A Machine to Evaluate Decomposed Multi-Terminal Multi-valued Decision Diagrams for Characteristic Functions

Hiroki Nakahara*, Tsutomu Sasao†, and Munehiro Matsuura†
*Kagoshima University, Japan, †Kyushu Institute of Technology, Japan

***Abstract*—A decomposed multi-terminal multi-valued decision diagrams for characteristic function (MTMDDs for CF) represents decomposed circuits. It can represent complex functions compactly. This paper shows a machine that evaluates decision diagrams. First, we introduce the decomposed MTMDDs for CF. Then, we consider two instructions to evaluate the decomposed MTMDDs for CF. Next, we show a machine that evaluates the decision diagrams. We compare that machine with embedded processors. As for the power-delay product, our machine running at 100 MHz is 60.84 times smaller than Nios II processor running at 100 MHz, and it is 18.66 times smaller than Atom N455 processor running at 1.67 GHz.**

I. Introduction

Before the era of deep submicron, the performance of micro processors was the major concern, and it was improved by the scaling down of device dimensions. However, in the era of deep submicron, the power-efficiency is the major concern of the modern processors [4]. To improve the power-efficiency, this paper proposed a decision diagram machine (DDM) [7], [1] which evaluates decision diagrams (DDs). A heterogeneous multi-valued decision diagram (HMDD) may have nodes with different number of binary variables for different super variables [10]. Since the HMDD can use the optimal partition of the input variables to reduce the path length, the evaluation time of the HMDD is shorter than that of the corresponding binary decision diagram (BDD) [15]. We showed that, the HMDD machine is faster and dissipates lower power than the conventional processor [12]. Applications for the HMDD machine include a sequencer [22]; an accelerator for a logic simulation [5]; and a packet classifier [14]. However, for general applications including arithmetic circuits, since the number of the nodes in a conventional HMDD increases exponentially with the number of inputs, such HMDDs are difficult to represent, when the number of inputs is large [20]. Thus, the conventional DDM is unapplicable for general purpose.

To represent a logic circuit by DDs with a reasonable size, we proposed *decomposed MTMDDs for CF* representing a decomposed circuit [11]. With an enough size of memory, the DDM for the decomposed MTMDD is faster than that for the conventional HMDD. When the memory was expensive, it was difficult to use a memory with large size. However, a memory with more than one Giga bytes is available at a low price today. Therefore, adopting the DDM for *the decomposed HMDDs for CF* with a large memory is practical.

In this paper, we realize DDM for the decomposed HMDDs for CF using an FPGA and an off-chip SRAM. Also, we compare the proposed one with embedded processors with respect to delay time, power consumption, and power-delay product.

The rest of the paper is organized as follows: Chapter 2 shows definitions; Chapter 3 introduces the multi-terminal multi-valued decision diagrams for characteristic function representing a decomposed circuit (decomposed MTMDDs for CF); Chapter 4 realizes the DDM for decomposed MTMDDs for CF; Chapter 5 shows the experimental results; and Chapter 6 concludes the paper.

II. Definition of Decision Diagrams

A. Decision Diagram (DD)

Definition 2.1: A **binary decision diagram (BDD)** is obtained by applying **Shannon expansions** repeatedly to a logic function f [2]. Each non-terminal node labeled with a variable x_i has two outgoing edges which indicate nodes representing cofactors of f with respect to x_i. When the Shannon expansions are performed with respect to k variables, all the non-terminal nodes have 2^k edges. In this case, we have a **multi-valued decision diagram (MDD(k))** [6].

Definition 2.2: In a DD, a sequence of edges and non-terminal nodes leading from the root node to a terminal node is a **path**. **An ordered BDD (OBDD)** has the same variable order on any path. **A reduced ordered BDD (ROBDD)** is derived by applying as follows:

1. Share equivalent sub-graphs.
2. If all the outgoing edges of a non-terminal node v point the same succeeding node u, then delete v and connect the incoming edges of v to u.

An **ROMDD(k)** can be defined similarly to the ROBDD. Note that, an MDD(1) means a BDD. In this paper, a BDD and an MDD(k) mean an ROBDD and an ROMDD(k), respectively, unless stated otherwise.

If the evaluation time for all the nodes are the same, then the average evaluation time for an HMDD is proportional to the **average path length (APL)** [3].

