PAPER Special Section on VLSI Design and CAD Algorithms

A Realization of Multiple-Output Functions by a Look-Up Table Ring

Hui QIN[†a)], *Nonmember*, **Tsutomu SASAO**[†,††b)], **Munehiro MATSUURA**[†c)], *Members*, **Shinobu NAGAYAMA**[†d)], *Student Member*, **Kazuyuki NAKAMURA**[††e)], *and* **Yukihiro IGUCHI**[†††f)], *Members*

SUMMARY A look-up table (LUT) cascade is a new type of a programmable logic device (PLD) that provides an alternative way to realize multiple-output functions. An LUT ring is an emulator for an LUT cascade. Compared with an LUT cascade, the LUT ring is more flexible. In this paper we discuss the realization of multiple-output functions with the LUT ring. Unlike an FPGA realization of a logic function, accurate prediction of the delay time is easy in an LUT ring realization. A prototype of an LUT ring has been custom-designed with 0.35 μm CMOS technology. Simulation results show that the LUT ring is 80 to 241 times faster than software programs on an SH-1, and 36 to 93 times faster than software programs on a PentiumIII when the frequencies for the LUT ring and the MPUs are the same, but is slightly slower than commercial FPGAs.

key words: *LUT cascade, LUT ring, multiple-output function, reconfigurable logic, programmable logic device*

1. Introduction

Programmable logic devices (PLDs) are widely used for prototyping and final products to reduce turnaround time and financial risk. In this paper, we consider the realization of multiple-output logic functions using PLDs. Various methods exist to realize multiple-output logic functions. Among them, RAMs and programmable logic arrays (PLAs) directly implement logic functions. However, when the number of input variables n is large, the necessary amount of the hardware becomes too large. Thus, field programmable gate arrays (FPGAs) are often used. However, FPGAs require both physical and logic design. Also, without a complete physical design, the prediction of the performance of FPGAs is hard because the area and delay for the interconnections are often much larger than for logic cells.

Another method to realize a logic functions is a branching program on a general-purpose microprocessor. Figure 1 shows an example for the function $f = x_1x_3 \vee \overline{x_1}\, x_2$. A logic function can be realized by a branching program as follows [1]:

1. Represent a given function by a binary decision diagram (BDD) [2], [3] (Fig. 1(a)).
2. Replace each non-terminal node of the BDD with an *if then else* statement, and derive the branching program representing f (Fig. 1(b)).
3. Execute the program on a general-purpose microprocessor.

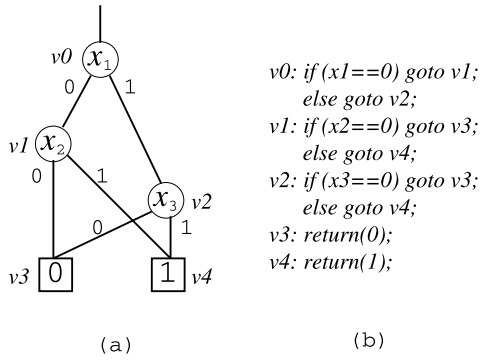

Fig. 1 Binary decision diagram and branching program.

In this way, an SBDD (shared BDD), an MTBDD (multi-terminal BDD), or a BDD-for-CF (characteristic function) can be used to represent a multiple-output logic function. The evaluation time for a BDD-for-CF or an MTBDD is $O(n + m)$ [1], [4], while the evaluation time for an SBDD is $O(n \times m)$, where n denotes the number of input variables, and m denotes the number of output variables. Thus, binary decision diagrams (BDDs) for characteristic functions (CFs) and MTBDDs are suitable for high speed evaluation.

A look-up table (LUT) cascade [5] is a new type of a PLD. Since it has a memory-like structure and the interconnections are simple, prediction of the circuit performance is easy. An LUT cascade shown in Fig. 2 consists of multiple memories (cells) connected in series to realize a given function. The lines that connect adjacent cells are called rails. Each cell may have external outputs and rail outputs except that the last cell has just external outputs. Although the LUT cascade is simple and fast, the logical capability is low. Once the number of inputs and outputs per cell, and the number of cells are fixed, the number of realizable functions is limited. Moreover, it is almost impossible to use all cells.

An LUT ring shown in Fig. 3 emulates an LUT cascade

Manuscript received March 19, 2004.
Manuscript revised June 7, 2004.
Final manuscript received August 5, 2004.

[†]The authors are with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
[††]The authors are with the Center for Microelectronic Systems, Kyushu Institute of Technology, Iizuka-shi, 820–8502 Japan.
[†††]The author is with the Department of Computer Science, Meiji University, Kawasaki-shi, 214–8571 Japan.
a) E-mail: qinhui@aries02.cse.kyutech.ac.jp
b) E-mail: sasao@cse.kyutech.ac.jp
c) E-mail: matsuura@cse.kyutech.ac.jp
d) E-mail: nagayama@aries02.cse.kyutech.ac.jp
e) E-mail: nakamura@cms.kyutech.ac.jp
f) E-mail: iguchi@cs.meiji.ac.jp

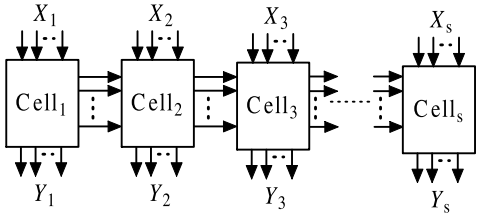

Fig. 2 LUT cascade.

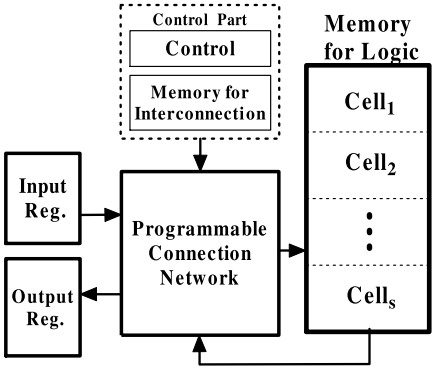

Fig. 3 LUT ring.

sequentially using memory for logic. Since the number of inputs and the number of outputs per cell and the number of cells can be changed by using a programmable connection network, the LUT ring can implement a wider range of functions. Also, memory-packing [5] can be used to reduce the total amount of memory. In [5], the LUT ring was called the LUT cascade. However, in this paper, the sequential circuits that emulate an LUT cascade will be called an LUT ring.

