A Hardware Simulation Engine Based on Decision Diagrams

Yukihiro IGUCHI†
Munehiro MATSUURA‡
†Dept. of Computer Science
Meiji University
Kawasaki 214-8571, JAPAN

Tsutomu SASAO‡
Atsumu ISENO†
‡Dept. of Electronics and Computer Science
Kyushu Institute of Technology
Iizuka 820-8502, JAPAN

Abstract – A hardware logic simulation engine based on decision diagrams is presented. For the data structure of the engine, we propose PMDDs (Paged reduced ordered Multi-valued Decision Diagrams). A unit of this engine consists of memory (RAMs) and control circuits: RAMs store the PMDD data, and the control circuits trace the edges according to the input vectors. The engine consists of several units, and is accelerated by pipelining. Experimental results using a prototype are shown.

Keywords – Logic simulation, Hardware simulation engine, BDD, MDD, PMDD.

I. Introduction

In this paper, we propose a cycle-based hardware logic simulation engine, or an engine for short. The paper is organized as follows: Section 2 introduces logic simulation based on decision diagrams. Section 3 presents an engine based on a PMDD (Paged reduced ordered Multi-valued Decision Diagrams). Section 4 shows performance evaluation and a preliminary experimental result.

II. Logic Simulations Based on Decision Diagrams

From here, we will review simulation methods based on DDs (decision diagrams). These methods are theoretically much faster than LCC-based ones. In a BDD (binary decision diagram)[2], each node corresponds to a variable, and edges labeled with 0 and 1 represent $low(v)$ and $high(v)$, respectively. We only consider ordered decision diagrams, where the input variables appear in a fixed order on all the paths from the root node to a terminal node.

We can evaluate the function by traversing the BDD from the root node to a terminal node. Clearly, the evaluation time for an n input logic function is $O(n)$. Because the simulation time of an LCC-based gate level simulator is $O(n^2)$, a DD-based one is $O(n)$ times faster than an LCC-based one. By using an MDD (multi-valued decision diagram) [6], we can make the simulator [3] faster. To represent multiple-output functions, we use SMDD (Shared reduced ordered Multi-valued Decision Diagram). From here, SMDDs are simply denoted by MDDs.

Definition 1 *An MDD(k) is the multi-valued decision diagram where each non-terminal node has 2^k edges. Note that an MDD(1) and a BDD are the same.*

Since an MDD(k) evaluates k binary variables at a time, a logic simulator based on an MDD(k) is k times faster than one based on a BDD. An MDD(k) is derived from the corresponding BDD easily [3].

Example 1 *Fig. 1(a) shows the BDD for an 8-input 2-output function. Partition the input variables into (X_1, X_2, X_3, X_4), where $X_1 = (x_1, x_2)$, $X_2 = (x_3, x_4)$, $X_3 = (x_5, x_6)$, and $X_4 = (x_7, x_8)$. Then, we have the MDD(2) in Fig. 1(b). By using MDD(2), the simulation time is reduced into a half, because the path-length from the root node to the terminal nodes is a half of the BDD.* ∎

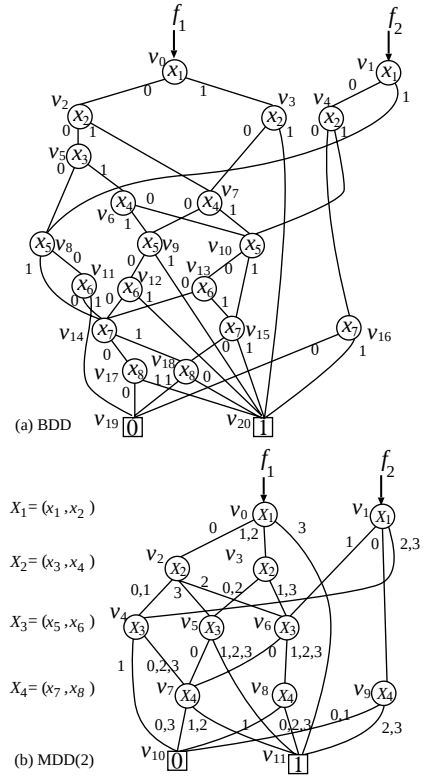

Fig. 1. BDD and MDD(2).

III. Hardware Simulation Engine

To speed up the engine, we use the following three methods:

1. Use the DD-based simulation. By this, the engine will be $O(n)$ times faster than gate level logic simulators.
2. Use MDD(k)s instead of BDDs. By this, the engine will be k times faster.
3. Use a pipeline of r processing units. By this, the engine will be r times faster.

A. Operation of Simulation Engine

Fig. 1 shows the concept of the simulation system. The proposed engine consists of memories and control circuits. The host computer prepares the data for MDDs representing the simulation target, and send them to the memories

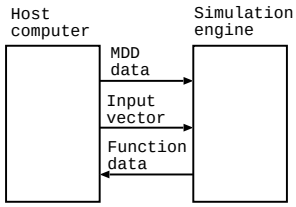

Fig. 1. Concept of the simulation system.

