A Comparison of Heterogeneous Multi-valued Decision Diagram Machines for Multiple-output Logic Functions

Hiroki Nakahara*, Tsutomu Sasao*, and Munehiro Matsuura*
*Kyushu Institute of Technology, Iizuka, Japan

***Abstract*—A heterogeneous multi-valued decision diagram (HMDD) may have nodes with different numbers of variables. By partitioning the input variables into optimal disjoint sets, the HMDDs evaluate the function faster than BDDs with the same amount of memory. In this paper, we compare multi-output HMDD machines. First, we introduce three types of HMDDs: plural single-output HMDDs, Multi-Terminal HMDD, and HMDD for ECFN. Next, we show three HMDD machines (HMDDMs). Then, we compare three HMDDMs with respect to the memory size, the execution time, and the area-time complexity. The comparison shows that, as for the area-time complexity, the HMDD for ECFN machine is the best.**

I. Introduction

Decision diagram machines (DDMs) are special purpose processors that evaluate logic functions [6], [1]. Applications for DDMs include industrial process controllers [18]; logic simulators [4]; and packet classifier [9]. The parallel branching program machine (PBM128) is 21.4-96.1 times faster than the Core2Duo, and the total (dynamic and static) power consumption is 23.6% of that for the Core2Duo [10].

The heterogeneous multi-valued decision diagram (HMDD) may have nodes with different number of variables [7]. By selecting an optimal partition of the input variables, the HMDDs can evaluate logic functions about two times faster than BDDs using the same amount of memory [11]. In the previous work, we only considered the HMDDs for single output function. In this paper, we consider HMDDs for multiple-output functions. For BDDs, various representations of multiple-output functions are considered: plural single-output BDDs; a multi-terminal BDD (MT-BDD) [15]; and BDD for encoded characteristic function for non-zero outputs (ECFN) [14]. In this paper, first, we extend these decision diagrams to HMDDs. Then, we develop HMDD machines for multiple-output functions. Next, we compare these machines with respect to the memory size, the execution time. In the embedded system, since the memory size and the execution time are limited, compact and fast systems are required. Thus, we compare these machines with respect to the area-time complexity.

The rest of the paper is organized as follows: Chapter 2 defines important words; Chapter 3 introduces HMDDs representing multiple-output functions; Chapter 4 develops the HMDD machines for multiple-output functions; Chapter 5 compares the HMDD machines; and Chapter 6 concludes the paper.

II. Preliminary

Definition 2.1: Let $f(X) : B^n \rightarrow B$ be a two-valued logic function, where $B = \{0, 1\}$. Let $X = (x_1, x_2, \ldots, x_n), x_i \in B$ be an ordered set of binary variables. Let $\{X\}$ denote the unordered set of variables in X. If $\{X\} = \{X_1\} \cup \{X_2\} \cup \cdots \cup \{X_u\}$ and $\{X_i\} \cap \{X_j\} = \phi (i \neq j)$, then $(X_1, X_2, \ldots, X_u)$ is a **partition of** X, where X_i is **a super variable**. When $k_i = |X_i| (i = 1, 2, \ldots, u)$, $k_1 + k_2 + \cdots + k_u = n$.

Definition 2.2: A **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))** [5].

Definition 2.3: 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 the following two reduction rules to an OBDD:

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

Definition 2.4: Let $X = (X_1, X_2, \ldots, X_u)$ be a partition of the input variables, and $k_i = |X_i|$ be the number of inputs for node i. When $k = |X_1| = |X_2| = \cdots = |X_u|$, an ROMDD is **a homogeneous MDD (MDD(k))**. On the other hand, if there exists a pair (i, j) such that $|X_i| \neq |X_j|$, then, it is **a heterogeneous MDD (HMDD)**.

If the evaluation time for all the DD nodes are the same, then the evaluation time for a DD is proportional to the **average path length (APL)** [3]. We assume that a DD machine evaluates each node in a fixed time. In this case, we can use APL to estimate the computation time.

