A Low Power-Delay Product Processor Using Multi-valued Decision Diagram Machine

[1]Hiroki Nakahara [2]Tsutomu Sasao [2]Munehiro Matsuura
[1]Kagoshima University, Japan [2]Kyushu Institute of Technology, Japan

Abstract— A heterogeneous multi-valued decision diagram of encoded characteristic function for non-zero outputs (HMDD for ECFN) represents a multi-output logic function efficiently. As for the speed, the HMDD for ECFN machine is 3.02 times faster than the Core i5 processor, and is 12.50 times faster than the Nios II processor. As for the power-delay product, it is 32.72 times lower than the Core i5 processor, and is 57.92 times lower than the Nios II processor.

I. Introduction

Decision diagram machines (DDMs) are special purpose processors that evaluate logic functions [6]. Applications for DDMs include industrial process controllers; logic simulators; and packet classifiers. The previous work considered heterogeneous multi-valued decision diagram (HMDD) machines for multiple-output functions [3]. As for the area time complexity, the HMDD machine for encoded characteristic function for non-zero outputs (ECFN) [4] is the best. In this paper, we compare with the Intel's Core i5 Processor and the Altera's Nios II embedded processor with respect for the delay time and the power-delay product.

II. HMDD for ECFN

A. Multi-valued Decision Diagram (MDD)

Definition 2.1 *A* **binary decision diagram (BDD)** *is obtained by applying* **Shannon expansions** *repeatedly to a logic function f [1]. 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))** *[2].*

Definition 2.2 *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)**.

B. HMDD for ECFN

Here, we consider representation of multiple-output function by decision diagrams (DDs). A BDD for ECFN (Encoded Characteristic Function for Non-zero outputs) [4] requires smaller amount of memory than MTBDD (Multi-Terminal BDD). This part shows the properties of a BDD for ECFN.

Definition 2.3 *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 representation of the binary vector $\vec{b}$. For an m-output function f_i (i=0, 1, ..., m-1), the ECFN is $F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i$, 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 $u = \lceil log_2 m \rceil$.*

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

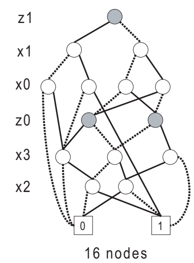

Fig. 1. A BDD for ECFN for 2-bit adder.

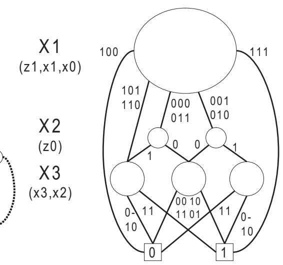

Fig. 2. A HMDD for ECFN for 2-bit adder.

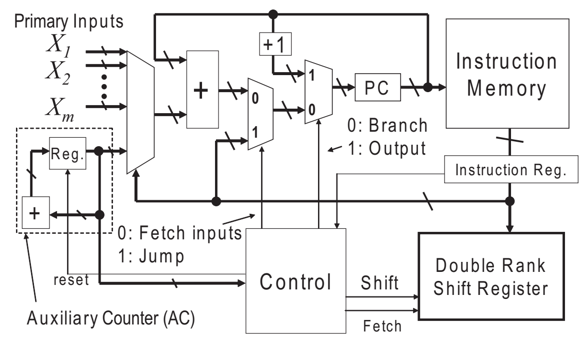

Fig. 3. HMDD for ECFN machine (HMDDM for ECFN).

Example 2.1 *Fig. 1 shows a BDD for ECFN for the 2-bit adder, while Fig. 2 shows a HMDD for ECFN for the 2-bit adder.* ■

III. HMDD for ECFN Machine

A. Architecture of HMDD for ECFN Machine

In the HMDD for ECFN, the non-terminal node is evaluated by **an indirect branch instruction**, while the terminal node is evaluated by **a single-output and jump instruction**. Fig. 3 shows **HMDD for ECFN machine (HMDDM for ECFN)**. In Fig. 3, **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

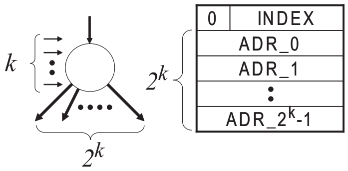

Fig. 4. An indirect branch instruction.

