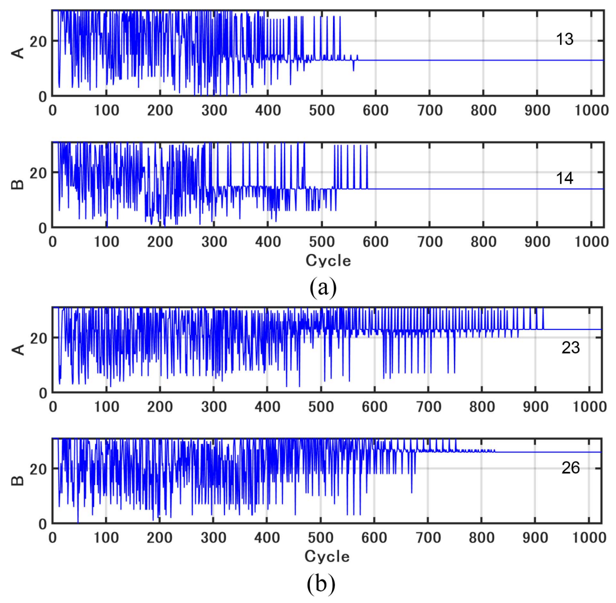


Fig. 14. Factorization results using SystemVerilog model with: (a) $(A \times B) = 182$ and (b) $(A \times B) = 598$. In both cases, the outputs are correctly factorized. The noise-control parameters are obtained based on the grid search.

energy defined by Eq. (3) reaches the global minimum of -14, the correct inputs are obtained.

Fig. 13 shows simulated waveforms of a 10-b ReLU function in the backward mode with the same noise control used in the previous simulation. When the output of Y is fixed to 121 in Fig. 13(a), the input of X reaches the correct value of 121 at the global minimum energy of -28. In contrast, when the output is fixed to 0 in Fig. 13(b), the input can be any negative values as the correct values because of the function: $Y = \mathrm{ReLU}(X) = \max(0, X)$.

Fig. 14 shows simulated waveforms of the invertible factorizer (adder-based) based on the architecture of Fig. 6(a).

TABLE IV
SIMULATION TIME PER SAMPLE (CYCLE) IN INVERTIBLE MULTIPLIERS (FACTORIZERS)

Bit width	Microcontroller [9]	SystemC (Proposed)
4-bit	100 - 300 ms	5.3 μs
10-bit	N/A	26 μs
32-bit	N/A	774 μs

For small invertible-logic circuits simulated in the previous paragraph, it is easy to reach the global minimum energy with many different noise parameters. In contrast, for large invertible-logic circuits, such as invertible multipliers, specific noise parameters are required for the convergence. In order to converge to the global minimum, first, the grid search on MNR is used to find the optimum control of w_{rnd} of Eq. (10). The total number of cycles is 9.5×10^7 in the grid search to find the optimum noise parameters on MNR. In case of $(A \times B) = 182$, w_{rnd} is reduced with *RND_WEIGHT*=8, *RND_STEP* = 1, $N_s = 7$, and $T = 128$. In contrast, in case of $(A \times B) = 598$, w_{rnd} is reduced with *RND_WEIGHT*=16, *RND_STEP* = 1, $N_s = 15$, and $T = 64$. Depending on the outputs $(A \times B)$, the optimum noise parameters are different, causing long convergence time, even though our fast simulation environment is used. The evaluation of noise control is described in Section VII-D.

C. Comparisons of Simulation Speed

Table IV summarizes the simulation time per sample (cycle) in a 2×2-bit invertible multiplier (factorizer). In the conventional work [9], the complicated device models of (2) are realized using software running on a microcontroller. The sample time is slow, such as 100–300 ms. In such the environment, a noise-control optimization of I_0 for convergence to the global minimum requires significantly large time.

In contrast, our SystemVerilog model using stochastic computing is simulated as the emulation of the device model in Verilator and SystemC-2.3.2. As a result, the cycle (sample) time is around 5.3 μs, leading to around five order of magnitude reductions. As opposed to the conventional work, larger invertible multipliers can be also designed and simulated, such as 32-b.