Definition 2.5: Let $(X_1, X_2, \ldots, X_u)$ be a partition of the input variables X. Suppose that X_i can take any value c, where $c \in \{0, 1, \ldots, r-1\}, r = 2^{k_i}$. Then, $P(X_i = c)$ denotes the probability that X_i has the value c. **The Path Probability (PP)** of a path p_i, denoted by $PP(p_i)$, is the probability that the path p_i is selected in all assignments of values to the r-valued

0195-623X/11 $26.00 © 2011 IEEE
DOI 10.1109/ISMVL.2011.15

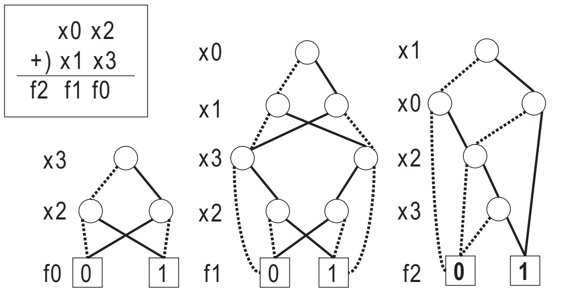

Fig. 1. An example of SO-BDDs for 2-bit adder.

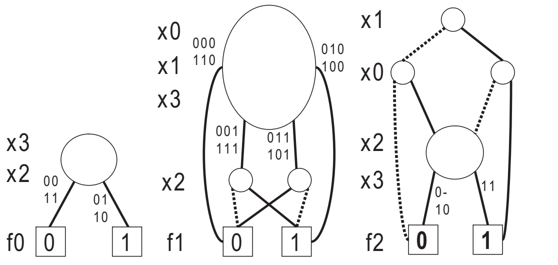

Fig. 2. An example of SO-HMDDs for 2-bit adder.

variables. Then, we have $PP(p_i) = \sum_{\vec{c} \in C_i} P(X_1 = c_1) \cdot P(X_2 = c_2) \cdot \ldots \cdot P(X_u = c_u)$, where C_i denotes a set of assignments of values to the variables X selecting the path p_i, and $\vec{c} = (c_1, c_2, \ldots, c_u)$. **The average path length (APL)** of a DD is $APL = \sum_{i=1}^{N} PP(p_i) \cdot l_i$, where N denotes the number of paths, and l_i denotes the path length of path p_i.

III. HMDDs for Multiple-Output Functions

The HMDD machine for single-output function has been shown [11]. However, many practical applications use multiple-output functions. In this paper, we consider the following multiple-output representations for decision diagrams (DDs).

· Plural single-output DDs [11]
· Multi-Terminal DD [15]
· DD for encoded characteristic function for non-zero outputs (ECFN) [14]

First, we introduce DDs for multiple-output functions. Then, we extend them to the HMDD. Note that, in this paper, m denotes the number of outputs.

A. Plural single-output HMDDs

The simplest method to represent a multi-output function is to use **plural Single-Output HMDDs (SO-HMDDs)**. For the SO-HMDDs, each HMDD can be optimized by using its own variable order, independently.

Example 3.1: Fig. 1 shows the plural single-output BDDs (SO-BDDs) for a 2-bit adder. Note that, these BDDs have different variable orders. Also, Fig. 2 shows the SO-HMDDs for the 2-bit adder. For SO-DDs, since the variable order for DDs can be different, the nodes cannot be shared among different DDs. ■

B. Multi-Terminal HMDD

A multi-terminal BDD (MTBDD) is the BDD that has m-bit terminal nodes. Also, we can define **a multi-terminal heterogeneous MDD (MT-HMDD)**. Since each terminal node stores m output values, the output values are obtained by just traversing the MT-HMDD from the root node to the terminal node. Since many outputs are evaluated by one traversal of the MT-HMDD, the evaluation is fast. However, the numbers of nodes tend to be too large to store in a memory for many practical applications [13].

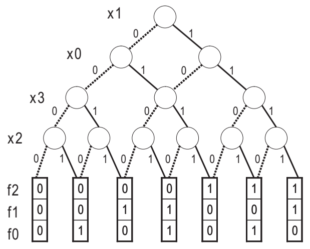

Fig. 3. An example of the MTBDD for 2-bit adder.

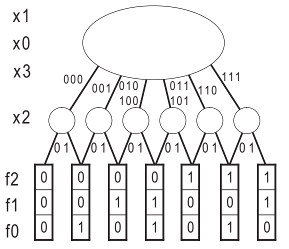

Fig. 4. An example of the MT-HMDD for 2-bit adder.

