# On a Prefetching Heterogeneous MDD Machine

Hiroki Nakahara*, Tsutomu Sasao*, and Munehiro Matsuura*
*Kyushu Institute of Technology, Kawazu 680-4, Fukuoka, Japan
Email: nakahara,sasao,matsuura@cse.kyutech.ac.jp

*Abstract*— **This paper considers a heterogeneous multi-valued decision diagram machine (HMDDM). First, we introduce a standard heterogeneous multi-valued decision diagram machine (standard HMDDM). Then, we show a method to prefetch index for the HMDDM (prefetching HMDDM). The standard HMDDM requires two memory references to read the jump address and the index separately, while the prefetching HMDDM requires only one memory reference to them at a time. Thus, the prefetching HMDDM is twice faster than the standard HMDDM. We implemented the prefetching HMDDM on an FPGA. Also, we compared with Intel's Core2Duo (1.2 GHz). As for the execution time, the prefetching HMDDM is 14.53-18.97 times faster than the Core2Duo. Since the HMDDM consists of the small FPGA and the off-chip RAM, the power consumption for the HMDDM is smaller than that for the conventional CPU. Thus, the HMDDM is the power-performance efficient processor.**

## I. Introduction

Various decision diagrams (DDs), e.g., BDD[3], MDD[10], QRBDD[19], QRMDD[6], heterogeneous MDD (HMDD)[12], exist. DD machines (DDMs) are special purpose processors that evaluate these DDs [2].

Various DDMs have been proposed [7], [2], [20], [11], [16]. Applications of DDMs include industrial process controllers [22], logic simulators [7], and packet classifiers [15].

A DD machine (DDM) consists of the control part and the memory storing the data for the nodes. Two types implementations exist: One using on-chip memory on a high-performance-and-expensive device (e.g., the ASIC or the FPGA) [15], [16]; and the other using off-chip off-the-shelf memory with a low-performance-and-inexpensive device (e.g., the CPLD) [7]. For the first type of the implementation, the cost per a bit is high[1]. On the other hand, consider the second type of the implementation, where off-the-shelf memory is used. The heterogeneous MDD (HMDD) machine can utilize most of the given memory space by selecting the optimal partition of the input variables [13]. Since the price of off-the-shelf memory per a bit is much lower than the on-chip memory in the FPGA, the evaluation speed may be increased by using larger off-chip off-the-shelf memory. We proposed the standard HMDD machines that have a good compatibility to the off-the-shelf memory [17].

In this paper, we propose the prefetching HMDD machine, an improved version of the HMDD machine. Then, we compare the proposed prefetching HMDD machine with the Intel's Core2Duo.

The rest of the paper is organized as follows: Chapter 2 defines important words; Chapter 3 shows the standard HMDDM; Chapter 4 introduces the prefetching HMDDM; Chapter 5 shows experimental results; and Chapter 6 concludes the paper.

[1]As of September 2010, the price of 1 Mega Bytes off-the-shelf SRAM is lower than \$10. On the other hand, the price of the high-end FPGA that contains several Mega Bytes on-chip memories is about \$10,000 [5].

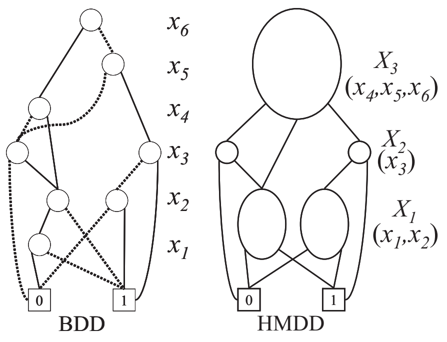


Fig. 1. Various Decision Diagrams (DDs).

## 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 $f$. Each non-terminal node labeled with a variable $x_i$ has two outgoing edges which point nodes representing cofactors of $f$ with respect to $x_i$. When the Shannon expansions are performed with respect to $k$ variables, each non-terminal node has $2^k$ edges. In this case, we have a **Multi-valued Decision Diagram (MDD($k$))**.