D. Comparisons of Noise Control in Invertible Multipliers

Table V summarizes the total number of cycles and the simulation time of invertible multipliers with different noise controls described in Section VI. The total number of cycles are ones until good noise parameters for convergence are obtained. Using the grid search, both the total number of cycles and the simulation time are significantly increased as the bit width is increased. As a result, in larger invertible logic circuits, it is hard to converge to the global minimum and hence obtain correct values.

In contrast, using TPR, both total number of cycles and the simulation time are negligibly increased as the bit width is increased. In case of the 16-b invertible multiplier, the simulation time is a five order-of-magnitude faster than that of the

TABLE V
SIMULATION TIME OF INVERTIBLE MULTIPLIERS WITH DIFFERENT NOISE CONTROLS IN SYSTEMC

	Grid search		Tuning parameter repeat (TPR)	
	Total cycle	Simulation time [s]	Total cycle	Simulation time [s]
4-bit × 4-bit (8-bit)	9.5×10^7	2.7×10^3	5.9×10^2	27
5-bit × 5-bit (10-bit)		5.2×10^3	1.9×10^3	65
6-bit × 6-bit (12-bit)		7.8×10^3	8.9×10^3	69
7-bit × 7-bit (14-bit)	7.6×10^8	9.4×10^4	4.2×10^4	120
8-bit × 8-bit (16-bit)	1.2×10^{10}	1.7×10^6	2.9×10^5	180

TABLE VI
SYNTHESIS RESULTS OF INVERTIBLE LOGIC CIRCUITS IN DIGILENT GENESYS 2

	LUT	FF	Number of nodes	Number of non-zero weights in Hamiltonian	Number of maximum connections per node
32-bit invertible adder	9,583	2,133	128	632	8
64-bit invertible adder	18,324	3584	256	1,272	8
7-bit bitcount	1,008	404	10	45	9
10-bit ReLU	2,277	558	20	74	18
16-bit invertible multiplier (adder-based)	15,430	2,901	756	1,440	16
32-bit invertible multiplier (adder-based)	62,969	9,393	3,072	6,288	32
32-bit invertible multiplier (bitcount-based)	58,929	5,832	423	8,530	36

grid search. The gap of the simulation time can be larger as the bit width is increased. Hence, TPR would be more effective for larger invertible logic circuits.

E. FPGA Implementation for Prototyping

As the SystemVerilog model is synthesizable, invertible logic circuits can be evaluated using FPGA as prototyping. Table VI summarizes the synthesis results of invertible logic circuits in Xilinx Vivado 2016.4 for Digilent Genesys 2 with the clock frequency of 100 MHz. As the clock cycle is 100 MHz, the sample time is 10-ns that significantly increases simulation speed in comparison with the conventional work and the proposed SystemC summarized in Table IV. Note that generating bitstream files for FPGA takes a much longer time than compiling to the SystemC binary files. Therefore, the SystemC-based environment is useful for small invertible circuits while the FPGA environment is useful large ones that require longer simulation time.

Considering the hardware resources, in general, the number of LUTs and FFs are large when the number of nodes and nonzero weights in Hamiltonian are large. Note that the number of nonzero weights are obtained from h and J of Hamiltonians. When adder-based and bitcount-based invertible multipliers are compared, the number of nodes in the bitcount is significantly smaller than that of the adder-based structure as described in Fig. 6. In contrast, the number of nonzero weights are larger because of the denser matrix of J. As a result, the bitcount-based invertible multiplier reduces LUT by 7% and FF by 38% in comparison with the adder-based one.

VIII. CONCLUSION

In this article, we have presented the design framework for large-scale invertible logic. The Hamiltonian library created using LP provides the minimum number of nodes in Hamiltonians for basic functions, where the library includes Boolean logic, adders, bitcount, and ReLU functions. As a design example of a large invertible logic circuit, the Hamiltonians for invertible multipliers (factorizers) are constructed using the library, resulting in more than 80% reduction in the number of nodes in comparison with that of the conventional structure. For fast simulations, the probabilistic device model used for invertible logic is approximated using stochastic computing in SystemVerilog running with the compiled SystemC binary, providing almost five orders-of-magnitude reductions in simulation time in comparison with the conventional environment. In our fast simulation environment, the tuning-parameter repeat method as noise-control optimization is introduced, reducing the convergence time by five orders-of-magnitude in comparison with the grid search method.