Example 3.2: Fig. 3 shows the MTBDD for the 2-bit adder, and Fig. 4 shows the MT-HMDD for the 2-bit adder. ■

C. HMDD for ECFN

A BDD for ECFN (Encoded Characteristic Function for Non-zero outputs) [14] requires smaller amount of memory than MT-HMDDs. A BDD for ECFN is considered as a generalization of a shared BDD (SBDD) [13]. BDDs for ECFNs are often smaller than corresponding SBDDs. This part shows the properties of the BDD for ECFN.

Definition 3.6: Let n be the number of the inputs, and m be the number of the outputs. An ECFN represents the mapping: $F : B^n \times B^u \rightarrow B$, where $u = \lceil log_2 m \rceil$. $F(\vec{a}, \vec{b}) = 1$ iff $f_{\nu(\vec{b})}(\vec{a}) = 1$, where $\nu(\vec{b})$ is an integer represented by the binary vector $\vec{b}$.

Definition 3.7: $x^0 = \bar{x}$, $x^1 = x$.

Definition 3.8: For an m-output function f_i (i=0, 1, ..., m-1), the ECFN is

$$F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i, \tag{1}$$

where $\vec{b} = (b_{u-1}, b_{u-2}, \ldots, b_0)$ is a binary representation of the integer i, $z_0, z_1, \ldots, z_{u-1}$ are the auxiliary variables that represent the outputs, and $u = \lceil log_2 m \rceil$.

Example 3.3: The four-output function (f_0, f_1, f_2, f_3) can be represented by the ECFN as follows: $F = \bar{z_1}\bar{z_0}f_0 \vee \bar{z_1}z_0 f_1 \vee z_1\bar{z_0}f_2 \vee z_1 z_0 f_3$. ■

Example 3.4: Fig. 5 shows the BDD for ECFN for the 2-bit adder, and Fig. 6 shows the HMDD for ECFN for the 2-bit adder. ■

The evaluation for the HMDD for ECFN is more complex than the SO-HMDDs and the MT-HMDD. The following algorithm shows the evaluation of the HMDD for ECFN.

Algorithm 3.1: (Evaluation of HMDD for ECFN)

```
1. Reset the auxiliary variables to zeros.
2. Evaluate the HMDD for ECFN corresponding to primary
   inputs and auxiliary variables.
3. Increment the value represented by the auxiliary vari-
   ables.
4. If all outputs are evaluated, then Terminate. Otherwise,
   go to Step 2.
```

AUX var. z1 z0	Function
0 0	f0
0 1	f1
1 -	f2

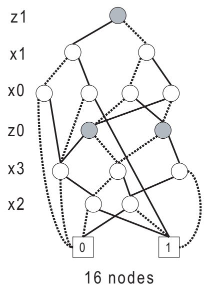

Fig. 5. An example of the BDD for ECFN for 2-bit adder.

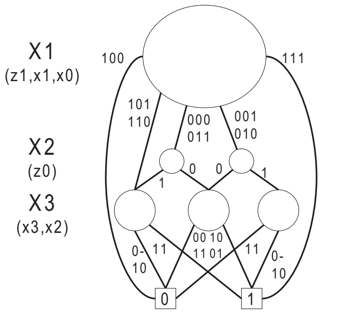

Fig. 6. An example of the HMDD for ECFN for 2-bit adder.

AUX var. z1 z0	Function
0 0	f2
0 1	f1
1 -	f0

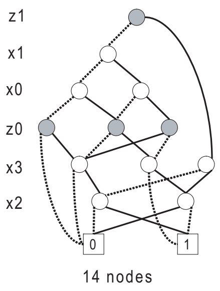

Fig. 7. Optimization of the encoding.

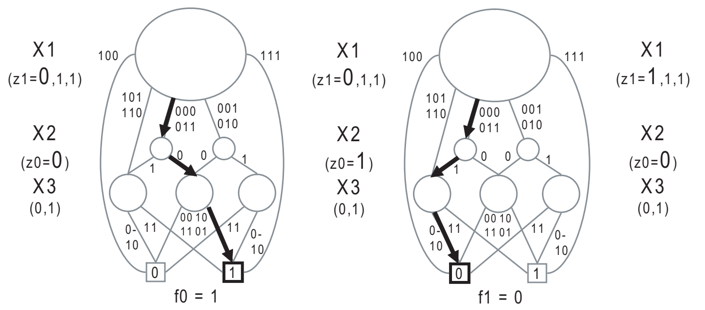

Fig. 8. An example of evaluation of the HMDD for ECFN for 2-bit adder.