B. Representation of Multi-output Logic Function Using Decision Diagrams for Characteristic Function

Many practical applications use multiple-output functions. Here, we represent an n-input m-output logic function using

0195-623X/13 $26.00 © 2013 IEEE
DOI 10.1109/ISMVL.2013.6

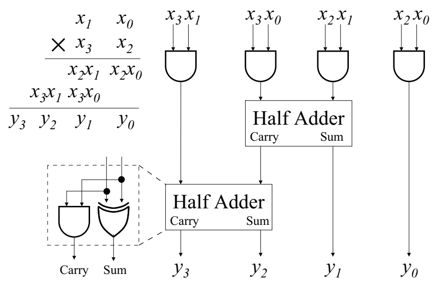

Fig. 1. Circuit for a 2-bit multiplier.

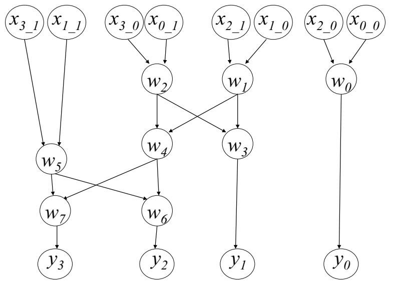

Fig. 2. DAG representing the multiplier shown in Fig. 1.

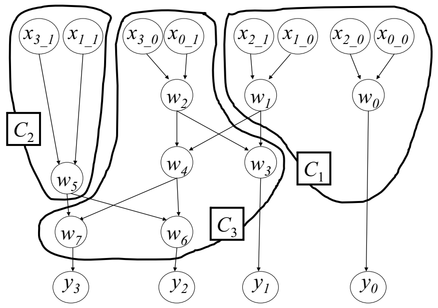

Fig. 3. An example of a cluster decomposition.

a decision diagram (DD).

Definition 2.3: Let $\vec{X} = (x_1, x_2, \ldots, x_n)$ be the input variables, $\vec{Y} = (y_1, y_2, \ldots, y_m)$ be the output variables, and $\vec{f} = (f_1(\vec{X}), f_2(\vec{X}), \ldots, f_m(\vec{X}))$ be a multiple-output function. **The characteristic function (CF) of a multiple-output function** is $\chi(\vec{X}, \vec{Y}) = \bigwedge_{i=1}^{m} (y_i \equiv f_i(\vec{X}))$.

The characteristic function of an n-input m-output function is a two-valued logic function with $(n+m)$ inputs. It has input variables x_i $(i = 1, 2, \ldots, n)$, and output variables y_j for outputs f_j. Let $B = \{0,1\}$, $\vec{a} \in B^n$, $\vec{F} = (f_1(\vec{a}), f_2(\vec{a}), \ldots, f_m(\vec{a})) \in B^m$, and $\vec{b} \in B^m$. Then, the characteristic function satisfies the relation

$$\chi(\vec{a}, \vec{b}) = \begin{cases} 1 & (when\ \vec{b} = \vec{F}(\vec{a})) \\ 0 & (otherwise) \end{cases}$$

Definition 2.4: **A support variable** of a function f is a variable on which f actually depends.

Definition 2.5: [19] **A multi-terminal binary decision diagram for characteristic function (MTBDD for CF)** of a multiple-output function $\vec{f} = (f_1, f_2, \ldots, f_m)$ represents the characteristic function χ. We assume that the root node is in the top of the MTBDD, and the variable y_i is below the support variable of f_i, where y_i is the variable representing f_i.

III. Decomposed Multi-Terminal Multi-valued Decision Diagrams for Characteristic Function

A. Cluster Decomposition of a Circuit

In this paper, a combinational circuit is represented by **a directed acyclic graph (DAG)**. In a DAG, **a primary input node** denotes a primary input; **a primary output node** denotes a primary output; and **an intermediate node** denotes a 2-input logic gate[1]. When the output of a logic gate i is connected to a logic gate j, the DAG has an edge from a node i to a node j. In the DAG, we assume that a primary input does not fan-out. Thus, in general, we need to duplicate the input variables. Therefore, the number of the primary input nodes can be greater than the number of the primary inputs. We denote an input node by *(primary input name)_(unique number)*.

[1]A NOT gate is converted into a NAND gate having the same inputs. Also, a gate with more than two inputs is decomposed into multiple gates with two inputs.

Example 3.1: Fig. 1 shows the circuit for a two-bit multiplier, where $\{x_0, x_1, x_2, x_3\}$ denotes the primary inputs and $\{y_0, y_1, y_2, y_3\}$ denotes the primary outputs. Fig. 2 shows the DAG for the multiplier in Fig. 1, where $X = \{x_{0\_0}, x_{0\_1}, x_{1\_0}, x_{1\_1}, x_{2\_0}, x_{2\_1}, x_{3\_0}, x_{3\_1}\}$ denotes the primary input nodes; $Y = \{y_0, y_1, y_2, y_3\}$ denotes the primary output nodes; and $W = \{w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$ denotes the intermediate nodes. ■

Definition 3.6: **A cut** (S, T) is a partition of nodes in the DAG. For $s \in S$ and $t \in T$, no node of T has an edge directed to any node of S.