In this paper, we show a prototype of an LUT ring custom-designed with $0.35\,\mu$m CMOS technology, and compare its performance with microprocessors and FPGAs. The rest of the paper is organized as follows: Sect. 2 explains the architecture and features of the LUT ring. Section 3 presents the circuit design of the prototype. Section 4 evaluates the performance of the LUT ring, and Sect. 5 concludes the paper.

2. LUT Ring

In this section, we explain the architecture and features of an LUT ring.

The LUT ring shown in Fig. 3 consists of five parts: The Input Reg. stores the values of primary inputs; the Output Reg. stores the values of primary outputs; the Memory for Logic stores the LUT data for cells, where all of the LUT data are stored in one memory. The Control Part consists of the Control block that generates control signals and the Memory for Interconnections that stores the information of the interconnections among cells; the Programmable Connection Network implements the interconnections among cells.

An LUT ring sequentially emulates the LUT cascade.

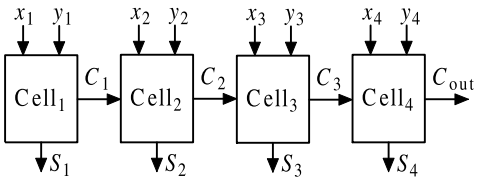

Fig. 4 Structure of the LUT cascade for 4-bit adder.

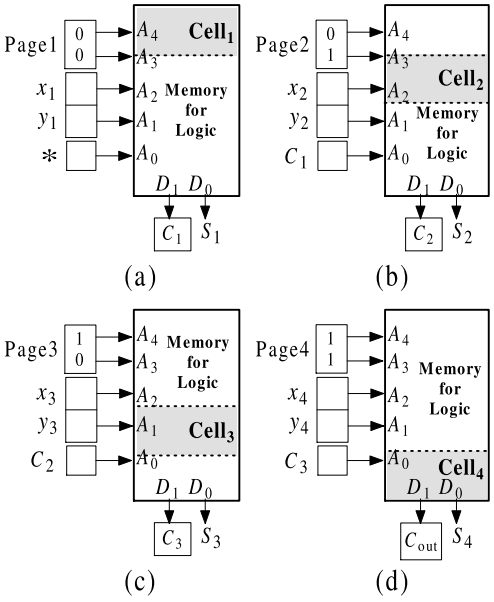

Fig. 5 Operation of 4-bit adder.

Although it is slower than the LUT cascade, its logical capability is much higher. In the LUT ring, the number of inputs and outputs per cell, as well as the number of cells, can be changed by using the programmable connection network.

Compared with the LUT cascade, the LUT ring shown in Fig. 3 has the following features:

- Requires only one memory for LUT data.
- Can implement a wide range of functions.
- Can benefit from memory-packing [5]–[7] by reducing the total amount of memory.

Example 2.1: Consider a 4-bit adder.

	x_4	x_3	x_2	x_1
+)	y_4	y_3	y_2	y_1
Cout	S_4	S_3	S_2	S_1

The adder can be implemented by an LUT cascade with four independent cells as shown in Fig. 4. $Cell_1$ has an external output S_1 and a rail output C_1. Note that the number of cells is four.

However, if we use an LUT ring, we need just one memory for LUT data. Figures 5(a)–(d) show the operation of the 4-bit adder by the LUT ring. Suppose that the memory for logic has five address lines $(A_4, A_3, A_2, A_1, A_0)$, and two output lines (D_1, D_0).

The LUT cascade consists of four cells. In this case, the LUT ring requires four memory accesses to compute

the outputs of the adder. We partition the memory into four pages, and assume that two most significant bits of the address denote the page. In the first memory access, the two most significant bits are set to (0,0), as shown in Fig. 5(a). This corresponds to the page 1 in the LUT ring, and the first cell in the LUT cascade. We have to read the memory to obtain C_1 and S_1. Since the first cell has only two inputs, the least significant bit of the address can be either 0 or 1. This operation is symbolically denoted by
$(A_4, A_3, A_2, A_1, A_0) \leftarrow (0, 0, x_1, y_1, *)$, where $*$ denotes either 0 or 1.

After reading the values of (C_1, S_1), C_1 is transferred to the least significant bit of the address for the next lookup. At the same time, S_1 is set to the least significant bit of the output register. This wiring is done by the programmable connection network in Fig. 3. The information for the connection for this memory access is stored in the memory for interconnections. This operation is symbolically denoted by
Read (D_1, D_0), and $(C_1, S_1) \leftarrow (D_1, D_0)$;
OUT_REG [0] $\leftarrow (S_1)$.

In the second memory access, the 2nd page is used to obtain (C_2, S_2) as shown in Fig. 5(b). This operation is denoted by
$(A_4, A_3, A_2, A_1, A_0) \leftarrow (0, 1, x_2, y_2, C_1)$;
Read (D_1, D_0), and $(C_2, S_2) \leftarrow (D_1, D_0)$;
OUT_REG [1] $\leftarrow (S_2)$.

In the third memory access, the 3rd page is used to obtain (C_3, S_3) as shown in Fig. 5(c). This operation is denoted by
$(A_4, A_3, A_2, A_1, A_0) \leftarrow (1, 0, x_3, y_3, C_2)$;
Read (D_1, D_0), and $(C_3, S_3) \leftarrow (D_1, D_0)$;
OUT_REG [2] $\leftarrow (S_3)$.

In the last memory access, the 4th page is used to obtain (C_{out}, S_4) as shown in Fig. 5(d). This operation is denoted by
$(A_4, A_3, A_2, A_1, A_0) \leftarrow (1, 1, x_4, y_4, C_3)$;
Read (D_1, D_0), and $(C_{out}, S_4) \leftarrow (D_1, D_0)$;
OUT_REG [4:3] $\leftarrow (C_{out}, S_4)$.

Note that the LUT ring emulates the LUT cascade in Fig. 4. In this way, the 4-bit adder is evaluated by accessing the memory four times. (End of Example)

The architecture of the LUT ring is quite different from that of FPGAs. To realize multiple-output functions, we can easily predict the evaluation time before physical design. In the LUT ring, from the number of the cells, the evaluation time for a logic function is easy to predict. The details will be discussed in the next section.

3. Circuit Design

This section presents a transistor-level design using 0.35 μm 3.3 V CMOS technology, and evaluates the delay of the prototype using the SPICE circuit simulator.

3.1 Circuit Design of the Prototype

The specification of the prototype of LUT ring is as follows:

- The number of primary inputs is at most 32.
- The number of primary outputs is at most 24.
- The number of cells in the LUT ring, s, is at most 8.
- The maximum number of inputs for each cell, k, is 13. The cells may have different numbers of inputs.
- The maximum number of outputs for each cell is 8.
- Memory-packing can be used to reduce memory.

