J. of Mult.-Valued Logic & Soft Computing, Vol. 0, pp. 1–13
Reprints available directly from the publisher
Photocopying permitted by license only

A Heterogeneous Multi-valued Decision Diagram Machine for Encoded Characteristic Function for Non-zero Outputs

HIROKI NAKAHARA[1], TSUTOMU SASAO[2] AND MUNEHIRO MATSUURA[2]

[1]*Kagoshima University, Kagoshima, 890-0065, Japan*
E-mail: nakahara@eee.kagoshima-u.ac.jp
[2]*Kyushu Institute of Technology, Iizuka, 820-8502, Japan*
E-mail: sasao@cse.kyutech.ac.jp, matsuura@cse.kyutech.ac.jp

Received: September 28, 2011. Accepted: May 7, 2012.

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. A HMDD for encoded characteristic function for non-zero outputs (ECFN) represents a multi-output logic function efficiently. This paper shows an HMDD for an ECFN machine. First, we introduce the HMDD for ECFN. Then, we show an architecture for the HMDD for ECFN machine. Also, by experiment, we show that compared with the Intel's Core i5 processor running at 2.4 GHz, as for the speed, the HMDD for ECFN machine is 1.40-4.27 times faster, and as for the power-delay product, it is 15.1-46.4 times smaller.

Keywords: Decision diagram machine, heterogeneous multi-valued decision diagram

1 INTRODUCTION

Decision diagram machines (DDMs) are special purpose processors that evaluate logic functions [2, 10]. Applications for DDMs include industrial process controllers [21]; logic simulators [6]; and packet classifier [13]. A recent study shows that a 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 Intel's Core2Duo processor [14].

A decision diagram machine (DDM) consists of the control part and the memory storing the data for the nodes. Two types of implementations exist: One using on-chip memory on a high-performance-and-expensive device (e.g., the ASIC or the FPGA) [13,14]; and the other using off-chip off-the-shelf memory with a low-performance-and-inexpensive device (e.g., the CPLD) [6]. For the first type of the implementation, the cost per a bit is high*. On the other hand, consider the second type of the implementation, where off-the-shelf memory is used. We use a heterogeneous MDD (HMDD) machine that can utilize most of the given memory space by selecting the optimal partition of the input variables [11]. 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 HMDD machine to evaluate a single-output logic function, which has a good compatibility to the off-the-shelf memory [15]. In the previous work, we considered HMDD machines for multiple-output functions. As for the area time complexity, the HMDD machine for encoded characteristic function for non-zero outputs (ECFN) [18] is the best [12]. In this paper, first, we review the HMDD machine for ECFN. Then, we compare HMDD machines with the Intel's Core i5 Processor. Finally, we analyze delay and power-delay product for these processors.

The rest of the paper is organized as follows: Chapter 2 defines important words; Chapter 3 introduces the HMDD machine for ECFN; Chapter 4 shows the experimental results; and Chapter 5 concludes the paper.

This paper is built on the previous publication [12].

2 PRELIMINARY

Definition 1. *Let $f(X) : B^n \to B$ 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 denotes* **a super variable**. *When $k_i = |X_i| (i = 1, 2, \ldots, u)$, we have the relation $k_1 + k_2 + \cdots + k_u = n$.*

Definition 2. *A* **binary decision diagram (BDD)** *is obtained by applying* **Shannon expansions** *repeatedly to a logic function f [3]. 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*

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

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*))** *[9].*

Definition 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 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)** [4]. 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.

2.1 HMDD for Encoded Characteristic Function for Non-zero Outputs (ECFN)

The HMDD machine for single-output function has been developed [15]. However, many practical applications use multiple-output functions. Here, we consider the multiple-output representations for decision diagrams (DDs). A BDD for ECFN (Encoded Characteristic Function for Non-zero outputs) [18] requires smaller amount of memory than MT-HMDDs. A BDD for ECFN is considered as a generalization of a shared BDD (SBDD), and is often smaller than the corresponding SBDD [17]. This part shows the properties of BDD for ECFNs.

Definition 5. *Let n be the number of the inputs, and m be the number of the outputs. An ECFN represents the mapping:* $F : B^n \times B^u \to B$, *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 6. *For an m-output function f_i (i=0, 1, ..., m-1), the ECFN is*

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

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, $z^0 = \bar{z}$, $z^1 = z$, and $u = \lceil log_2 m \rceil$.

Example 1. *A four-output function (f_0, f_1, f_2, f_3) can be represented by the ECFN as follows: $F = \bar{z}_1 \bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1 \bar{z}_0 f_2 \vee z_1 z_0 f_3$.* ∎

Next, we extend the BDD for ECFN to the HMDD for ECFN.