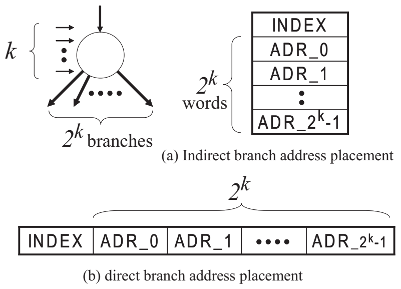

Fig. 9. Two types of branch address placements.

Example 3.5: Let the primary input be $(x_0, x_1, x_2, x_3) = (1, 1, 0, 1)$. Fig. 8 illustrates the evaluation of the HMDD for ECFN shown in Fig. 6. First, we set $(z_1, z_0) = (0, 0)$, and obtain $f_0 = 1$. Then, we increment the value to $(z_1, z_0) = (0, 1)$, and obtain $f_1 = 0$. Finally, we increment the value to $(z_1, z_0) = (1, 0)$, and obtain $f_2 = 1$. ■

From above example, to evaluate the ECFN by the hardware, the register that retains the auxiliary variables, the counter, and its controller are necessary.

D. Output Encoding of ECFN

In Definition 3.8, the integer i is represented by a binary vector $\vec{b}$ using the natural binary encoding. However, different encodings can simplify the HMDD for ECFN. To find an optimal encoding of the output is not easy. So, to construct HMDDs for ECFNs, a heuristic method that finds a suboptimal encoding is used [16].

Example 3.6: Fig. 7 shows the optimal encoding of the BDD for ECFN, that is different from the natural encoding shown in Fig. 5. In these cases, the variable orders are the same. This example shows that the output encoding influences the sizes of the ECFNs. ■

IV. Multi-Output HMDD Machines

A. Direct and Indirect Branch Address Placement [11]

In homogeneous DDs (e.g. BDD, MDD (k)), the numbers of branches are the same for all nodes. Thus, the lengths of fields for the branch instructions are also the same. These machines can directly get the branch address by reading input variables and the branch instruction. Since the HMDD can accept different numbers of input variables for nodes, the numbers of branch addresses can be different. Two types of branch address placements exits: one is **a direct branch address placement**; and the other is **a indirect branch address placement** [11]. In the direct branch address placement, the index and branch addresses are located to the same word. Although the field lengths are different for different k, the direct branch address placement can directly get the branch address. On the other hand, in the indirect branch address placement, the index and branch addresses are stored in the separated words. To evaluate a node, first, the current index is read. Then, the jump address corresponding to the value of the current input variables is read. Although the machine using indirect branch address placement is slower than the machine using direct branch placement, it uses the memory efficiently, since the words have the same length.

Example 4.7: Fig. 9 compares the indirect branch address placement with the direct branch address placement for k-input HMDD node. Although, the indirect branch address placement requires $2^k + 1$ words, the length for the words are the same. ■

The indirect jump can use the memory efficiently for heterogeneous DDs. In this paper, to evaluate a node for the HMDD, we use the indirect branch.

B. SO-HMDDs Machine (SO-HMDDsM)

In the SO-HMDDs, the non-terminal node is evaluated by **an indirect branch instruction** shown in Fig. 10, while the terminal node is evaluated by **a single-output and jump instruction** shown in Fig. 11. Fig. 12 shows **a SO-HMDDs Machine (SO-HMDDsM)**. In Fig. 12, **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 output counter (OC)** in the controller retains the assigned

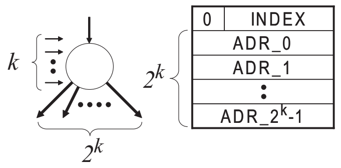

Fig. 10. An indirect branch instruction.

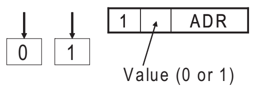

Fig. 11. Single-output and jump instruction.