Figure 6 shows the architecture of the LUT ring. It consists of the following components:
Input Reg.: 32-bit, stores the values of primary inputs.
Output Reg.2: 24-bit, stores the values of primary outputs.
MAR: 13-bit memory address register, stores the values of cell address.
Memory for Logic: 13 inputs and 8 outputs, 64 K-bit SRAM. Stores the LUT data for cells.
Memory for Interconnections: Stores the information on how to set the interconnections among cells.
Control Part: Includes a counter and an 3-bit multiplexer and generates the control signals for the LUT ring.
Shifter Network: Consists of an 32-by-13 Input Shifter, an 8-by-8 Feedback Shifter and 11 Input / Feedback Selectors. It implements the programmable interconnections among cells.
Shifting Register: Consists of an 8-by-8 Output Shifter1, 32-by-24 Output Shifter2 and 24-bit Output Reg.1. It stores the values of the external outputs of the different cells step by step.

The LUT ring has three important parts: The *Memory for Logic*, the *Shifter Network* and the *Shifting Register*. Since the *Memory for Logic* operates as a data path in the *LUT ring mode*, which is shown in 3.3.1, we use an asynchronous SRAM. Because design methods of SRAMs are described in literature [8], we just show the circuit design of the barrel shifter. The delay time of an n-bit shifter is proportional to $\log n$, so combined with fast transmission gates, shift can be fast [8]. Furthermore, we reduced chip area by using a single n-channel pass transistor gate instead of a full transmission gate. Figure 7 shows an example of the detailed circuit design of an 4-by-4 barrel shifter.

A novel circuit in the LUT ring is the *Shifting Register*, which stores the external outputs of cells in the *Output Reg.1*, step by step. The external outputs of a cell are produced when the corresponding page of the *Memory for Logic* is accessed.

A conventional method to write the output signals to the selected bits of the *Output Reg.1* is shown in Fig. 8. This method uses flip-flops with LOAD inputs. To load the signal to the flip-flops, the LOAD signal is set to 1. However, this method requires a control memory with size (the number of pages) $\times$ (the number of bits in the output registers)=8×24=192. In addition, it requires interconnections between the memory and registers. Therefore, we developed a novel method that uses *Output Shifter1*, *Output Shifter2* and *Output Reg.1*. Note that the outputs of *Output Reg.1* are fed back to the inputs of the *Output Reg.1* through *Output Shifter2*. As will be shown in Sect. 3.3, the size of

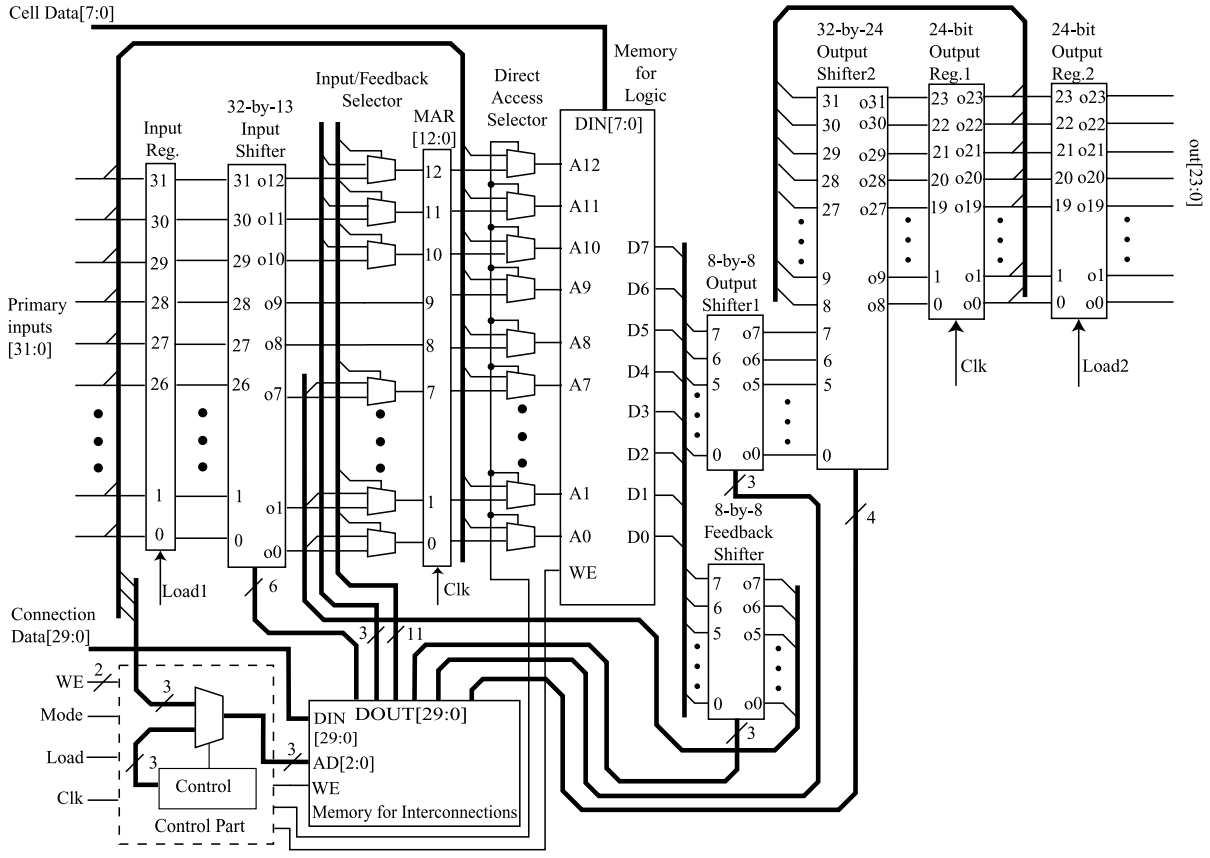

Fig. 6 Architecture of the LUT ring.

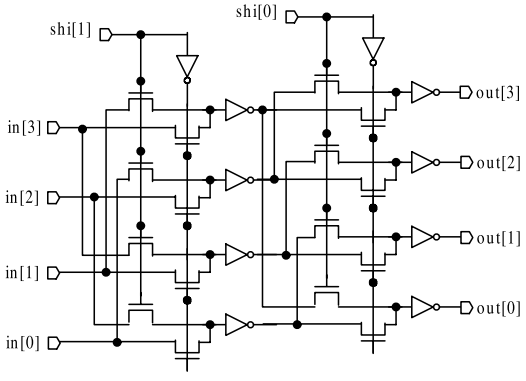

Fig. 7 Details of 4-by-4 barrel shifter.