Definition 3.7: Let X be the set of primary input nodes of the DAG. The set of nodes that depend on X is denoted by $D(X)$. Note that, $D(X)$ includes the primary input nodes X, while it does not include the primary output nodes Y.

Example 3.2: In Fig. 2, let $X = \{x_{1\_0}, x_{2\_1}, x_{0\_1}, x_{3\_0}\}$ be a subset of primary input nodes of the DAG. In this case, $D(X) = \{x_{1\_0}, x_{2\_1}, x_{0\_1}, x_{3\_0}, w_1, w_2, w_3, w_4\}$. ■

Definition 3.8: Let V be a set of all the nodes in the DAG; $(X_1, X_2, X_3, ..., X_q)$ be a partition of the primary input nodes X, where $X_i \cap X_j = \emptyset$ $(i \neq j)$; and Y be the set of the primary output nodes. **A cut set with topological order** $\{(S_i, T_i) | i = 1, 2, \ldots, q\}$ is the set satisfying:

1. For first cut (S_1, T_1), $S_1 = D(X_1)$ and $T_1 = V - S_1$.
2. For i-th cut (S_i, T_i), $S_i = D(X_1 \cup X_2 \cup \ldots \cup X_i)$ and $T_i = V - S_i$, where $1 < i < q$.
3. For q-th cut (S_q, T_q), $S_q = D(X)$ and $T_q = Y$.

When a partition of the primary input nodes is given, the cut set with topological order is uniquely determined. By evaluating a set S_i $(i = 1, 2, \ldots, q)$ in order, we can evaluate the DAG.

Definition 3.9: Suppose that the DAG is decomposed into a cut set with topological order $\{(S_i, T_i) | i = 1, 2, \ldots, q\}$. Then, $C_i = S_i - S_{i-1}$ is **a cluster**, where $S_0 = \emptyset$. A decomposition of the DAG into a set of clusters $\{C_1, C_2, \ldots C_q\}$ is **a cluster decomposition**.

Example 3.3: Fig. 3 shows an example of a cluster decomposition, where $C_1 = \{x_{0\_0}, x_{2\_0}, x_{1\_0}, x_{2\_1}, w_0, w_1\}$, $C_2 = \{x_{1\_1}, x_{3\_1}, w_5\}$, and $C_3 = \{x_{0\_1}, x_{3\_0}, w_2, w_4, w_6, w_7, w_3\}$. ■

In a set of clusters $\{C_1, C_2, \ldots, C_q\}$, when $i < j$, the cluster C_i may have edges directed to the cluster C_j, while the cluster C_j does not have edges directed to the cluster C_i.

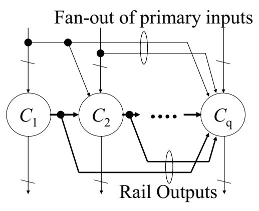

Fig. 4. Circuit realizing the cluster decomposition.

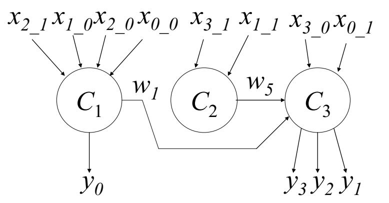
$x_{2\_1}\ x_{1\_0}\ x_{2\_0}\ x_{0\_0}\ x_{3\_1}\ x_{1\_1}\ x_{3\_0}\ x_{0\_1}$
C_1
w_1
C_2
w_5
C_3
y_0
$y_3\ y_2\ y_1$

Fig. 5. Circuit realizing the cluster decomposition of 2-bit multiplier.

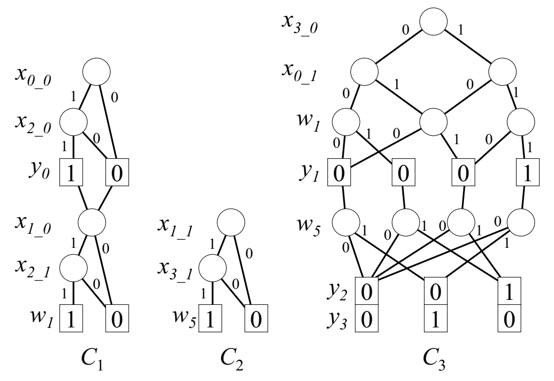

Fig. 6. MTBDDs for CF representing clusters shown in Fig. 5.