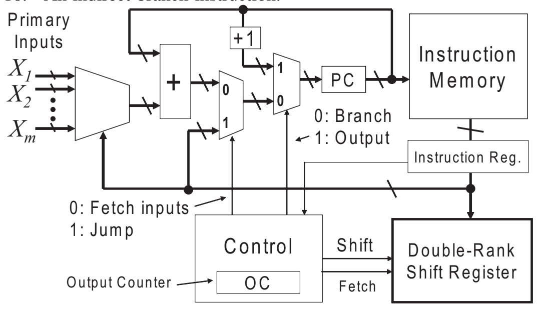

Fig. 12. SO-HMDDsM.

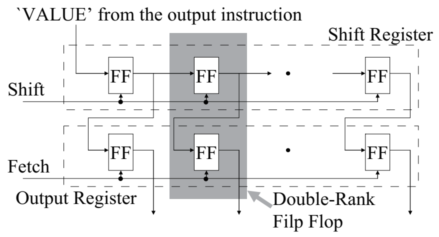

Fig. 13. Double-rank shift register.

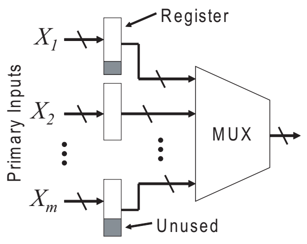

Fig. 14. Input selector.

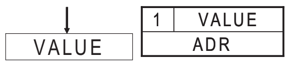

Fig. 15. Multi-output and jump instruction.

number for each HMDD; **the double-rank shift register** retains the output value; and **the input selector** shown in Fig. 14 selects an arbitrary size of super variables.

Fig. 13 shows the double-rank shift register consisting of the shift register and **the output register**. In the double-rank shift register, each flip-flop consists of **a double-rank flip-flop** [12]. The shift register retains outputs of the SO-HMDDs. When all outputs are evaluated, the value of the shift register is sent to the output register.

The following examples show the execution of the branch instruction and the output instruction for the SO-HMDDsM.

Algorithm 4.2: (2^k indirect branch instruction for the SO-HMDDsM)

```
1. Read indirect branch address
   1.1 Read the index corresponding to index filed in the branch instruction.
   1.2 To obtain the indirect branch address, 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.
```

Algorithm 4.3: (Single-output and jump instruction for the SO-HMDDsM) Let OC be the value of the output counter, and m be the number of outputs.

```
1. After reset of the machine, $OC \leftarrow 0$.
2. Output the value
   2.1 Read the output value and the jump address corresponding to the PC.
   2.2 Set the output value to the shift register in the double-rank shift register.
   2.3 $OC \leftarrow OC + 1$.
   2.4 If all outputs are evaluated ($OC = m$), then the values of the shift register are sent to the output register, and $OC \leftarrow 0$.
3. Perform the jump operation, similarly to the Step 2 of Algorithm 4.2.
```

Let n be the number of the primary inputs, m be the number of the primary outputs, $p(q)$ be the number of the non-terminal nodes for the SO-HMDD that represents the function f_q $(1 \leq q \leq m)$, $s_i(q)$ be the size of index for the node i. Then, the number of addresses of the HMDD for f_q is

$$2 + \sum_{i=1}^{p(q)} (2^{s_i(q)} + 1),$$

where the first term denotes the number of addresses for two terminal nodes, and the second term denotes the number of addresses for non-terminal nodes. Thus, the number of addresses A_{SO} of the SO-HMDDsM for m output functions is

$$A_{SO} \quad = \quad \sum_{q=1}^{m}(2 + \sum_{i=1}^{p(q)}(2^{s_i(q)} + 1)).$$

Therefore, the word length for the SO-HMDDsM W_{SO} is

$$W_{SO} \quad = \quad max(\lceil log_2 A_{SO} \rceil + 2, \lceil log_2 n \rceil + 1).$$

From above expressions, the memory size for the SO-HMDDsM is

$$A_{SO} W_{SO}. \tag{2}$$

C. MT-HMDD Machine (MT-HMDDM)

In an MT-HMDD, a non-terminal node is evaluated by the indirect branch instruction, similar to the case of the SO-HMDDs. However, the terminal node is evaluated by **a multi-output and jump instruction** shown in Fig. 15. Fig. 16 shows **an MT-HMDD machine (MT-HMDDM)**. In Fig. 16, the instruction memory, the instruction register, and the input selector are the same as that of the SO-HMDDsM. For the MT-HMDD, since m outputs are evaluated at a time, only the output register is used. Also, the output counter is not necessary. Thus, the controller for the MT-HMDDM is simpler than that for the SO-HMDDsM.