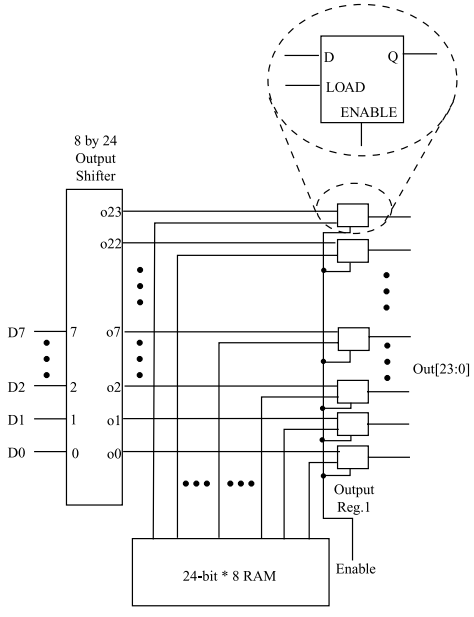

Fig. 8 Conventional method to store the outputs.

the control memory for two *Output Shifters* is 7×8=56 bits, much smaller than the size of control memory with conventional methods. Compared with conventional methods, this novel method is simple and easy to implement. The detail of its operations will be shown in 3.3.2.

3.2 Chip Size

With the 0.35 μm CMOS technology of Rohm Co. LTD, the size of 64 K-bit SRAM is 2080 μm by 2340 μm. The chip size excluding the 64 K-bit SRAM is roughly 700 μm by 900 μm. Thus, we can use the less expensive 4.9 mm by 4.9 mm chip of the VLSI Design and Education Center (VDEC).

3.3 Operation of the LUT Ring

3.3.1 Operation Modes in the LUT Ring

The LUT ring has two modes: The configuration mode and the LUT ring mode.

Configuration Mode:

To store the cell data of the LUT cascade to the *Memory for Logic*, we use the *Direct Access Selector*. By connecting the primary inputs[31:19] to the inputs of the address of the *Memory for Logic*, we load the cell data through DIN [7:0] into the *Memory for Logic*. Also, we store the connection data of the LUT cascade into the *Memory for Connections* by connecting the primary inputs[2:0] to the inputs of address of the *Memory for Connections*.

LUT Ring Mode:

In this mode, all the address lines of the *Memory for Logic* are connected to the MAR, while the address lines of the *Memory for Connections* are connected to the control block, and both memories are in the READ operation. Let the cascade consist of s cells, where $1 \leq s \leq 8$. In this case, the p most significant bits of the addresses of the *Memory for Logic* specify the page number, where $p = \lceil \log_2 s \rceil$.

To use the LUT ring, first the chip is set to the *Configuration Mode*: We need to store LUT data into the *Memory for Logic* and store the interconnection information into the *Memory for Connections*. Next, the chip is set to the *LUT Ring Mode*.

3.3.2 Detailed Operation in the LUT Ring Mode

The LUT ring will operate in four steps as follows:

1) Acquisition of the values of the primary inputs

We assume that the order of the primary inputs and the order of the variables in the cascade are the same. At time 0, the values of the primary inputs are sent into the Input Reg.

2) Set the data for the MAR

Assume that we are going to compute the i-th cell of the LUT cascade, where $1 \leq i \leq 8$. The inputs to the MAR are: the page signals (p bits), signals from the input register ($|X_i|$ bits), and signals from the memory (u_{i-1} bits), where $u_{i-1} + p + |Xi| \leq 13$. To make the argument simple, let $p + |X_i| + u_{i-1} = 13$.

Page number stored in the MAR[$12 : 12 - p + 1$]: In the MAR, the p most significant bits denote the page number. The *Input / Feedback Selector* chooses the signals from the *Memory for Connections*. The control signals for the *Input / Feedback selectors* are also stored in the *Memory for Connection*.

X_i inputs stored in the MAR[$12 - p : 12 - p - |X_i| + 1$]: The data for X_i are obtained as follows: When $i = 1$, the *Input Shifter* shifts 0-bit. When $i > 1$, the data of the input register are shifted by $|X_1| + |X_2| + \cdots + |X_{i-1}|$ bits by the *Input Shifter*. In this case, the *Input / Feedback Selector* chooses the signals from the *Input Shifter*.

Feedback inputs stored in the MAR[$12 - p - |X_i| : 12 - p - |X_i| - u_{i-1} + 1$]: Among the outputs of the *Memory for Logic*, we select the corresponding u_{i-1} bits of the *Feedback Shifter*, using the *Input / Feedback Selector*. To reduce the size of the *Memory for Logic* by memory packing [5], we use this *Feedback Shifter*.

3) Evaluation of function by the Memory for Logic

After setting the values to the MAR, by accessing the corresponding address of the *Memory for Logic*, we have the data D [7:0]. These data show the rail outputs to the next cell(u_i bits) and the external outputs of the i-th cell($|Y_i|$ bits).

4) Store the external outputs of cells into the Output Registers

The *Output Shifter1* shifts Y_i to the higher bit position. As a result, the signals are sent to the inputs (in[7:7-$|Y_i|$+1]) of the *Output Shifter2*. Next, the *Output Shifter2* shifts the data to the upper position by $|Y_i|$ bits. By this, we can set the data into the corresponding position of the *Output Reg.1*. At this time, the values that were stored in that positions are also shifted by $|Y_i|$ bit to the higher positions. The total number of bits in the control signals for the two *Output Shifters* is 7, thus, we need only $(7 \times s)$ bits. Note that the size of the control memory for the two *Output Shifters* is $7 \times 8 = 56$ bits when s is 8. When all the computations are finished, the values of the *Output Reg.1* are sent to the *Output Reg.2*.

In this way, we can get the desired result after four steps.

3.4 Operation Speed of the Prototype

To estimate the operation speed of the prototype, we partition the LUT ring into three parts, as shown in Fig. 9. The In Part consists of input pad (I Pad) and *Input Reg.*; the Out Part consists of output pad (O Pad) and *Output Reg.2*; and the others are represented as the Sequential Part dotted line. Figure 10 shows the delay time in the signal path, where the values are obtained by SPICE simulation for a 0.35 μm, 3.3 V CMOS process.