Invertible logic was recently presented to demonstrate integer factorization in 2017 [1] and have been studied in several aspects, such as scalability, applications, and implementations. The scalability has been studied and discussed in [32]; however, optimization algorithms for specific problems (e.g., graph coloring) have not been studied and would be a future research. One of the possible applications could be machine learning, especially training neural networks using the bidirectional operations of invertible logic [6]. In future prospects, our design framework would be useful as a design and test tool for implementing invertible logic with the probabilistic magnetoresistive devices. In addition, larger invertible logic circuits using stochastic computing with standard CMOS devices could be designed for several applications.

REFERENCES

[1] K. Camsari, R. Faria, B. Sutton, and S. Datta, "Stochastic p-bits for invertible logic," *Phys. Rev. X*, vol. 7, Jul. 2017, Art. no. 031014.
[2] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley, "Boltzmann machines: Constraint satisfaction networks that learn," Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, USA, Rep. CMU-CS-84-119, 1984.

[3] P. Debashis, R. Faria, K. Y. Camsari, J. Appenzeller, S. Datta, and Z. Chen, "Experimental demonstration of nanomagnet networks as hardware for Ising computing," in *Proc. IEEE Int. Electron Devices Meeting (IEDM)*, Dec. 2016, pp. 34.3.1–34.3.4.
[4] J. V. Monaco and M. M. Vindiola, "Factoring integers with a brain-inspired computer," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 3, pp. 1051–1062, Mar. 2018.
[5] Z. Zhou *et al.*, "Energy-efficient optimization for concurrent compositions of WSN services," *IEEE Access*, vol. 5, pp. 19994–20008, 2017.
[6] N. Onizawa, S. C. Smithson, B. H. Meyer, W. J. Gross, and T. Hanyu, "In-hardware training chip based on CMOS invertible logic for machine learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 5, pp. 1541–1550, May 2020.
[7] J. D. Biamonte, "Non-perturbative k-body to two-body commuting conversion Hamiltonians and embedding problem instances into ising spins," *Phys. Rev. A*, vol. 77, May 2008, Art. no. 052331.
[8] J. D. Whitfield, M. Faccin, and J. D. Biamonte, "Ground-state spin logic," *Europhys. Lett.*, vol. 99, no. 5, 2012, Art. no. 57004.
[9] A. Z. Pervaiz, L. A. Ghantasala, K. Camsari, and S. Datta, "Hardware emulation of stochastic p-bits for invertible logic," *Sci. Rep.*, vol. 7, no. 1, Sep. 2017, Art. no. 10994.
[10] B. R. Gaines, "Stochastic computing systems," *Adv. Inf. Syst. Sci. Plenum*, vol. 2, no. 2, pp. 37–172, 1969.
[11] B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.
[12] V. C. Gaudet and W. J. Gross, *Stochastic Computing: Techniques and Applications*. Cham, Switzerland: Springer Int., 2019.
[13] V. C. Gaudet and A. C. Rapley, "Iterative decoding using stochastic computation," *Electron. Lett*, vol. 39, no. 3, pp. 299–301, Feb. 2003.
[14] S. S. Tehrani, W. J. Gross, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Commun. Lett.*, vol. 10, no. 10, pp. 716–718, Oct. 2006.
[15] S. S. Tehrani, S. Mannor, and W. J. Gross, "Fully parallel stochastic LDPC decoders," *IEEE Trans. Signal Process.*, vol. 56, no. 11, pp. 5692–5703, Nov. 2008.
[16] S. S. Tehrani, A. Naderi, G. A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Trans. Signal Process.*, vol. 58, no. 9, pp. 4883–4896, Sep. 2010.
[17] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proc. 50th DAC*, May 2013, pp. 1–6.
[18] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.
[19] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W. J. Gross, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 10, pp. 2588–2599, Oct. 2017.
[20] M. Saeedi and I. L. Markov. (2011). *Synthesis and Optimization of Reversible Circuits—A Survey*. [Online]. Available: http://arxiv.org/abs/1110.2574
[21] A. Zulehner and R. Wille, "One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 996–1008, May 2018.
[22] A. Zulehner and R. Wille, "Make it reversible: Efficient embedding of non-reversible functions," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Mar. 2017, pp. 458–463.
[23] W. Snyder, P. Wasson, and D. Galbi. (2012). *Verilator: Convert Verilog Code to C++/SystemC*. [Online]. Available: https://www.veripool.org/wiki/verilator
[24] S. Mitchell, S. M. Consulting, and I. Dunning, *PuLP: A Linear Programming Toolkit for Python*, Univ. Auckland, Auckland, New Zealand, 2011. [Online]. Available: http://www.optimization-online.org/DB FILE/2011/09/3178.pdf
[25] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. 28th Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.
[26] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. (2016). *XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks*. [Online]. Available: http://arxiv.org/abs/1603.05279
[27] Y. LeCun, Y. Bengio, and G. Hinton., "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, Jul. 2015.
[28] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
[29] S. Vigna, "Further scramblings of Marsaglia's xorshift generators," *J. Comput. Appl. Math.*, vol. 315, pp. 175–181, May 2017.
[30] K. Nishino *et al.*, "Study of stochastic invertible multiplier designs," in *Proc. 25th IEEE Int. Conf. Electron. Circuits Syst. (ICECS)*, Dec. 2018, pp. 649–650.
[31] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu, "Efficient CMOS invertible logic using stochastic computing," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 6, pp. 2263–2274, Jun. 2019.
[32] K. Y. Camsari, S. Chowdhury, and S. Datta. (2019). *Scalable Emulation of Sign-Problem-Free Hamiltonians With Room Temperature p-Bits*. [Online]. Available: http://arxiv.org/abs/1810.07144
[33] A. Z. Pervaiz, B. M. Sutton, L. A. Ghantasala, and K. Y. Camsari, "Weighted p -bits for FPGA implementation of probabilistic circuits," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 6, pp. 1920–1926, Jun. 2019.