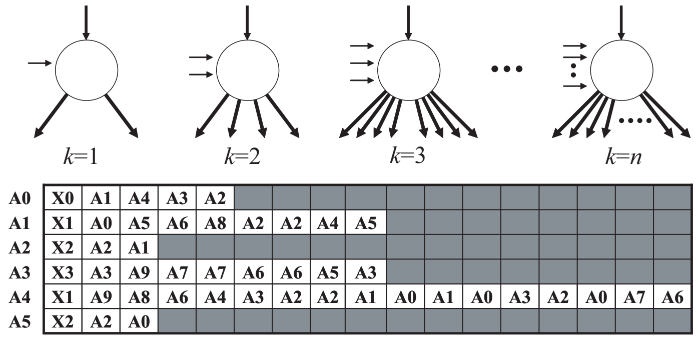
*Definition 2.3:* In a DD, a sequence of edges and non-terminal nodes leading from the root node to a terminal node is a **path**. **An ordered BDD (OBDD)** has the same variable order on any path. **A reduced ordered BDD (ROBDD)** is derived by applying the following two reduction rules to an OBDD:

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) means 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)**.

*Example 2.1:* Fig. 1 illustrates the BDD and the HMDD for the MCNC benchmark function C17 [21].



| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A0 | X0 | A1 | A4 | A3 | A2 | | | | | | | | | | | | |
| A1 | X1 | A0 | A5 | A6 | A8 | A2 | A2 | A4 | A5 | | | | | | | | |
| A2 | X2 | A2 | A1 | | | | | | | | | | | | | | |
| A3 | X3 | A3 | A9 | A7 | A7 | A6 | A6 | A5 | A3 | | | | | | | | |
| A4 | X1 | A9 | A8 | A6 | A4 | A3 | A2 | A2 | A1 | A0 | A1 | A0 | A3 | A2 | A0 | A7 | A6 |
| A5 | X2 | A2 | A0 | | | | | | | | | | | | | | |

Fig. 2. Direct branch instructions for the HMDD.

Suppose that 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)** [4]. We assume that a DD machine evaluates each node in the same 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 $c \in \{0, 1, \ldots, r-1\}$. Then, $P(X_i = c)$ denotes the probability that $X_i$ has 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 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. Standard Heterogeneous MDD Machine

### A. Direct Branch and Indirect Branch

To evaluate a node for the HMDD, the jump operation is performed as follows:

1. Reads index $X_i$ for the node.
2. Reads the branch address corresponding $X_i$, and set it to the PC.

Two types of the allocations for the branch address exist. A **direct branch** allocates both the index and branch addresses to the same word, while an **indirect branch** allocates the index and branch addresses to the different word. Since the direct branch reads them at a time, it can perform the jump operation faster than the indirect branch. However, for the HMDD, the direct branch requires larger memory than the indirect branch.

*Example 3.2:* Fig. 2 shows direct branch instructions for an HMDDM that evaluates non-terminal nodes. When the number of input variables for a node is $k$, the number of branches is $2^k$. Thus, words for the branch instruction of nodes may have different numbers of branch addresses. ■

To use the memory efficiently in the HMDDM, we use the standard indirect branch that reads the index and the jump address separately. The machine first reads the current index, and then reads the jump address corresponding to the values of the index and the input variables. Although the standard indirect branch is slower than the direct branch, it efficiently uses the memory, since the words have no redundant part. The next example explains this.

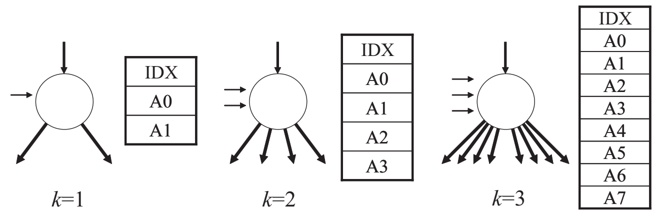


Fig. 3. Standard indirect branch instructions for the HMDDM.