The indirect branch instruction is executed in the similar way to the SO-HMDDsM shown in Algorithm 4.2. The following example shows the execution of the multi-output and jump instruction.

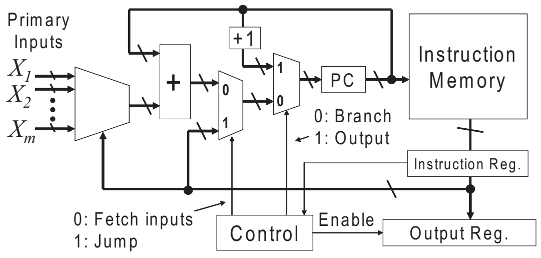

Fig. 16. MT-HMDD machine (MT-HMDDM).

Algorithm 4.4: (Multi-output and jump instruction for the MT-HMDDM)

1. Output the value.
   1.1 Read the output value corresponding to the PC.
   1.2 Set the output value to the output register. Also, increment the PC.
2. Perform the jump operation, similarly to the Step 2 of Algorithm 4.2.

Let p be the number of nodes for the MT-HMDD, and s_i [1] be the size of node i. The number of addresses A_{MT} for the MT-HMDDM is

$$A_{MT} = \sum_{i=1}^{p}(2^{s_i}+1).$$

Let n be the number of primary inputs, and m be the number of primary outputs. The word length W_{MT} for the MT-HMDDM is

$$W_{MT} = max(\lceil log_2 A_{MT}\rceil, \lceil log_2 n\rceil + 1, m+1).$$

Therefore, the memory size for the MT-HMDDM is

$$A_{MT}W_{MT}. \quad (3)$$

D. HMDD for ECFN Machine (HMDDM for ECFN)

In the HMDD for ECFN, the non-terminal node is evaluated by the indirect branch instruction shown in Fig. 10. On the other hand, the terminal node is evaluated by the single-output and jump instruction shown in Fig. 11. In the **HMDD for ECFN machine (HMDDM for ECFN)** shown in Fig. 17, the instruction memory, the instruction register, and the double-rank shift register are the same as Fig. 13. To evaluate the HMDD for ECFN, we use **the auxiliary variable counter (AC)** that retains the value of the auxiliary variable. The input selector for the HMDDM for ECFN selects both the primary inputs and the auxiliary variables from the AC.

The indirect branch instruction is executed in a similar way to the SO-HMDDsM. The following example shows the execution of the single-output and jump instruction for the HMDDM for ECFN.

Algorithm 4.5: (Single-output and jump instruction for the HMDDM for ECFN). Let AC be the value of the auxiliary counter, and m be the number of outputs.

1. After reset of the machine, $AC \leftarrow 0$.
2. Output the value.
   2.1 Read the value and the jump address corresponding to the PC.

[1]For the terminal node, $s_i = 0$

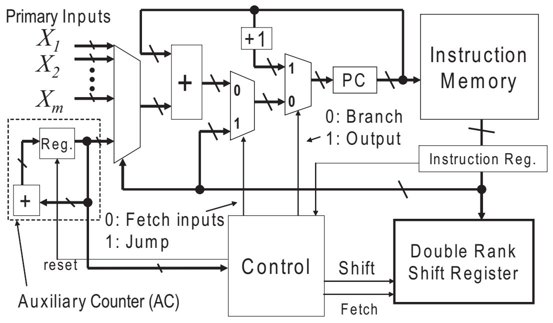

Fig. 17. HMDD for ECFN machine (HMDDM for ECFN).

TABLE I
COMPARISON OF THREE HMDDMS.

HMDD Machine	LEs		Max. Freq.[MHz]
	Comb.	Reg.	
SO-HMDDsM	354	89	92.11
MT-HMDDM	348	57	93.10
HMDDM for ECFN	399	95	90.31

   2.2 Set the value to the double-rank shift register, and $AC \leftarrow AC + 1$.
   2.3 If all outputs are evaluated ($AC = m$), then send the values of the shift register to the output register, and $AC \leftarrow 0$.
3. Perform the jump operation, similarly to the Step 2 of Algorithm 4.2.