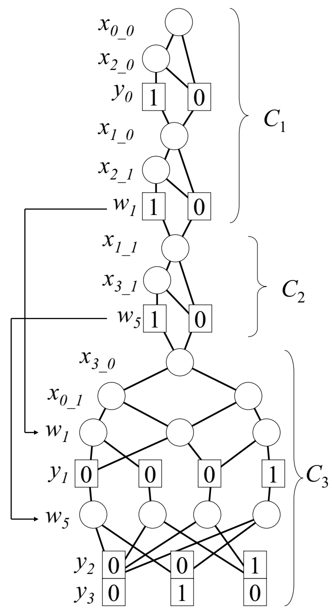

Fig. 7. An example of the decomposed MTBDDs for CF.

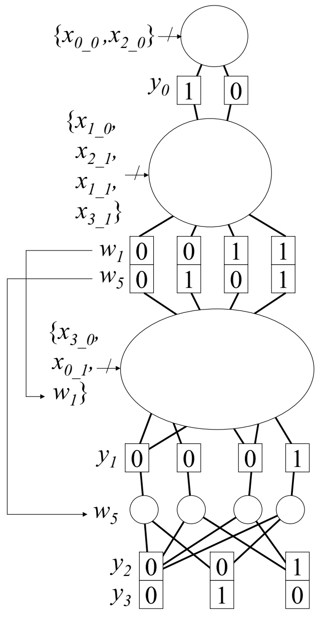
$\{x_{0\_0}, x_{2\_0}\}$
y_0
$\{x_{1\_0}, x_{2\_1}, x_{1\_1}, x_{3\_1}\}$
w_1
w_5
$\{x_{3\_0}, x_{0\_1}, w_1\}$
y_1
w_5
y_2
y_3

Fig. 8. An example of the decomposed MTMDDs for CF.

Definition 3.10: Fig. 4 shows the circuit representing a set of clusters $\{C_1, C_2, \ldots C_q\}$. Let $i < j$. The outputs of the cluster C_i directed to the cluster C_j are **rail outputs**, and the inputs of the cluster C_j directed from the cluster C_i are **rail inputs**.

Example 3.4: Fig. 5 shows the circuit realizing the cluster decomposition of the two-bit multiplier shown in Fig. 3. ■

B. MTBDDs for CF Representing a Cluster Decomposition

In a set of clusters $\{C_1, C_2, \ldots, C_q\}$, C_i is represented by an MTBDD for CF, where the inputs consist of primary inputs and rail inputs, and the outputs consist of primary outputs and rail outputs.

Example 3.5: Fig. 6 shows MTBDDs for CF representing clusters shown in Fig. 5. ■

Definition 3.11: Let $\{C_1, C_2, \ldots, C_q\}$ be a set of clusters, where C_i is represented by an MTBDD for CF. The **MTBDDs for CF representing a cluster decomposition (decomposed MTBDDs for CF)** is obtained by connecting MTBDDs for CF in the topological order of $\{C_1, C_2, \ldots, C_q\}$.

Example 3.6: Fig. 7 shows the decomposed MTBDDs for CF representing the two-bit multiplier in Fig. 3. ■

C. Decomposed MTMDDs for CF

A decomposed MTBDDs for CF can be extended to a **decomposed multi-valued decision diagrams for CF (decomposed MTMDDs for CF)**. To reduce its evaluation time, the DDM for decomposed MTMDDs for CF can use more memory than that for the decomposed MTBDDs for CF.

Example 3.7: Fig. 8 shows the decomposed MTMDDs for CF converted from the MTBDDs for CF shown in Fig. 7. Note that, the decomposed MTMDDs for CF is obtained by merging the cluster C_1 and a part of cluster C_2, and by changing the variable order of a node. ■

As shown in Example 3.7, in MTBDDs for CF, multiple clusters are merged and the variable order of a node is changed to obtain MTMDDs for CF.

Note that, the decomposed MTBDDs for CF lost canonicity of the BDD. Thus, the application of decomposed MTBDDs for CF to formal verification is difficult. However, the application to logic evaluation is straightforward, since the canonicity is not used.