Naoya Onizawa (Member, IEEE) received the B.E., M.E., and D.E. degrees in electrical and communication engineering from Tohoku University, Sendai, Japan, in 2004, 2006, and 2009, respectively.

He is currently an Assistant Professor with the Research Institute of Electrical Communication, Tohoku University, and a JST PRESTO Researcher. He was a Postdoctoral Fellow with the University of Waterloo, Waterloo, ON, Canada, in 2011, and McGill University, Montreal, QC, Canada, from 2011 to 2013. In 2015, he was a Visiting Associate Professor with the University of Southern Brittany, Lorient, France. His main interests and activities are in the energy-efficient VLSI design based on asynchronous circuits and probabilistic computation, and their applications, such as brain-like computers.

Dr. Onizawa received the Best Paper Award IEEE ISVLSI in 2010, the Best Paper Finalist IEEE ASYNC in 2014, the Kenneth C. Smith Early Career Award for Microelectronics Research IEEE ISMVL in 2016, and the MEXT Young Scientists' Prize, Japan, in 2020.

Kaito Nishino received the B.E. degree in electrical and electronic engineering and the M.E. degree in communication engineering from Tohoku University, Sendai, Japan, in 2017 and 2019, respectively.

His research interests include design of invertible logic circuits and its application to stochastic-computing LSI systems.

Sean C. Smithson (Student Member, IEEE) was born in Calgary, AB, Canada, in 1983. He received the B.Eng. (with Distinction) and M.A.Sc. degrees in electrical engineering from Concordia University, Montreal, QC, Canada, in 2009 and 2014, respectively. He is currently pursuing the Ph.D. degree with McGill University, Montreal.

Brett H. Meyer (Senior Member, IEEE) received the B.S. degree in electrical engineering, computer science and math from the University of Wisconsin–Madison, Madison, WI, USA, in 2003, and the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2005 and 2009, respectively.

He is a Chwang-Seto Faculty Scholar and an Assistant Professor with the Department of ECE, McGill University, Montreal, QC, Canada. He worked as a Postdoctoral Research Associate with the Computer Science Department, University of Virginia, Charlottesville, VA, USA. He has been on the Faculty with McGill University since 2011. His research interests are focused on the design and architecture of resilient multiprocessor computer systems.