Let n be the number of primary inputs, m be the number of outputs, p be the number of non-terminal nodes for the HMDDM for ECFN, s_i be the size of node i. Similarly to the case of SO-HMDDsM, the number of addresses A_{ECFN} for the HMDDM for ECFN is obtained as

$$A_{ECFN} = 2 + \sum_{i=1}^{p}(2^{s_i}+1).$$

Since the ECFN has $n + \lceil log_2 m\rceil$ bits inputs, the word length W_{ECFN} is

$$W_{ECFN} = max(\lceil log_2 A_{ECFN}\rceil + 2, \lceil log_2(n + \lceil log_2 m\rceil)\rceil + 1).$$

Therefore, the memory size for the HMDDM for ECFN is

$$A_{ECFN}W_{ECFN}. \quad (4)$$

V. COMPARISON OF MULTI-OUTPUT HMDDMS

A. Implementation of HMDDMs

We implemented three types of HMDDMs on the Altera's Cyclone III FPGA (EP3C25, 24,624 LEs, 66 M9Ks). Table I compares three types of HMDDMs. From Table I, the number of LEs for three machines are slightly different. However, their LE usages are quite small compared with the available LEs in the FPGA. Also, the clock frequencies for their machines are slightly different.

B. Comparison of HMDDMs

We generated SO-HMDDs, MTHMDDs, and HMDDs for ECFN for selected MCNC benchmark functions [17]. Then, we compared their memory size (area), execution time (time), and the area-time complexity. To generate HMDDs, we

TABLE II
COMPARISON OF MEMORY SIZE ([KB]).

Name	I/O	SO	MT	ECFN
alu4	14/8	2.7	16.4	**2.2**
apex2	39/3	2.7	2.7	**2.6**
cc	21/26	0.9	3,370.7	**0.6**
lal	26/19	1.4	912.5	**1.3**
pcler8	27/17	1.3	38.3	**1.0**
spla	24/21	7.6	230.7	**3.7**
ttt2	16/46	2.0	22,140.7	**1.2**
ts10	22/16	2.5	6.459.7	**1.2**
C1355	41/32	14,936.5	—	**406.3**
C1908	33/25	5,339.6	—	**101.2**
C3540	50/22	14,978.4	—	**844.4**

TABLE III
COMPARISON OF EXECUTION TIME ([NSEC]).

Name	I/O	SO	MT	ECFN
alu4	14/8	565	**81**	876
apex2	39/3	592	**286**	731
cc	21/26	1,035	**172**	2,731
lal	26/19	1,494	**258**	2,921
pcler8	27/17	1,031	**130**	1,929
spla	24/21	2,685	**92**	3,578
ttt2	16/46	1,676	**215**	3,618
ts10	22/16	944	**86**	1,150
C1355	41/32	**9,095**	—	10,608
C1908	33/25	**3,238**	—	5,811
C3540	50/22	**2,731**	—	5,362

TABLE IV
COMPARISON OF AREA-TIME COMPLEXITY.

Name	I/O	SO	MT	ECFN
alu4	14/8	1,544	**1,333**	1,998
apex2	39/3	1,637	**775**	1,912
cc	21/26	**1,015**	579,287	1,432
lal	26/19	**2,166**	235,251	3,860
pcler8	27/17	**1,423**	4,973	1,973
spla	24/21	20,602	21,114	**13,292**
ttt2	16/46	**3,514**	4,756,318	4,586
ts10	22/16	2,422	555,081	**1,430**
C1355	41/32	135,850,851	—	**4,310,247**
C1908	33/25	17,291,736	—	**587,491**
C3540	50/22	40,907,235	—	**4,528,292**

used the CPU (Intel's Core2Duo U7600@1.2GHz); DDR2-SODIMM 3GBytes memory; and Windows XP SP2. For each function, we built the DD that minimizes APL with the memory size limitation [8]. To construct the HMDDM, the memory size limitation is set to that of the BDD. To obtain the memory size, we used Exprs. (2), (3), and (4). Note that, we could not generate MT-HMDDs for several functions (C1355, C1908, C3540), since the memory size exceeded 3 GBytes. The implemented HMDDM performs one instruction per four clocks, thus, the execution time is $\frac{APL}{F} \times 4$, where F is the maximum clock frequency shown in Table I, and APL is generated from HMDDs. Table II compares the memory sizes, Table III compares the execution time, and Table IV compares the area-time complexity.