In Fig. 10, the delay for In Part is denoted by Delay1, and the delay for Out Part is denoted by Delay4. The delay for the sequential part depends on the number of clocks. In the prototype, during one clock cycle we can access the 64 K-bit SRAM once. Note that before the memory access,

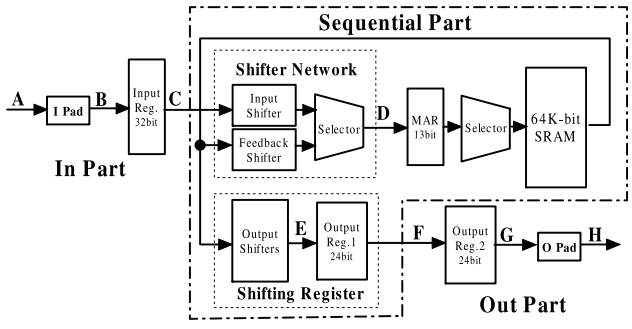

Fig. 9 Signal path of the prototype.

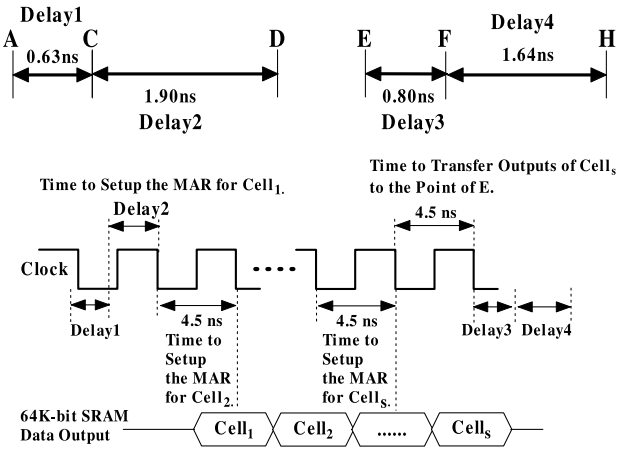

Fig. 10 Delay time in the signal path.

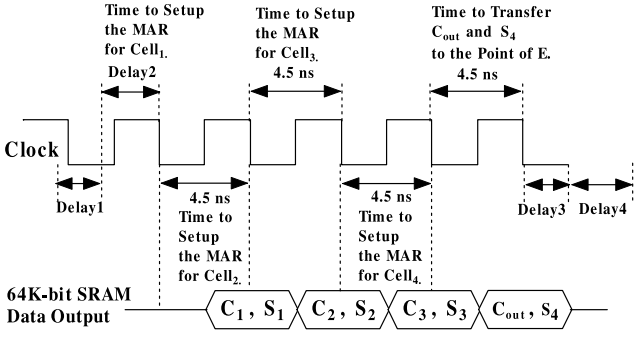

Fig. 11 Delay time for the 4-bit adder.

we need to setup the MAR with the values of cell address. The time to setup the MAR for the first cell is denoted by Delay2, which is shorter than the time for the rest of the cells because its path is C to D. The time to setup the MAR for the other cells is equal to one clock cycle. Thus, when the LUT cascade consists of s cells, the setup time of the MAR for these cells is equal to (Delay2 + (s-1) × CLK), where CLK is 4.5 ns. However, we also need one clock cycle to transfer the outputs of the last cell to the point E. After Delay3, the primary outputs are out of the sequential part. Therefore, the delay time for the sequential part is equal to (Delay2 + s × CLK + Delay3). Thus, the total delay time of the LUT ring is obtained by the following:

$$\begin{aligned}&\text{Delay Time}\\&= \text{Delay1} + \text{Delay2} + s \times \text{CLK} + \text{Delay3} + \text{Delay4}\\&= 0.63 + 1.9 + 4.5 \times s + 0.8 + 1.64\\&= 4.5 \times s + 5.0 \text{ (ns)} \qquad (1)\end{aligned}$$

Example 3.1: Consider the 4-bit adder in Example 2.1, where s is 4. The evaluation time is $4.5 \times 4 + 5.0$ (ns).

Figure 11 shows the distribution of the evaluation time for the 4-bit adder. First, after Delay1, the values of $(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$ are stored in the Input Reg. Second, before accessing the 64 K-bit SRAM, we need to spend Delay2 to setup the MAR with the values of (0, 0, 0, x_1, y_1, $*, *, *, *, *, *, *, *$) for $Cell_1$, where $*$ denotes either 0 or 1, to obtain C_1 and S_1. Then, to obtain C_2 and S_2, we need to spend one clock cycle to setup the MAR with the values of (0, 0, 1, x_2, y_2, C_1, $*, *, *, *, *, *, *$) for $Cell_2$, meanwhile S_1 is transferred to the point of E. During the next clock, the values of (0, 1, 0, x_3, y_3, C_2, $*, *, *, *, *, *, *$) have to be sent to the MAR for $Cell_3$ to obtain C_3 and S_3, meanwhile S_1 is stored in the *Output Reg.1*[0], and S_2 is transferred to the point of E. Similarly, to obtain C_{out} and S_4, the values of (0, 1, 1, x_4, y_4, C_3, $*, *, *, *, *, *, *$) have to be sent to the MAR for $Cell_4$, meanwhile S_2 is stored in the *Output Reg.1*[0] while S_1 is stored in the *Output Reg.1*[1], and S_3 is transferred to the point of E. During the next clock, S_3 is stored in the *Output Reg.1*[0] while S_1 and S_2 are stored in the *Output Reg.1*[2:1], then C_{out} and S_4 are transferred to the point of E. During Delay3, both C_{out} and S_4 are stored in the *Output Reg.1*[1:0] while S_1, S_2 and S_3 are stored in the *Output Reg.1*[4:2], and then the values of $(S_1, S_2, S_3, C_{out}, S_4)$ are transferred to the *Output Reg.2*. Finally, we can get the evaluated results after Delay4. (End of Example)

4. Performance Evaluation and Comparison with Other Devices

We evaluated the performance of the prototype, and compared it with microprocessors and FPGAs. To make the argument simple, we selected functions from MCNC combinational benchmark set [9].

Table 1 compares the performance of the prototype with commercial FPGAs and two microprocessors. In this table, "Name," "In," "Out," and "BDD size" denote the function name, the number of inputs, the number of outputs, and the number of non-terminal nodes in BDDs-for-CFs, respectively. The column "s" denotes the number of cells in the LUT ring. To obtain the number of cells, we used the newly developed logic synthesis tool reported in [10], where the number of inputs for each cell k is set to 10. The column "Time" denotes the evaluation time for the prototype estimated by Eq. (1) in Sect. 3.4, in nano-seconds.