Warren J. Gross (Senior Member, IEEE) received the B.A.Sc. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1996, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 1999 and 2003, respectively.

He is currently a Professor, the Louis-Ho Faculty Scholar of technological innovation, and the Chair of the Department of Electrical and Computer Engineering, McGill University, Montreal, QC, Canada. His research interests are in the design and implementation of signal processing systems and custom computer architectures.

Prof. Gross served as the Chair of the IEEE Signal Processing Society Technical Committee on Design and Implementation of Signal Processing Systems. He has served as the General Co-Chair of IEEE GlobalSIP 2017 and IEEE SiPS 2017, and as the Technical Program Co-Chair of SiPS 2012. He has also served as organizer for the Workshop on Polar Coding in Wireless Communications at WCNC 2017, the Symposium on Data Flow Algorithms and Architecture for Signal Processing Systems (GlobalSIP 2014), and the IEEE ICC 2012 Workshop on Emerging Data Storage Technologies. He served as an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING and as a Senior Area Editor. He is a licensed Professional Engineer in the Province of Ontario.

Hitoshi Yamagata received the B.E. and M.S. degrees from the Department of Electronic Engineering, Tohoku University, Sendai, Japan, in 1978 and 1980, respectively, and the Ph.D. degree from the Department of Information Engineering, Tohoku University in 1983.

He joined the Division of Medical Systems, Toshiba, Minato, Japan, in 1983 and he has been developed full-digital X-ray system since 1984. He was a Postdoctoral Research Fellow with the University of California at San Francisco, San Francisco, CA, USA, and the University of California at Davis, Davis, CA, USA, from 1987 to 1990. He has been engaged in research and development of 1.5T MRI system, 3-D ultrasound system as a System Manager since 1991. He has been managed software development of clinical applications for medical diagnostic imaging systems including CT, MRI, and ultrasound since 2003.

Dr. Yamagata has been a Senior Fellow since 2009. After CANON acquired the division in 2016, he has been a fellow of CANON Medical Systems Corporation (CMSC) and is currently a member of Advanced Research Laboratory, CMSC.

Hiroyuki Fujita received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Tokyo, Tokyo, Japan, in 1975, 1977, and 1980, respectively.

He is a Director of the Advanced Research Laboratory, Canon Medical Systems Corporation, Otawara, Japan, in 2017. He is also a Professor with Tokyo City University, Tokyo, and a Professor Emeritus with the University of Tokyo, where he served as a Professor with the Institute of Industrial Science, over 38 years. He is currently engaged in the investigation of MEMS/NEMS and applications to bio/nano technology and IoT. He has published more than 300 academic papers and was a Visiting Professor with MIT, Cambridge, MA, USA, and the University of California at Berkeley, Berkeley, CA, USA.

Dr. Fujita received many awards, including the l'Ordre des Palmes Academiques from Government of France, the Docteur Honoris Causa from Ecole Normale Superieure de Cachan, the Prize for Science and Technology -Research Category from Ministry of Education, Culture, Sports, Science and Technology, the Outstanding Achievement Award from the Institute of Electrical Engineers of Japan, and the IEEE Robert Bosch Award for MEMS.

Takahiro Hanyu (Senior Member, IEEE) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1984, 1986, and 1989, respectively.

He is currently a Professor and the Vice Director of the Research Institute of Electrical Communication, Tohoku University. His general research interests include nonvolatile logic circuits and their applications to ultralow-power and/or highly dependable VLSI processors, and post-binary computing and its application to brain-inspired VLSI systems.

Prof. Hanyu received the Sakai Memorial Award from the Information Processing Society of Japan in 2000, the Judge's Special Award at the ninth LSI Design of the Year from the Semiconductor Industry News of Japan in 2002, the Special Feature Award at the University LSI Design Contest from ASP-DAC in 2007, the APEX Paper Award of Japan Society of Applied Physics in 2009, the Excellent Paper Award of IEICE, Japan, in 2010, the Ichimura Academic Award in 2010, the Best Paper Award of IEEE ISVLSI in 2010, the Paper Award of SSDM in 2012, the Best Paper Finalist of IEEE ASYNC in 2014, and the Commendation for Science and Technology by MEXT, Japan, in 2015.