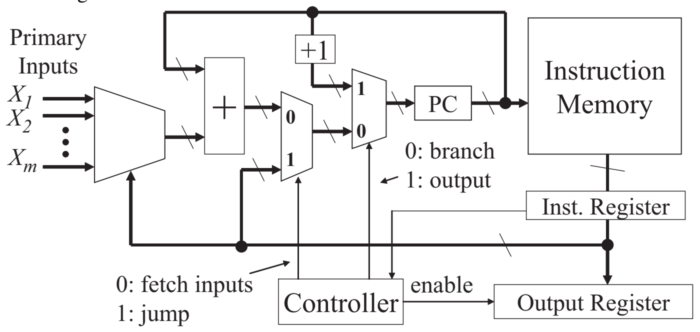


Fig. 4. Standard HMDDM.

*Example 3.3:* Fig. 3 shows the standard indirect branch instructions for the HMDDM that evaluates non-terminal nodes. In Fig. 3, the *IDX* stores the index for the input variable. ■

In the HMDDM, even if the sizes of super variables are different, the instructions for standard indirect branch has no redundant part. So, the standard indirect jump can use the memory efficiently to evaluate heterogeneous DDs.

### B. Standard Heterogeneous MDD Machine (HMDDM)

Fig. 4 shows a standard HMDD Machine (HMDDM), and Fig. 5 shows its instructions. The standard HMDDM consists of the instruction memory, the instruction register, the output register, and the PC. It uses standard indirect $2^k$-branch instructions and output instructions. The standard indirect $2^k$-branch instruction is operated in two modes: **the fetch mode** and **the jump mode**. In the fetch mode, index is read, and input variables are selected. In the jump mode, the jump addresses are read and the branch operations are performed. On the other hand, the output instruction is operated in two modes: **the output mode** and the jump mode. To change the modes, the controller generates control signals, and two multiplexors select the mode. An adder is used to compute the address in the jump mode, and an increment circuit is used in the output mode.

Since the size of the super variables can be different, input registers with $\max_i\{k_i\}$ bits shown in Fig. 6 is used.

*Algorithm 3.1:* (Standard indirect $2^k$-branch instruction)

Step 1. Fetch mode.

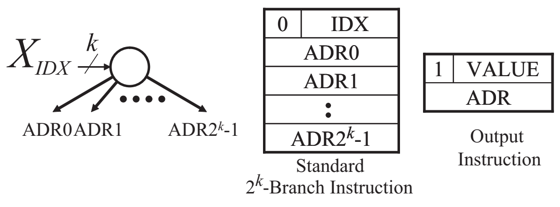


Fig. 5. Instructions for the standard HMDDM.

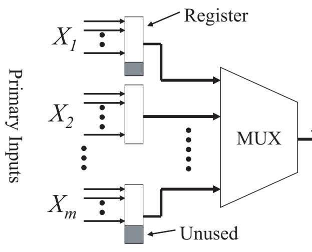


Fig. 6. Input selector for the HMDDM.

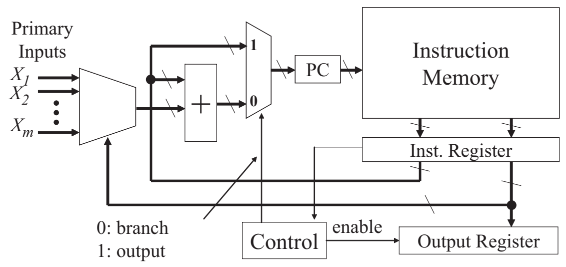
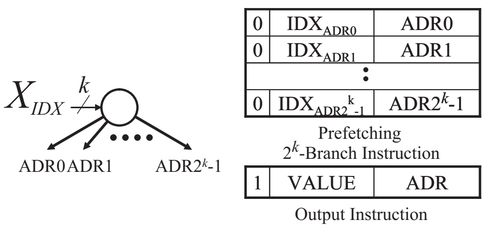