Example 2. *Figure 1 shows the 2-bit adder consisting of three-output function (f_0, f_1, f_2). In this case, it can be represented by the ECFN as $F = \bar{z}_1 \bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1 f_2$. Figure 2 shows a BDD for ECFN for the 2-bit adder, while Figure 3 shows a HMDD for ECFN for the 2-bit adder.* ∎

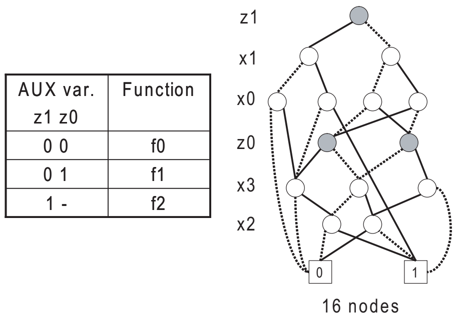
The evaluation for the HMDD for ECFN is more complex than that for an ordinary MDD. The following algorithm shows the evaluation method.

Algorithm 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.*

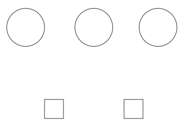
$$\begin{array}{r} x_0\ x_2 \\ +)\ x_1\ x_3 \\ \hline f_2\ \ f_1\ f_0 \end{array}$$

FIGURE 1
Two bit adder.



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

FIGURE 2
A BDD for ECFN for 2-bit adder.



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 [19].

Example 4. *Figure 4 shows the optimal encoding of the BDD for ECFN, that is different from the natural encoding shown in Figure 2. In these cases, the order of the input variables are the same. This example shows that the output encoding influences the sizes of the ECFNs.* ■

3 HMDD FOR ECFN MACHINE

3.1 Direct and Indirect Branch Address Placement [15]

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** [15]. In the direct branch address placement, the index and branch addresses are located to the same word. Although the field lengths may be 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 efficiently uses the memory, since the words have the same length.

Example 5. *Figure 6 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.

3.2 Architecture of HMDD for ECFN Machine

In the HMDD for ECFN, the non-terminal node is evaluated by **an indirect branch instruction** shown in Figure 7, while the terminal node is

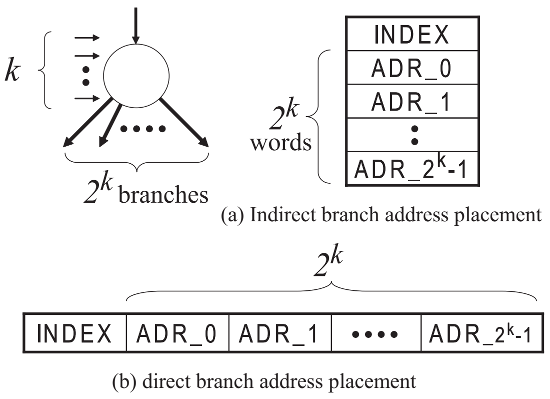


FIGURE 6
Two types of branch address placements.

evaluated by **a single-output and jump instruction** shown in Figure 8. Figure 9 shows **HMDD for ECFN machine (HMDDM for ECFN)**. In Figure 9, **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 auxiliary variable counter (AC)** retains the value of the auxiliary variable; **the double-rank shift register** retains the output value; and **the input selector** shown in Figure 10 selects both the primary inputs and the auxiliary variables from the AC.

Figure 11 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** [16]. The shift register retains outputs of the HMDDM for ECFN. When all outputs are evaluated, the value of the shift register is sent to the output register.

The following algorithms show the execution of the branch instruction and the output instruction for the HMDDM for ECFN.

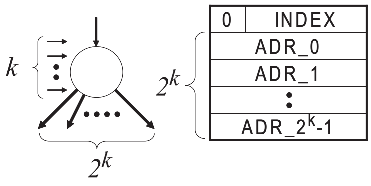


FIGURE 7
An indirect branch instruction.

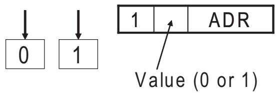


FIGURE 8
Single-output and jump instruction.

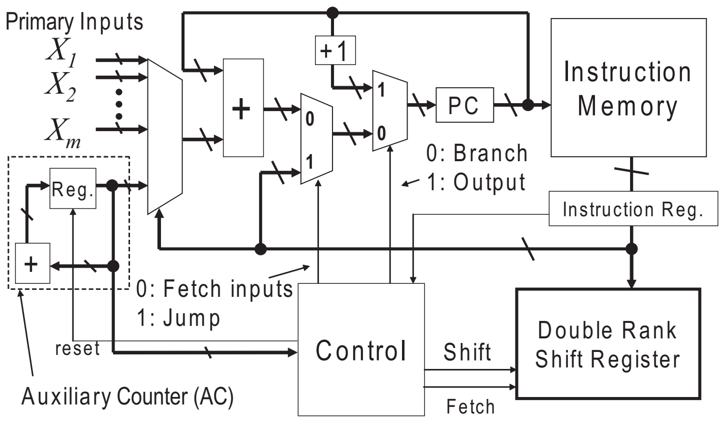


FIGURE 9
HMDD for ECFN machine (HMDDM for ECFN).