IV. DDM for Decomposed MTMDDs for CF

A. Instructions to Evaluate the Decomposed MTMDDs for CF [13]

In a DDM, for a non-terminal node, we use the indirect branch instruction shown in Fig. 9. The index and branch addresses are stored in the separated words to use the memory efficiently as shown in Fig. 10. For a non-terminal node, first, the current index is read. Then, the jump address corresponding to the value of the current input variables is read. To evaluate a terminal node, we use **the output and jump instruction** shown in Fig. 11. To evaluate a terminal node, first, the output value is read. Then, the jump address is read. Since we use only two instructions, only one bit field is necessary to distinct them.

B. Interconnections Among the Decomposed MTMDDs for CF

Since the decomposed MTMDDs for CF allows fan-outs of both the primary inputs and the rail outputs, interconnections

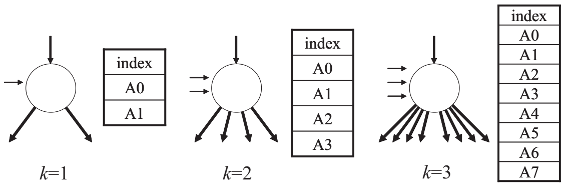

Fig. 9. Example of indirect branch instruction.

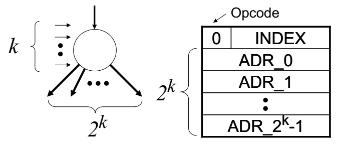

Fig. 10. Indirect branch instruction.

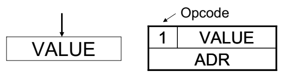

Fig. 11. Output and jump instruction.

are necessary. They are implemented by the random logic in the FPGA. Since the most parts of the circuit is realized by the nodes of the decomposed MTMDDs for CF, hardware for the interconnections tends to be small. Thus, the interconnections can be realized with a small FPGA. Experimental results will demonstrate this.

C. DDM for Decomposed MTMDDs for CF

Fig. 12 shows the DDM for a decomposed MTMDDs for CF. **The instruction memory** stores the instructions; **the instruction register** stores the instruction from the instruction memory; **the program counter (PC)** retains the address for the instruction memory; **the interconnection module** selects both the primary inputs and the rail inputs; and **the double-rank shift register** retains the output value. Fig. 13 shows the double-rank shift register consisting of **double-rank flip-flops** [18]. The shift register retains outputs of the decomposed MTMDDs for CF by the C_Clock. When all outputs are evaluated, values are sent to the primary outputs by the S_Clock.

The following algorithms show the execution of the indirect branch instruction and the output and jump instruction.

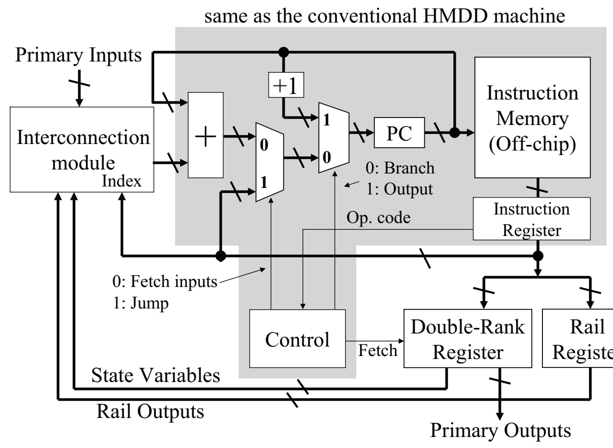

Fig. 12. DDM for Decomposed MTMDDs for CF.

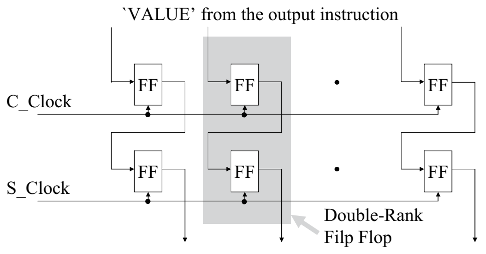

Fig. 13. Double-rank shift register.

Algorithm 4.1: (2^k indirect branch instruction)

```
1. Obtain indirect branch address
   1.1 Read the index corresponding to the index filed of
       the branch instruction.
   1.2 Add it to the PC.
2. Perform the jump operation
   2.1 Read the jump address corresponding to the PC.
   2.2 Set the jump address to the PC.
3. Terminate.
```

Algorithm 4.2: (Output and jump instruction).

```
1. Output the value.
   1.1 Read the output value and the jump address corre-
       sponding to the PC.
   1.2 Set the value to the double-rank register.
   1.3 If all outputs are evaluated, then send them to the
       primary outputs.
2. Perform the jump operation, similarly to the Step 2 of
   Algorithm 4.1.
3. Terminate.
```

Example 4.8: The left column of Fig. 14 shows the interconnections for the decomposed MTMDDs for CF shown in Fig. 8.