4.1 Comparison with Microprocessors

At least two approaches exist to implement software for combinational benchmark functions. The first approach is to simulate the multi-level combinational circuit by some logic simulator. The second approach is to represent the function by a BDD, and then traverse the BDD by a special program [11], [12]. In this experiment, we used the second approach, since it is faster than the first approach for the benchmark functions. We represented benchmark functions using BDDs-for-CFs. The numbers of nodes in the BDDs-for-CFs were reduced by using a sifting algorithm [13]. Then, we generated a table that represents the BDD, and we used a special program to traverse the BDD data.

To compare the performance of the prototype with the speed for the software programs, we used a 32-bit RISC mi-

Table 1 Comparison of the LUT rings with MPUs and FPGAs.

Name	In	Out	BDD size	LUT rings		SH-1	PenIII	FPGA-30E		FPGA-100E		Relative speed of LUT rings			
				s	Time [ns]	[μs]	[μs]	LEs	Delay [ns]	LEs	Delay [ns]	SH-1	PenIII	30E	100E
misex2	25	18	205	5	27.5	41.9	0.36	35	13.2	34	16.4	137	59	0.48	0.60
in4	32	20	727	8	41.0	45.6	0.39	109	17.3	109	19.7	100	43	0.42	0.48
b3	32	20	649	7	36.5	45.3	0.37	103	18.4	103	20.2	112	46	0.50	0.55
t2	17	16	480	5	27.5	42.3	0.33	67	14.4	67	16.5	139	54	0.52	0.60
mlp6	12	12	4482	7	36.5	42.0	0.34	1063	27.3	1063	30.7	104	42	0.75	0.84
b2	16	17	642	6	32.0	43.0	0.33	216	25.9	216	30.0	121	46	0.81	0.94
chkn	29	7	224	5	27.5	24.3	0.22	178	24.8	178	27.3	80	36	0.90	0.99
bc0	26	11	561	5	27.5	31.9	0.25	417	24.0	417	28.1	105	41	0.87	1.02
gary	15	11	415	4	23.0	29.8	0.24	181	20.4	181	22.6	117	47	0.89	0.98
misex3	14	14	3544	4	23.0	38.6	0.30	263	23.4	263	26.6	151	59	1.02	1.15
apex4	9	19	2230	4	23.0	49.0	0.39	812	23.2	812	25.5	192	76	1.01	1.11
intb	15	7	756	3	18.5	32.3	0.25	393	36.2	393	40.7	157	61	1.96	2.20
exam	10	10	690	2	14.0	34.3	0.26	148	18.8	148	20.7	221	84	1.34	1.48
m4	8	16	496	3	18.5	39.8	0.30	204	19.9	204	26.5	194	73	1.08	1.43
prom2	9	21	4014	3	18.5	49.4	0.38	734	25.1	734	23.1	241	93	1.36	1.25
tial	14	8	775	3	18.5	34.0	0.27	443	33.4	443	33.8	166	66	1.81	1.83
amd	14	24	463	7	36.5	54.6	0.43	107	16.5	107	17.3	135	53	0.45	0.47
p1	8	18	954	6	32.0	46.2	0.38	91	14.0	91	15.3	130	53	0.44	0.48
x9dn	27	7	204	5	27.5	28.2	0.26	31	15.9	31	17.1	92	43	0.58	0.62

croprocessor SH-1 (SH7020) 20 MHz [14] and an Intel PentiumIII 1 GHz with a 256 KB cache [15]. On the one hand, SH-1 is an embedded MPU used in DVDs, navigation systems, digital cameras, etc. On the other hand, PentiumIII is a high-performance CPU for desktop PCs. For the SH-1, we compiled the software program using the SH C/C++-compiler [16] with the optimization option for speed, and obtained the CPU time using the SH simulator [17]. For the PentiumIII, we compiled the software program using the GNU C-compiler gcc with the -O2 option, and obtained the CPU time by executing it on the Linux operating system with 4 GB memory.

In Table 1, the column "SH-1" denotes the average CPU time per test vector for the software program on SH-1, in micro-seconds. When the number of inputs for a benchmark function is smaller than 17, we obtained the average CPU time for the function using an exhaustive test (i.e., 2^n test vectors, where n is the number of inputs). When the number of inputs for a benchmark function is larger than or equal to 17, we obtained the average CPU time for the function using 1,000,000 random test vectors. Similarly, the column "PenIII" denotes the average CPU time per test vector for the software program on the PentiumIII, in micro-seconds. The PentiumIII was too fast to obtain the average CPU time accurately using a small number of test vectors. Thus, for all benchmark functions, we used 1,000,000 random test vectors. In the last four columns, the column "SH-1" denotes the relative speed of the LUT rings to the speed of the SH-1, where the speed of the SH-1 with 222 MHz is set to 1. Similarly, the column "PenIII" denotes the relative speed of the LUT rings to the speed of the PentiumIII, where the speed of the PentiumIII with 222 MHz is set to 1.0. That is, they are calculated by

$$\text{LUT/SH-1} = \frac{\text{SH-1 time} \times 20\,\text{MHz}/222\,\text{MHz}}{\text{Delay time of LUT ring}},$$

$$\text{LUT/PenIII} = \frac{\text{PenIII time} \times 1\,\text{GHz}/222\,\text{MHz}}{\text{Delay time of LUT ring}}.$$

Note that 222 MHz is the clock frequency for the LUT ring.

Table 1 shows that the LUT rings are 80 to 241 times faster than the SH-1, and 36 to 93 times faster than the PentiumIII when the clock frequencies for the LUT ring and the MPUs are equal.

4.2 Comparison with FPGAs

We also implemented the same MCNC benchmark functions by two types of Altera APEX20KE series FPGAs [20]: EP20K30EFC144-1 and EP20K100EFC144-1. The two devices have the same package size 13 mm by 13 mm and are fabricated by a 1.8 V, 0.18 μm process. In this package (144-pin FineLine Ball-Grid-Array), the EP20K30EFC144-1 has the minimum number of logic elements while the EP20K100EFC144-1 has the maximum number of logic elements.

To implement a logic function, first, we reduced the dependency of the original representations for each benchmark function with the SIS tool [18] with *script.algebraic*, and then we converted it into Verilog HDL source code. Second, we used Synplify Pro (version: 7.3.3) [19] to synthesize the Verilog HDL code and generate Verilog Quartus Mapping files for the EP20K30EFC144-1 device and the EP20K100EFC144-1 device. Finally, we performed place, route and timing analysis by using Quartus (version 2000.09) [20]. Note that the logic optimization results by using Synplify Pro may be affected by the different choices of *script* command of SIS for some functions.