C. Discussions

a) Memory Size: Table II shows that, for all functions, the HMDD for ECFN requires the minimum memory. Although the SO-HMDDs can reduce the number of nodes by changing variable order for each BDD, for large functions (C1355, C1908, C3540), the reduction technique does not work. For some MT-HMDDs, the numbers of nodes are too large to implement. So, the MT-HMDDs is limited to small applications. On the other hand, since the HMDD for ECFN reduces the number of nodes by sharing the subgraph, all functions can be represented compactly. Thus, the HMDD for ECFN is the best for the memory size.

b) Execution Time: Table III shows that, for small functions, the MT-HMDDM is faster. Since many outputs are evaluated by one traversal of the MT-HMDD, the evaluation is fast. However, as shown in Table II, the number of nodes are too large to store in a memory. For large functions, the SO-HMDDsM is faster. An SO-HMDDsM evaluates the input variables m times, while an HMDDM for ECFN evaluates both the input variables and the auxiliary variables m times. Thus, the SO-HMDDsM is faster than the HMDDM for ECFN. Therefore, for practical applications, the SO-HMDDsM is the best for the execution time.

c) Area-Time Complexity: Table IV shows that, for small functions (alu4, apex2) with small MT-HMDDs, the MT-HMDDM is the best. On the other hand, for large functions (C1355, C1908, C3540), since the HMDD for ECFN is relatively fast to evaluate the function and is compact, the HMDDM for ECFN is the best.

From these discussion, we can conclude that, for practical applications, the HMDDM for ECFN is fast and compact.

VI. CONCLUSION

In this paper, we showed three types of HMDDMs: SO-HMDDsM, MT-HMDDM, and HMDDM for ECFN. We compared three machines with respect to the memory size, the execution time, and the area-time complexity. The comparison shows that, as for the area-time complexity, the HMDDM for ECFN is the best.

VII. ACKNOWLEDGMENTS

This research is supported in part by the Grants in Aid for Scientific Research of JSPS, and the grant of Innovative Cluster Project of MEXT (the second stage). Discussion with Dr. Shinobu Nagayama was 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. Comput.*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

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

[4] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno, "A hardware simulation engine based on decision diagrams," *ASPDAC2000*, Jan., 26-28, Yokohama, Japan, pp.73-76.

[5] T. Kam, T. Villa, R. K. Brayton, and A. L. Sagiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and Applications," *Multiple-Valued Logic: An International Journal,* Vol. 4, No. 1-2, pp. 9-62, 1998.

[6] 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.

[7] S. Nagayama and T. Sasao, "Compact representations of logic functions using heterogeneous MDDs," *ISMVL2003*, May, 2003, pp.247-255.

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

[9] H. Nakahara, T. Sasao, and M. Matsuura, "Packet classifier using a parallel branching program machine," *DSD2010*, Lille, France, Sept., 2010, pp.745-752.

[10] H. Nakahara, T. Sasao, M. Matsuura, and Y. Kawamura, "A parallel branching program machine for sequential circuits: Implementation and evaluation," *IEICE Transactions on Information and Systems*, Vol. E93-D, No.8, Aug., 2010, pp.2048-2058.

[11] 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.

[12] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, "Realization of sequential circuits by look-up table ring," *MWSCAS2004*, Hiroshima, July 25-28, 2004, pp.I:517-I:520.

[13] T. Sasao, Y. Iguchi and M. Matsuura, "Comparison of decision diagrams for multiple-output logic functions," *IWLS2002*, New Orleans, Louisiana, June 4-7, 2002, pp.379-384.

[14] T. Sasao, M. Matsuura, Y. Iguchi, and S. Nagayama, "Compact BDD representations for multiple-output functions and their applications to embedded system," *IFIP VLSI-SOC'01*, Montpellier, France, December 3-5, 2001, pp. 406-411.

[15] T. Sasao, M. Matsuura, and Y. Iguchi, "Cascade realization of multiple-output function and its application to reconfigurable hardware," *IWLS2001*, pp. 225-230, Lake Tahoe, June, 2001.

[16] T. Sasao, "Compact SOP representations for multiple-output functions: An encoding method using multiple-valued logic," *ISMVL2001*, Warsaw, Poland, May 22-24, 2001, pp.207-212.

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

[18] 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.