1. To evaluate the root node, select primary inputs x_0 and x_2. Then, set the primary output y_0 to the double-rank register (Fig. 14 (1)).
2. To evaluate the next node, select primary inputs x_1, x_2, and x_3. Then, set rail outputs w_1 and w_5 to the rail register (Fig. 14 (2)).
3. To evaluate the next node, select primary inputs x_0 and x_3, and the rail input w_1. Then, set the primary output y_1 to the double-rank register (Fig. 14 (3)).
4. To evaluate the leaf node, select the rail input w_5. Then, set primary outputs y_2 and y_3 to the double-rank register. Since all the outputs are evaluated, the double-rank register send them to the primary outputs (Fig. 14 (4)).

■

V. Experimental Results

A. Comparison with Embedded Processors

We implemented the DDM for decomposed MTMDDs for CF on the Altera Cyclone III starter kit (FPGA: Cyclone III, EP3C25). For the FPGA synthesis tool, we used

TABLE I
COMPARISON OF PROCESSORS.

Name	In	Out	FF	APL	DDM for MTMDDs for CF@100MHz				Atom N455@1.67GHz		Nios II@100MHz	
					#LEs	Max. Freq. [MHz]	Delay [nsec]	PowerDelay [Wnsec]	Delay [nsec]	PowerDelay [Wnsec]	Delay [nsec]	PowerDelay [Wnsec]
s641	37	23	14	30.0	421	323	600	562.5	1450	13268.5	10392	45148.2
s713	37	23	14	30.3	371	363	607	569.0	1370	12535.5	11050	48006.6
s820	20	19	5	49.9	425	332	998	935.2	1610	14731.5	9100	39536.0
s1196	16	14	18	99.1	636	264	1983	1858.6	2300	21045.0	24974	108497.0
s5378	35	49	164	343.8	1471	186	6877	6444.6	12330	112819.5	43825	190397.0
s9234	36	39	211	151.8	874	222	3037	2845.7	4740	43371.0	51223	222535.8
s38417	28	106	1636	2010.5	3577	129	40210	37677.2	103850	950227.5	553705	2405513.6
Ratio							1.00	1.00	1.91	18.66	13.12	60.84

QuartusII (v.9.1). We compare it with the Intel's Atom processor running at 1.67 GHz and the Altera's Nios II embedded processor running at 100 MHz. We obtained the delay time per one random test vector [nsec/work] as shown in Table I using MCNC benchmark functions [21]. To obtain the decomposed MTBDDs for CF, first, we generated the netlist from the Verilog-HDL description by using Quartus II logic synthesis tool ver.9.1 with area minimization option. Then, we decomposed the netlist into clusters by using the greedy algorithm [16]. Next, we converted clusters into the decomposed MTBDDs for CF. To reduce the number of nodes in MTBDDs, we used the sifting algorithm [17]. To obtain the decomposed MTMDDs for CF, first, we set the memory size limitation to 1 Mega bytes (MB). Then, we converted the decomposed MTBDDs for CF into an MTMDDs for CF by using the dynamic programming [9]. To obtain delay time for the Nios II processor and the Atom N455 processor, we generated C-code for the decomposed MTMDDs for CF. Then, we generated the executable code using gcc compiler with optimize option -O3. The delay time of the DDM for the decomposed MTMDDs for CF is 1.91 times shorter than that of the Atom processor, and it is 13.12 times shorter than that of the Nios II processor.

We measured the power consumption [W] to obtain the power-delay product. The power-delay product corresponds to the energy [Wnsec] to evaluate the function. To make the comparison fair, we tried to make the temperature of the DDM for decomposed MTMDDs for CF, the Nios II processor, and the Atom N455 processor the same. A linear shift register implemented on the FPGA was used to generate random test vectors. As for the DDM for decomposed MTMDDs for CF running at 100 MHz, we assume that the power consumption was the total power consumption for the Altera's Cyclone III starter kit and the off-chip SRAM board. The power-supply voltage of the DDM for the decomposed MTMDDs for CF was 9.017 [V], and the measured current was 0.104 [A]. Thus, the power consumption for the DDM for the decomposed MTMDDs for CF is 0.937 [W]. As for the Nios II processor running at 100 MHz, we implemented it on the same platform as the DDM for the decomposed MTMDDs for CF. The power-supply voltage was 9.017 [V], and the measured current was 0.481 [A]. Thus, the power consumption for the Nios II processor was 4.344 [W]. As for the Atom processor, to obtain the power consumption, we measured Atom N455 embedded evaluation board. To obtain power consumption for the Atom processor, we turned off the display, disconnected the network, and suspended applications except for the kernel and the clock counter for measurement of the delay time. The power-supply voltage was 5.000 [V], and the measured current was 1.830 [A]. Thus, the power consumption was 9.150 [W]. Table I compares power-delay products. The power-delay product of the DDM for the decomposed MTMDDs for CF is 18.66 times lower than that of the Atom processor, and it is 66.84 times lower than that of the Nios II processor. Thus, the DDM for the decomposed MTMDDs for CF is more power-efficient than existing embedded processors.