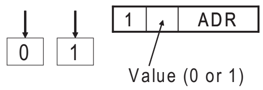

Fig. 5. Single-output and jump instruction.

variable; **the double-rank shift register** retains the output value; and **the input selector** selects both the primary inputs and the auxiliary variables from the AC.

Fig. 4 shows **the indirect branch instruction** to evaluate a non-terminal node for the HMDD, while Fig. 5 shows **the single-output and jump instruction** to evaluate a terminal node. The following algorithms show the execution of instructions for the HMDDM for ECFN.

Algorithm 3.1 *(2^k indirect branch instruction)*

1. *Read indirect branch address*
   First, read the index corresponding to index filed in the branch instruction. Then, add it to the PC to obtain the indirect branch address.
2. *Perform the jump operation*
   First, read the jump address corresponding to the PC. Then, set the jump address to the PC.

Algorithm 3.2 *(Single-output and jump instruction). 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.*
   First, read the value and the jump address corresponding to the PC. Then, set the value to the double-rank shift register, and $AC \leftarrow AC + 1$. Next, 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 3.1.*

IV. Experimental Results

We implemented an HMDDM for ECFN on the Altera Cyclone III starter kit (FPGA: Cyclone III, EP3C25). For the FPGA synthesis tool, we used QuartusII (v.9.1). The tool produced a circuit that consumes 239 logic elements (LEs), and works with the maximum clock frequency of 110.1 MHz.

We compare it with the Intel's Core i5 processor running at 2.4 GHz and the Altera's Nios II embedded processor running at 100 MHz. To measure the power-delay product, first, we obtained the delay time per one random test vector [nsec/work] shown in Table I using MCNC benchmark functions [5]. To obtain delay time for the Nios II processor and the Core i5 processor, we generated C-code for the HMDD for ECFN. Then, we generated the executable code using gcc compiler with optimize option -O3. As for the delay time, the HMDDM for ECFN is 3.02 times shorter than the Core i5 processor, and it is 12.50 times shorter than the Nios II processor.

TABLE I
Comparison of Delay Time (nsec/work) and Power-Delay Product (10^{-9}W$\cdot$ work).

Name	I/O	Delay Time			Power Delay Product		
		Core i5 2.4GHz	Nios II 100MHz	HMDDM 100MHz	Core i5 2.4GHz	Nios II 100MHz	HMDDM 100MHz
apex2	39/3	593	3315	265	6013	14401	248
cc	21/26	2871	14040	1123	29114	60995	1053
lal	26/19	3027	12255	980	30696	53240	919
pcler8	27/17	2714	8925	714	27522	38773	669
spla	24/21	2355	7875	630	23882	34212	590
ttt2	16/46	6801	25875	2070	68968	112411	1941
ts10	22/16	1775	5200	416	18000	22590	390
C1355	41/32	7316	65280	5222	74191	283601	4897
Ratio		3.02	12.50	1.00	32.72	57.92	1.00

We measured the momentary power consumption [10^{-9}W/nsec]. As for the HMDDM for ECFN running at 100 MHz, the momentary power consumption for the HMDDM for ECFN is 0.937×10^{-9} [W/nsec]. As for the Nios II processor running at 100 MHz, it was 4.344×10^{-9} [W/nsec]. As for the Core i5 processor, it was 10.141×10^{-9} [W/nsec]. Then, we calculated the power-delay product by multiplying them. Table I compares power-delay products. As for the power-delay product, the HMDDM for ECFN is 32.72 times lower than the Core i5 processor, and it is 57.92 times lower than the Nios II processor.

V. Conclusion and Comment

This paper compared the HMDDM for ECFN with the Intel's Core i5 general purpose processor and the Altera's Nios II embedded processor. Experimental results using MCNC benchmark function showed that the HMDDM for ECFN is the power-delay efficient processor. The future work is to use practical application in the comparison.

VI. 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).

References

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

[2] 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, 1998, pp. 9-62.

[3] H. Nakahara, T. Sasao, and M. Matsuura, "A Comparison of heterogeneous multi-valued decision diagram machines for multiple-output logic functions," *ISMVL2011*, 2011, pp.125-130.

[4] 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*, 2001, pp. 406-411.

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

[6] 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.