Fig. 7. Prefetching HMDDM.



| | | |
|---|---|---|
| 0 | IDX$_{ADR0}$ | ADR0 |
| 0 | IDX$_{ADR1}$ | ADR1 |
| | ⋮ | |
| 0 | IDX$_{ADR2^k-1}$ | ADR$2^k$-1 |

Prefetching $2^k$-Branch Instruction

| | | |
|---|---|---|
| 1 | VALUE | ADR |

Output Instruction

Fig. 8. Instructions for the Prefetching HMDDM.

1) 1.1 Read the instruction memory specified by the PC.
1.2 To add the value of the input variables and the content of the PC, the controller generates signals. Then, the indirect address is sent to the PC.

Step 2. Jump mode.

2) 2.1 Read the jump address specified by the PC.
2.2 To perform the jump, the controller generates signals. Then, the jump address is sent to the PC.

*Algorithm 3.2:* (Output instruction)

Step 1. Output mode.

1) 1.1 Read the instruction memory specified by the PC.
1.2 The controller generates signals, and the output data is sent to the output register. Concurrently, it increments the PC.

Step 2. Jump mode.

2.1 Perform the jump mode shown in Algorithm 3.1 Step 2.

Let $M_{STAND\_HMDDM}$ be the total memory size of the standard HMDDM, $k_i$ be the number of inputs for a node $i$, $(X_1, X_2, \ldots, X_u)$ be a partition of the inputs $X$, and $N_{HMDD}$ be the number of the nodes in the HMDD. Since in the standard HMDD, each node has an index and $2^{k_i}$ branch edges, each non-terminal node requires $2^{k_i}+1$ words. For a terminal node, we assume that $k_i$ is zero. Thus, two words (an index and the output value) are necessary. The word length for the index $w_{IDX}$ is

$$w_{IDX} \quad = \quad 1 + \lceil log_2 u \rceil, \tag{1}$$

where the first term corresponds to the opecode; and the second term corresponds to the index. The total number of words for the standard HMDDM is $a = \sum_{i=1}^{N_{HMDD}} (2^{k_i}+1)$. Thus, the word length for the jump address $w_{JMP}$ is

$$w_{JMP} \quad = \quad \lceil log_2 a \rceil. \tag{2}$$

In most cases, the number of nodes for the DDs $H_{HMDD}$ is larger than the number of super variables $u$. Thus, we have $w_{JMP} > w_{IDX}$ [2]. Then, we have the relation:

$$M_{STAND\_HMDDM} \quad = \quad a \cdot \lceil log_2 a \rceil. \tag{3}$$

## IV. Prefetching Heterogeneous MDD Machine

In the standard HMDDM shown in Fig. 4, two memory references are required to evaluate a node. In this part, we propose **a prefetching heterogeneous MDD machine (prefetching HMDDM)** that requires only one memory reference to read the jump address and the index. Thus, the prefetching HMDDM is faster than the standard HMDDM. The disadvantages for the prefetching HMDDM are longer instruction words and increase of the memory.

Fig. 7 shows the prefetching HMDDM, and Fig. 8 shows its instructions. The prefetching HMDDM consists of the instruction memory, the instruction register, the output register, and the PC. It uses prefetching $2^k$-branch instructions and output instructions. To execute the prefetching $2^k$-branch instruction, it uses **the branch mode**. Also, to execute the output instruction, it uses **the output mode**.

*Algorithm 4.3:* (Prefetching $2^k$-branch instruction)

1. Branch mode.
1.1 Read the instruction memory specified by the PC.
1.2 Add the input variables specified by the index to the PC, then, the jump address is sent to the PC. Also, the next index is read from the instruction.

*Algorithm 4.4:* (Output instruction)

1. Output mode.
1.1 Read the instruction memory specified by the PC.
1.2 The output value is sent to the output register. Also, the jump address is sent to the PC.