VI. CONCLUSION

This paper showed the DDM for the decomposed MTMDDs for CF. To evaluate a non-terminal node, it uses an indirect branch instruction, while to evaluate a terminal node, it uses an output and jump instruction. The FPGA realizes the interconnections and the controller, while the SRAM stores the instructions. As for the speed, the DDM for the decomposed MTMDDs for CF is 1.91 times faster than the Atom processor, and is 13.12 times faster than the Nios II processor. Also, as for the power-delay product, the DDM dissipates 18.66 times lower energy than the Atom processor, and dissipates 66.84 times lower energy than the Nios II processor. Thus, the DDM for the decomposed MTMDDs for CF is more power-efficient than existing embedded processors.

VII. ACKNOWLEDGMENTS

This research is supported in part by the Grants in Aid for Scientific Research of JSPS. Discussions with Prof. J. T. Butler and Prof. Shinobu Nagayama were quite useful.

REFERENCES

[1] R. T. Boute, "The binary-decision machine as programmable controller," *Euromicro Newsletter,* Vol. 1, No. 2, pp. 16-22, 1976.

[2] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Comput.*, Vol. C-35, No. 8, Aug. 1986, pp. 677-691.

[3] J. T. Butler, T. Sasao, and M. Matsuura, "Average path length of binary decision diagrams," *IEEE Trans. on Compt.*, Vol. 54, No. 9, Sep. 2005, pp. 1041-1053.

[4] P. P. Gelsinger, "Microprocessors for the new millennium: Challenges, opportunities, and new frontiers," *ISSCC ' 01*, pp. 22-25.

[5] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno, "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference (ASPDAC2000)*, Jan., 26-28, Yokohama, Japan, pp.73-76.

[6] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and applications," *Multiple-Valued Logic*, Vol.4, no.1-2, 1998, pp.9-62.

[7] D. Mange, "A high-level-language programmable controller: Part I-II," *IEEE Micro,* Vol. 6, No. 1, pp. 25-41 (Part I), Vol. 6, No. 2, pp. 47-63 (Part II), Feb/Mar, 1986.