In Table 1, columns "LEs" denote the number of logic elements actually used in the FPGAs. Columns "Delay" denote the delay time for each benchmark function obtained by Quartus, in nano-seconds. In the last four columns, the columns "30E" and "100E" denote the relative speeds of the LUT rings to the speeds of the EP20K30EFC144-1 device and the EP20K100EFC144-1 device, respectively, where the speeds of FPGAs are set to 1.0. That is, they are calculated by

$$\text{LUT/FPGA} = \frac{\text{Delay time of FPGA}}{\text{Delay time of LUT ring}}.$$

4.2.1 Chip Size

The APEX20K Device Family Data Sheet [20] shows that EP20K30EFC144-1 has 1200 logic elements, and each logic element contains a four-input LUT. Consider the case of the function *mlp6*. In the FPGA, the number of used LEs is 1063 that almost exhausted the total LEs of EP20K30EFC144-1, while in the prototype of the LUT ring, the number of used cells is 7. It is interesting to compare the chip sizes of these implementations. Unfortunately, since the die size and detailed circuit design of the Altera FPGA device have not been published, exact comparison is difficult.

Here, we roughly compare the chip sizes of the prototype and EP20K30EFC144-1. To make the problem simple, we partition the area of the chip into two parts: "LUT Area" that is for the area of the LUTs, and "Other Area" that is for the remainder of a chip area except for the area of LUTs. First, consider the area for the LUT ring. In the prototype, LUT Area is equal to the size of the 64 K-bit SRAM, i.e. 2080 μm by 2340 μm, and Other Area is 700 μm by 900 μm. Let the relative area of 64 K-bit SRAM be 1.0, then we have the relative chip size of the prototype as follows:

$$1 + \frac{700 \times 900}{2080 \times 2340} \approx 1.13.$$

In the EP20K30EFC144-1 device, LUT Area is the sum of the area of LUTs in the logic elements. Note that the total LUTs in EP20K30EFC144-1 have 19200 bits, which corresponds to 29.3% of the 64 K-bit SRAM. If we use the same circuit with the same process technology, then the LUT Area in EP20K30EFC144-1 would be less than 1/3 of the area of the 64 K-bit SRAM. Actually, the LUTs in Altera APEX20KE series FPGAs use latches and multiplexors, while the SRAM in the LUT ring uses sense amplifiers. Assume that the LUT Area in EP20K30EFC144-1 is 1/3 of the area of the 64 K-bit SRAM. Let α be a fraction of the area spent for Other Area in EP20K30EFC144-1, where $0 < \alpha < 1$. Let the relative area of 64 K-bit SRAM be 1.0, then the relative chip size of EP20K30EFC144-1 is given by:

$$\text{LUT Area + Other Area} = \frac{1}{3} + \frac{\alpha}{1-\alpha} \cdot \frac{1}{3}.$$

In this case, the chip size of EP20K30EFC144-1 is larger than that of the prototype if $\alpha > 0.7$.

Generally, the routing area in the FPGAs occupies from 50% to over 90% of the total area [21]. Clearly, Other Area in EP20K30EFC144-1 is larger than the routing area. Moreover, the interconnections between the LUTs is more complex in the APEX 20KE series FPGA. Thus, we can assume that $\alpha > 0.7$ in EP20K30EFC144-1. That is, the chip size of EP20K30EFC144-1 is larger than that of the LUT ring if we use the same process technology.

Therefore, although the LUT Area for the prototype is larger than that for the EP20K30EFC144-1 device, the chip size of the prototype would be smaller than that of the FPGA by using the same process technology.

4.2.2 Speed

In Table 1, we can see that the delay time of EP20K30EFC144-1 and EP20K100EFC144-1 for the functions in the lower rows are almost the same, but are different for the upper rows, even if the two devices use the same number of logic elements. This shows that we cannot estimate the delay time of one device from the delay time of another device. Also, for each device there is no clear relationship between "LEs" and "Delay." Thus, for a given logic function that will be implemented by an FPGA, even if we know the number of LEs used and the delay time of the another FPGA in the same series, we cannot estimate the delay time. Although we can estimate the delay time of an FPGA from the logic level of the network generated by a logic synthesis tool, the estimation is not accurate. In fact, it is difficult to predict the delay time of the FPGA before place and route. However, for the LUT ring, it is easy to accurately estimate the delay from the number of cells used.

Although FPGAs with the fastest speed grade are used, Table 1 shows that the FPGAs are just two times faster than the prototype for seven functions but are also slower for seven functions. Note that the FPGA uses 0.18 μm CMOS technology, while the prototype uses 0.35 μm CMOS technology. If we use a 0.18 μm CMOS process, the LUT ring will be at least twice fast [22]. In spite of the disadvantage of the process technology, the LUT ring gives a performance competitive to the FPGAs.

5. Conclusions and Comments

In this paper, we have shown a realization of an LUT ring that implements multiple-output functions. The major advantage of this method is that the prediction of the delay time for a logic function is easy. We custom-designed a prototype of an LUT ring by using 0.35 μm CMOS technology. Our simulation results show that the LUT ring is 80 to 241 times faster than software programs on an SH-1 with the same clock frequency as the LUT ring, and 36 to 93 times faster than software programs on a PentiumIII with the same clock frequency. However, it is slightly slower than the commercial FPGAs for many benchmark functions.

We also implemented an LUT ring by an FPGA board and a memory board to confirm its operation. The details will be shown in a separate paper.

Acknowledgments

This research is partly supported by JSPS, the Grant in Aid for Scientific Research, MEXT, the Kitakyushu area innovative cluster project, and the Takeda Foundation.