Let $M_{PREF\_HMDDM}$ be the total memory size for the prefetching HMDDM, $k_i$ be the number of inputs for a node $i$, $(X_1, X_2, \ldots, X_u)$ be a partition of the inputs $X$, and $N_{HMDD}$ be the number of nodes. Since in the prefetching HMDD, each non-terminal node has $2^{k_i}$ branch edges, the necessary number of addresses for each node is also $2^{k_i}$. So, the total number of instructions for the prefetching HMDDM is $b = \sum_{i=1}^{N_{HMDD}} (2^{k_i})$. Also, in this case, each terminal node has an output and $2^0 = 1$ branch address. Thus, $k_i = 0$. Therefore, we have the relation:

$$M_{PREF\_HMDDM} \quad = \quad b \cdot (1 + \lceil log_2 u \rceil + \lceil log_2 b \rceil), \tag{4}$$

where the first term corresponds to the opecode; the second term corresponds to the index; and the last term corresponds to the jump address.

## V. Experimental Results

### A. Implementation of Heterogeneous DD Machines

We implemented the standard and prefetching HMDDM on the Altera Cyclone III starter kit (FPGA: Cyclone III, EP3C25). Then, we obtained the necessary number of logic elements (LEs) and the maximum clock frequency. For the FPGA synthesis tool, we used QuartusII (v.9.1). From the implementation, as for the standard HMDDM, it consumes 348 LEs and the maximum clock frequency is 93.1 MHz, while as for the prefetching HMDDM, it consumes 239 LEs

[2]When $w_{JMP}$ is less than the number of outputs, we use multiple output instructions whose word length is less than $w_{JMP}$

TABLE I

COMPARISON OF VARIOUS DDMS.

| Name | In | Out | FF | C2D@1.2GHz | | | HMDDM+1MB SRAM | | | HMDDM+2MB SRAM | | | HMDDM+4MB SRAM | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | APL | MEM [KB] | TIME [ns] | APL | MEM [KB] | TIME [ns] | APL | MEM [KB] | TIME [ns] | APL | MEM [KB] | TIME |
| s5378 | 35 | 49 | 164 | 150.9 | 2048 | 12030 | 176.1 | 943 | 1761 | 150.9 | 2048 | 1509 | 100.5 | 3979 | 1005 |
| s9234 | 36 | 39 | 211 | 283.1 | 2048 | 13450 | 352.8 | 928 | 3528 | 283.1 | 2048 | 2831 | 256.7 | 3474 | 2567 |
| dsip | 229 | 197 | 224 | 109.0 | 1981 | 17500 | 204.1 | 942 | 2041 | 109.0 | 1981 | 1090 | 83.0 | 3950 | 830 |
| bigkey | 263 | 197 | 224 | 125.3 | 1979 | 19170 | 156.1 | 918 | 1561 | 125.3 | 1979 | 1253 | 110.3 | 4089 | 1103 |
| apex6 | 135 | 99 | | 24.9 | 2035 | 3700 | 28.1 | 1022 | 281 | 24.9 | 2035 | 249 | 24.8 | 4089 | 248 |
| cps | 24 | 102 | | 8.8 | 1990 | 3468 | 8.8 | 999 | 88 | 8.8 | 1990 | 88 | 8.3 | 3797 | 83 |
| frg2 | 143 | 139 | | 33.7 | 1919 | 6390 | 36.1 | 1002 | 361 | 33.7 | 1919 | 337 | 31.2 | 4093 | 312 |
| Ratio | | | | | | 1.00 | | | 14.53 | | | 16.75 | | | 18.97 |

and the maximum clock frequency is 110.1 MHz. Therefore, the prefetching HMDDM is faster and smaller than standard HMDDM.