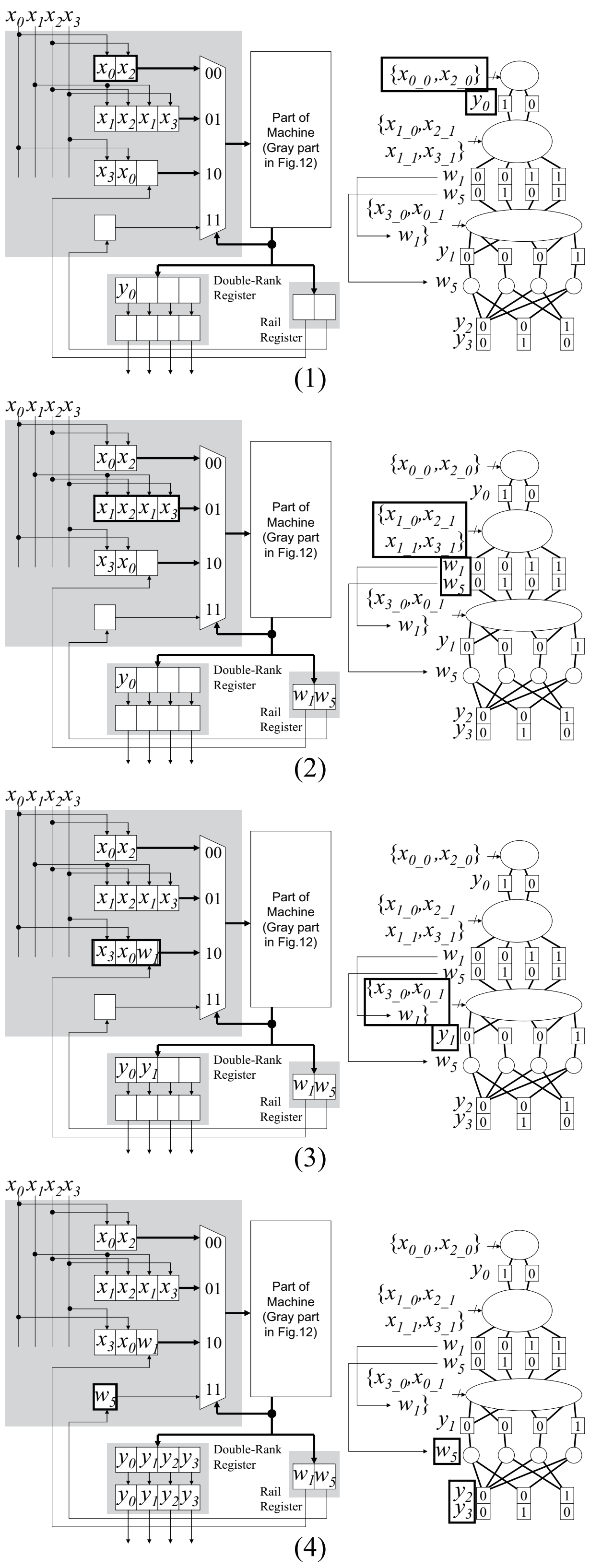

Fig. 14. Example of the input selector.

[8] M. Matsuura, T. Sasao, J. T. Butler, and Y. Iguchi, "Bi-partition of shared binary decision diagrams," *IEICE Trans. on Fund. of Electronics*, Vol.E85-A, No.12, Dec. 2002, pp.2693-2700.

[9] S. Nagayama and T. Sasao, "On the optimization of heterogeneous MDDs," *IEEE Trans. on CAD*, Vol.24, No.11, Nov., 2005, pp.1645-1659.

[10] S. Nagayama and T. Sasao, "Compact representations of logic functions using heterogeneous MDDs," *33rd IEEE Int'l Symp. on Multiple-Valued Logic (ISMVL2003)*, May, 2003, pp.247-255.

[11] H. Nakahara, T. Sasao, and M. Matsuura, "Multi-terminal multi-valued decision diagrams for characteristic function representing cluster decomposition," *The 42nd IEEE International Symposium on Multiple-Valued Logic (ISMVL 2012)*, May 14-16, 2012, pp.148-153.

[12] H. Nakahara, T. Sasao, and M. Matsuura, "A low power-delay product processor using multi-valued decision diagram machine," *The 17th Workshop on Synthesis And System Integration of Mixed Information Technologies (SASIMI 2012)*, March 8-9, 2012, pp.394-395.

[13] H. Nakahara, T. Sasao, and M. Matsuura, "A comparison of multi-valued and heterogeneous decision diagram machines," *Journal of Multiple-Valued Logic*, Vol. 19, No.1-3, pp.203-217.

[14] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," *13th EUROMICRO Conf. on Digital System Design (DSD-2010)*, Lille, France, Sept., 2010, pp.745-752.

[15] H. Nakahara, T. Sasao and M. Matsuura, "A comparison of architectures for various decision diagram machines," *ISMVL2010*, Barcelona, Spain, May, 26-28, 2010, pp.229-234.

[16] H. Nakahara, T. Sasao, and M. Matsuura, "A PC-based logic simulator using a look-up table cascade emulator," *IEICE Trans. on Fund. of Electronics, Communications and Computer Sciences*, Vol. E89-A, No. 12, Dec., 2006, pp. 3471-3481.

[17] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *IEEE/ACM Int'l Conf. on Computer-Aided Design (ICCAD1993)*, Nov., 1993, pp. 42-47.

[18] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, "Realization of sequential circuits by look-up table ring," *The 2004 IEEE Int'l Midwest Symp. on Circuits and Systems (MWSCAS 2004)*, Hiroshima, July 25-28, 2004, pp.I:517-I:520.

[19] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," *41st Design Automation Conference (DAC2004)*, June, 2004, pp. 428-433.

[20] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[21] S. Yang, "Logic synthesis and optimization benchmark user guide version 3.0," *MCNC*, Jan. 1991.

[22] P.J.A.Zsombor-Murray, L.J. Vroomen, R.D. Hudson, Le-Ngoc Tho, and P.H. Holck, "Binary-decision-based programmable controllers, Part I-III," *IEEE Micro* Vol. 3, No. 4, pp. 67-83 (Part I), No. 5, pp. 16-26 (Part II), No. 6, pp. 24-39 (Part III), 1983.