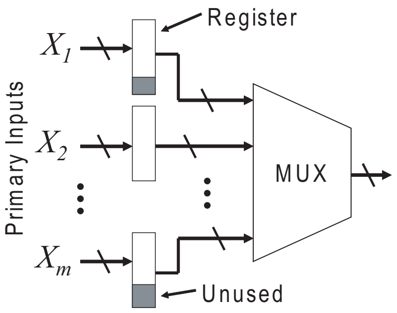


FIGURE 10
Input selector.

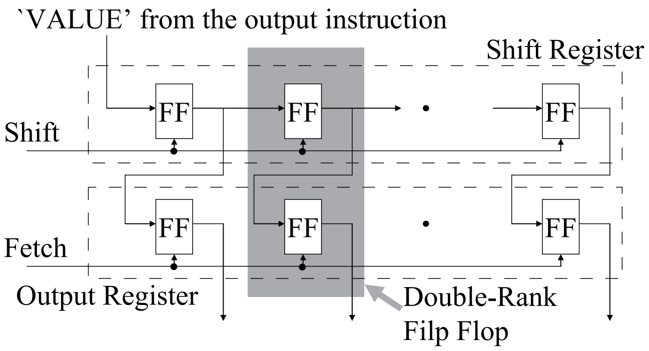


FIGURE 11
Double-rank shift register.

Algorithm 2. *(2^k indirect branch instruction for the HMDDM for ECFN)*

```
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 3. *(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.
    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 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, and A_{ECFN} be the number of addresses for the HMDDM for ECFN. Then, we have

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

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

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

Therefore, the memory size for the HMDDM for ECFN is

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

4 EXPERIMENTAL RESULTS

4.1 Implementation of HMDDM for ECFN

We implemented HMDDM for ECFN on the Altera Cyclone III starter kit (FPGA: Cyclone III, EP3C25). Then, we obtained the necessary number

```
// C-code
switch((in[x] >> y)& z){
case 1: goto N_1; break;
case 2: goto N_2; break;
case 3: goto N_3; break;

:

case 2^{k-1}: goto N_{2^{k-1}}; break;
}
```

```
// Assembly (ASM)-code
mov in+x, %eax
sal $y, %eax
and $z, %eax
test $-858993460, %eax
jmp *Label %eax
Label:
.long N_1
.long N_2
:
.long N_{2^{k-1}}
```

FIGURE 12
C-code and assembly-code for a node of an HMDD for ECFN.

of logic elements (LEs) and the maximum clock frequency. For the FPGA synthesis tool, we used QuartusII (v.9.1). The tool produced a circuit that consumes 239 LEs, and works with the maximum clock frequency of 110.1 MHz.

4.2 Comparison With Intel's Core i5 Processor

Delay Time

We compared the delay time for the HMDDM for ECFN with the Intel's Core i5 2520M (2.5 GHz, Smart Cache 3MBytes on Windows 7) using MCNC benchmark function [20]. To construct the HMDD for ECFN, the memory size limitation is set to the cache size of the Core i5 processor (3 [MBytes]). To evaluate a non-terminal node for the HMDD for ECFN, the assembly-code for Core i5 shown in Figure 12 performs the following operations:

1. Read an input variable by *mov* instruction.
2. Extract the specified bit by *sal* (shift) and *and* instructions.
3. Perform the indirect branch by *test* and *jmp* instructions.

The execution code for the HMDD for ECFN was generated by gcc compiler with optimization option -O3. To obtain the delay time per a vector ([nsec]), we generated random test vectors, and obtained the average delay time excluding the time for the reading and writing vectors. The operating frequency for the HMDDM for ECFN was 100 [MHz], while that for the Core i5 processor was 2.5 [GHz].

Table 1 compares the delay time for the Core i5 processor and the HMDDM for ECFN. In Table 1, *Name* denotes the name of benchmark

Name	In	Out	# of Nodes	APL	Core i5 2.5 GHz	HMDDM 100 HMz	Ratio
alu4	14	8	141	1.57	717	251	2.85
apex2	39	3	260	4.42	593	265	2.24
cc	21	26	27	2.16	2871	1123	2.56
lal	26	19	60	2.58	3027	980	3.09
pcler8	27	17	30	2.10	2714	714	3.80
spla	24	21	170	1.50	2355	630	3.74
ttt2	16	46	51	2.25	6801	2070	3.29
ts10	22	16	31	1.30	1775	416	4.27
C1355	41	32	10296	8.16	7316	5222	1.40
C1908	33	25	3299	5.06	4742	2530	1.87
C3540	50	22	17802	5.70	4801	2511	1.91

TABLE 1
Comparison of delay time [nsec].

function; *In* denotes the number of inputs; *Out* denotes the number of outputs; *# of Nodes* denotes the number of nodes for the HMDD for ECFN; *APL* denotes the average path length for the HMDD for ECFN; and *Ratio* denotes the that of the delay time (Core i5/HMDDM for ECFN). Table 1 shows that the HMDDM for ECFN is 1.40-4.27 times faster than the Core i5 processor.

Power-Delay Product

The power-delay product is an important measure of a circuit. To measured the power-delay product, first, we obtained the delay time [nsec] shown in Table 1. Then, we measured the momentary power consumption [W/nsec]. Finally, we calculated the power-delay product by multiplying them.

To make the comparison fair, we tried to make the temperature of the HMDDM for ECFN and the Core i5 processor the same. As for the HMDDM for ECFN, we assume that the momentary 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 for the HMDDM for ECFN was 9.017 [V], and the measured current per one second was 0.104 [A/sec]. Thus, the momentary power consumption for the HMDDM for ECFN is 0.937×10^9 [W/nsec]. As for the Core i5 processor, to obtain the momentary power consumption, we measured Panasonic Corp. Let’s note CF-9 computer. To obtain the pure momentary power consumption for the Core i5 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 for the Core i5 processor was 16.201 [V], and the passage of a current per one second was 0.626 [A/sec]. Thus, the momentary

Name	**In**	**Out**	**Core i5 2.5 GHz**	**HMDDM 100 HMz**	**Ratio**
alu4	14	8	7275.2	236.1	30.8
apex2	39	3	6017.0	249.3	24.1
cc	21	26	29131.1	1055.8	27.5
lal	26	19	30714.0	921.6	33.3
pcler8	27	17	27538.1	671.2	41.0
spla	24	21	23895.4	592.2	40.3
ttt2	16	46	69007.6	1945.9	35.4
ts10	22	16	18010.4	391.1	46.6
C1355	41	32	74233.2	4909.2	15.1
C1908	33	25	48115.6	2378.3	20.2
C3540	50	22	48714.2	2360.5	20.6

TABLE 2
Comparison of Power-Delay Product [$10^9 \cdot W/nsec$].

power consumption for the HMDDM for ECFN is 10.141×10^9 [W/nsec]. Therefore, the momentary power consumption for the HMDDM for ECFN is 9.2% of that for the Core i5 processor.

Table 2 compares the power-delay product of the HMDDM for ECFN and the Core i5 processor. Table 2 shows that, as for the power-delay product, the HMDDM for ECFN is 15.1-46.6 times smaller than the Core i5 processor.

5 CONCLUSION

In this paper, we considered the HMDDM for ECFN. We showed the architecture for the HMDDM for ECFN, and introduced instructions evaluating the non-terminal node and the terminal node. We implemented the HMDDM for ECFN on Altera's FPGA board. Compared with the Intel's Core i5 2.4GHz processor, as for the speed, the HMDD for ECFN machine is 1.40-4.27 times faster, and as for the power-delay product, it is 15.1-46.4 times smaller.

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 Prof. Shinobu Nagayama was quite useful. Reviewers comments were useful to improve the paper.

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. Compt.*, Vol. 54, No. 9, pp. 1041–1053, Sep. 2005.

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

[6] 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.

[7] Intel Corp., “Intel Core i5 processor families,” http://www.intel.com/.

[8] 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.

[9] 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.

[10] 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.

[11] S. Nagayama and T. Sasao, “Code generation for embedded systems using heterogeneous MDDs,” *SASIMI 2003*, April, 2003, pp. 258–264.

[12] H. Nakahara, T. Sasao, and M. Matsuura, “A Comparison of heterogeneous multi-valued decision diagram machines for multiple-output logic functions,” *41st Int’l Symp. on Multiple-Valued Logic (ISMVL2011)*, Tuusula, Finland, May 23–25, 2011, pp. 125–130.

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

[14] 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.

[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] 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.

[17] 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.

[18] 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.

[19] 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.

[20] S. Yang, “Logic synthesis and optimization benchmark user guide version 3.0,” *MCNC*, Jan. 1991.

[21] 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.