## B. Comparison of Various DDMs

We constructed two DDs from selected MCNC benchmark functions [21], and realized on the Intel's Core2Duo U7600 (1.2 GHz, cache L1 data 32 KBytes, L1 instruction 32 KBytes, and L2 2 MBytes) and the prefetching HMDDM. Then, we obtained the memory sizes and execution times for each DDM. As for a multi-output function, we partitioned into single output functions. We used the variable order that minimizes the memory size of the BDD [18].

*a) Core2Duo:* We built the HMDD that minimizes APL with a given memory size limitation [13]. We set the memory size limitation to 2 MBytes. When the execution code exceeds the cache size (2 MBytes in our implementation), the cache miss increases the execution time drastically. We converted the generated HMDD to the C-code, then, we obtained the execution code using the GCC compiler with the optimization option "-O3". Note that, we executed on Windows XP Service Pack 2.

*b) Prefetching HMDDM:* We built the prefetching HMDDM as same as the Core2Duo. We used three different limitations of the memory sizes; 1 MBytes, 2 MBytes, and 4 MBytes. The prefetching HMDDM requires one clock for the branch operation. We executed the prefetching HMDDM at 100 HMz, so the execution time is $APL \times 10$ nsec.

Table I compares the prefetching HMDDM with the other DDMs. In Table I, *NAME* denotes the benchmark name; *IN* denotes the number of inputs; *OUT* denotes the number of outputs; *APL* denotes the average path length; *MEM* denotes the amount of memory (KBytes); *TIME* denotes the execution time for a test vector (nsec); and *Ratio* denotes $\frac{TIME_{Core2Duo}}{TIME_{HMDDM}}$. From Table I, the prefetching HMDDM is 14.53-18.97 times faster than the Core2Duo. Especially, for the large function (s5378), we have the following: The twice memory size (2 MBytes) reduces 11.7% of the execution time, and the quadruple memory size (4 MBytes) reduces 41% of the execution time. On the other hand, as for small functions (apex6, cps, frg2), even if the memory size increased, the execution times reduced only slightly or did not reduce at all. Thus, the given memory size is enough for these functions. From the above discussion, when enough memory is available, we can decrease the execution time for the prefetching HMDDM.

## VI. CONCLUSION

This paper showed the prefetching HMDDM. The standard HMDDM requires two memory references to read the jump address and the index separately, while the prefetching HMDDM requires only one memory reference to them at a time. Thus, the prefetching HMDDM is faster than the standard HMDDM. We implemented the prefetching HMDDM on an Altera's FPGA. As for the execution time, the prefetching HMDDM is 14.53-18.97 times faster than the Core2Duo.

Since the HMDDM consists of the small FPGA (less than one watt [1]) and the off-chip RAM (several watts [9]), the estimated power consumption is several watts. On the other hand, that for the mobile ultra high energy efficient with TDP of the Core2Duo is 10-25 watts [8]. Thus, the HMDDM is the power-performance efficient processor.

## 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] Altera Corp., "Cyclone III FPGAs: Optimized for Low Power," http://www.altera.com/.

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

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

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

[5] DigiKey Corp., http://www.digikey.com

[6] Y. Iguchi, T. Sasao, and M. Matsuura, "Implementation of multiple-output functions using PROMDDs," *ISMVL2000*, May 23-25, 2000, pp.199-205.

[7] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno, "A hardware simulation engine based on decision diagrams," *ASPDAC2000*, Jan., 2000, pp.73-76.

[8] Intel Corp., "Intel Core2 processor families," http://www.intel.com/.

[9] W. Jiang, Q. Wang, and V. K. Prasanna, "Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup," *INFOCOM2008*, pp.1786-1794, 2008.

[10] 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.

[11] 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.

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

[13] S. Nagayama and T. Sasao, "Code generation for embedded systems using heterogeneous MDDs," *SASIMI 2003*, April, 2003, pp.258-264.

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

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

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

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

[18] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD93*, Nov., 1993, pp. 42-47.

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

[20] A. Thayse, M. Davio, and J. P. Deschamps, "Optimization of multi-valued decision algorithms," *ISMVL79*, May, 1979, pp.171-177.

[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.