References

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," Proc. International Conference on Computer-Aided Design, pp.408–412, Nov. 1995.
[2] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation," IEEE Trans. Comput., vol.C-35, no.8, pp.677–691, Aug. 1986.
[3] C. Lee, "Graph-based algorithms for Boolean function manipulation," Bell Syst. Tech. J., vol.19, pp.985–999, July 1959.
[4] T. Sasao and M. Fujita, ed., Representations of Discrete Functions, Kluwer, Academic Publishers, 1996.
[5] T. Sasao, M. Matsuura, and Y. Iguchi, "A cascade realization of multiple-output function for reconfigurable hardware," International Workshop on Logic and Synthesis(IWLS01), pp.225–230, Lake Tahoe, CA, June 2001.
[6] Y. Iguchi, T. Sasao, and M. Matsuura, "Realization of multiple-output functions by reconfigurable cascades," International Conference on Computer Design: VLSI in Computers & Processors (ICCD-2001), pp.388–393, Austin, TX, Sept. 2001.
[7] T. Sasao, M. Kusano, and M. Matsuura, "Optimization methods in look-up table rings," International Workshop on Logic and Synthesis (IWLS-2004), pp.431–437, Temecula, California, U.S.A., June 2004.
[8] N.H.E. Weste and K. Eshraghian, ed., Principles of CMOS VLSI Design: A Systems Perspective (second edition), Addision-Wesley Publishing, California, 1994.
[9] MCNC-Benchmark Set: http://www.cbl.ncsu.edu/www
[10] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," 41st Design Automation Conference, pp.428–433, San Diego, CA, USA, 2004.
[11] P.C. McGeer, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," Proc. International Conference on Computer-Aided Design, pp.402–407, Nov. 1995.
[12] S. Nagayama and T. Sasao, "Code generation for embedded systems using heterogeneous MDDs," 12th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2003), pp.258–264, Hirosima, Japan, April 2003.
[13] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," Proc. International Conference on Computer-Aided Design, pp.42–47, Nov. 1993.
[14] Hitachi SuperH 32-bit RISC CPU SH-1 (SH7020), Renesas Technology Co., http://www.renesas.com/eng/products/mpumcu/32 bit/sh/
[15] Intel Pentium III Processor, Intel Co., http://www.intel.com/products/desktop/processors/pentiumiii/
[16] Hitachi Embedded Workshop (HEW), SuperH RISC Engine C/C++ Compiler Package Ver. 6.0Ar2, Hitachi ULSI Systems Co., http://www.hitachi-ul.co.jp/MYICE/XSOFT/
[17] Hitachi Debugging Interface (HDI) for SH Series Simulator Ver. 5.01, Hitachi ULSI Systems Co., http://www.hitachi-ul.co.jp/MYICE/XSOFT/
[18] E.M. Sentovich et al., " SIS: A system for sequential circuit synthesis," Tech. Report, no.UCB/ERL M92/41, University of California, Berkeley, May 1992.
[19] http://www.synplicity.com
[20] http://www.altera.com
[21] J. Rose, R.J. Francis, D. Lewis, and P. Chow, "Architecture of field-programmable gate arrays: The effect of logic block functionality on area efficiency," IEEE J. Solid-State Circuits, vol.25, no.5, pp. 1217–1225, Oct. 1990.
[22] C. Mead and L. Conway, ed., Introduction to VLSI systems, Addision-Wesley Publishing, Reading, MA, 1980.

Hui Qin received the B.E. degree from Beijing University of Aeronautics and Astronautics, China, in 1994 and the M.E. degree from Kyushu Institute of Technology, Japan, in 2004. From 1994 to 2001, he worked at China Academy of Space Technology, involved in the design of China-Brazil Earth Resources Satellite. He received the Scientific Technology Progress Award for National Defense in China in 2000. Now he is a Ph.D. candidate at the Department of Computer Science and Electronics, Kyushu Institute of Technology. His current research interests include reconfigurable architecture, complex systems design, software synthesis.

Tsutomu Sasao received the B.E., M.E., and Ph.D. degrees in Electronics Engineering from Osaka University, Osaka Japan, in 1972, 1974, and 1977, respectively. He has held faculty/research positions at Osaka University, Japan, IBM T.J. Watson Research Center, Yorktown Height, NY and the Naval Postgraduate School, Monterey, CA. Now, he is a Professor of Department of Computer Science and Electronics, as well as the Director of the Center for Microelectronic Systems at the Kyushu Institute of Technology, Iizuka, Japan. His research areas include logic design and switching theory, representations of logic functions, and multiple-valued logic. He has published more than 9 books on logic design including, Logic Synthesis and Optimization, Representation of Discrete Functions, Switching Theory for Logic Synthesis, and Logic Synthesis and Verification, Kluwer Academic Publishers 1993, 1996, 1999, 2001 respectively. He has served Program Chairman for the IEEE International Symposium on Multiple-Valued Logic (ISMVL) many times. Also, he was the Symposium Chairman of the 28th ISMVL held in Fukuoka, Japan in 1998. He received the NIWA Memorial Award in 1979, and Distinctive Contribution Awards from IEEE Computer Society MVL-TC in 1987 and 1996 for papers presented at ISMVLs, and Takeda Techno-Entrepreneurship Award in 2001. He has served an associate editor of the IEEE Transactions on Computers. Currently, he was the Chairman of the Technical Committee on Multiple-Valued Logic, IEEE Computer Society. He is a Fellow of the IEEE.

Munehiro Matsuura was born in Kitakyushu City, Japan. He studied at the Kyushu Institute of Technology from 1983 to 1989, and received the B.E. degree from the University of the Air, in Japan, 2003. He has been working as a Technical Assistant at the Kyushu Institute of Technology since 1991. He has implemented several logic design algorithms under the direction of Professor Tsutomu Sasao. His interests include decision diagrams and exclusive-OR based circuit design.

Shinobu Nagayama was born in Kanagawa Japan, and received the B.S. and M.E. from Meiji University, Kanagawa Japan, in 2000 and 2002, respectively. He is now a doctoral student of Kyushu Institute of Technology. His research interest includes decision diagrams, software synthesis, and embedded systems.

Kazuyuki Nakamura received the B.E., M.E. and D.E. degrees in Electrical Engineering from Kyushu University, Fukuoka, Japan, in 1986, 1988 and 1998, respectively. He joined the Microelectronics Research Laboratories, NEC Corporation, Kanagawa, Japan, in 1988, where he was engaged in the research and development of High-speed ULSI circuits, especially for Memories and Communication LSIs. From 1994 to 1995, he was a visiting Scholar in the Computer Systems Laboratory at Stanford University, CA. Since 2001, he has been an associate professor of Center for Microelectronic Systems in Kyushu Institute of Technology, Iizuka, Japan. His recent research interests include design methodologies for high-performance circuits and signal integrity issues. Dr. Nakamura is a member of the IEEE Solid-state Circuits Society and the Institute of Electronics, Information and Communication Engineers of Japan.

Yukihiro Iguchi was born in Tokyo, and received the B.E., M.E., and Ph.D. degree in electronic engineering from Meiji University, Kanagawa Japan, in 1982, 1984, and 1987, respectively. He is now an associate professor of Meiji University. His research interest includes logic design, switching theory, and reconfigurable systems. In 1996, he spent a year at Kyushu Institute of Technology. He received Takeda Techno-Entrepreneurship Award in 2001.