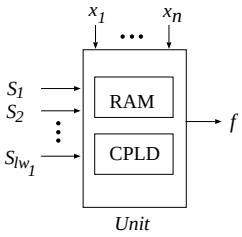

Fig. 2. Single-unit engine.

of the engine. The host computer also generates input vectors, and send them to the engine. The engine traverses the MDD according to the values of input vectors, and returns the value of the function to the host computer. Although the simulation speed is bounded by the communication speed between the engine and the host computer, we will not consider it here.

First, we will illustrate the operation of the BDD-based engine using only one processing unit. Fig. 2 shows the single-unit engine. The RAMs store the BDD data: Each non-terminal node has its index and two next addresses for the 0-edge and the 1-edge.

For an m-output function, we have to traverse the BDD m times. Thus, the engine shown in Fig. 2 requires the memory accesses of $O(n \cdot m)$.

B. Speedup of the Engine

In this part, we will show two methods to speed up the engine. First, by using MDD(k), we will make it k times faster. Second, by using r processors, we will make it r times faster.

B.1. MDD(k)

The data for an MDD(k) are stored in the memory similarly to the case of BDDs. Because the memory access in an MDD(k) is reduced by a factor of k, the engine will be k times faster than the BDD-based one. However, in general, MDD(k)s require more memory than BDDs.

Example 1 *Fig. 3 shows the MDD(2) data for Fig. 1. Along the arrows, by traversing the memory from the address 0 to 11, we have the value* $f_1(x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8)=f_1(0,1,1,1,1,1,1,0) = 1$. *As shown in Fig. 1(b), the MDD(2) has non-terminal nodes. While, BDD=MDD(1) has 19 non-terminal nodes. However, each non-terminal node in MDD(k) requires* 2^k *next addresses.* ∎

B.2. Speedup by Pipelining

When only one memory system is used to store a decision diagram, we can evaluate the value for only one input vector at a time. In this part, we propose a PMDD(k, r) (Paged reduced ordered Multi-valued Decision Diagram), which is an MDD(k) partitioned into r pages. Fig. 4 shows the concept of the engine having r processing units. In the PMDD-based engine, each processing unit has an independent memory system and a control circuit. Since the number of nodes in each page is smaller than the case of the single MDD(k), the memory for storing the index and the next addresses can be reduced. Since these units work in parallel, the engine consisting of r processing units has the r-fold throughput.

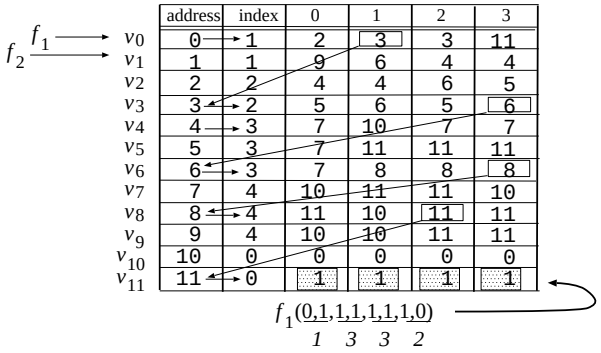

Fig. 3. MDD(2) data for Fig. 1(b) and simulation procedure.

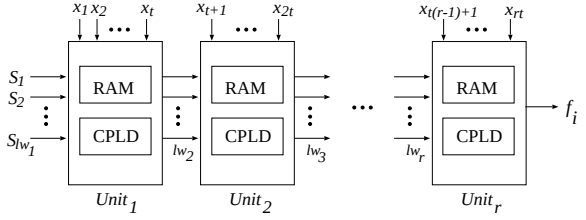

Fig. 4. Concept of the pipelined engine.

Definition 1 *Basic terminology and parameters of the engine are follows:*

- *Page: A memory having t primary inputs.*
- *Unit: A hardware consisting one page of memory and a control circuit.*
- *Pipeline: A series connection of r units.*
- *n: The number of the primary input variables.*
- *m: The number of the primary outputs.*
- *r: The number of pages = the number of units = the number of pipeline steps.*
- *t: The number of the primary input variables in a page.*
- *w_i $(i = 1,2,\ldots,r)$: Width of an MDD(k), where $w_1 = m$.*
- *k: The number of control variables in an MDD (in the case of a BDD, $k = 1$).*

Definition 2 *(Paged reduced ordered MDD: PMDD(k, r).)*
Let the input variables be $X = (x_1, x_2, \ldots, x_n)$, where $n = t \cdot r$. Consider the MDD which always has nodes at the $(t \cdot s + 1)$th level in all the path from root nodes to terminal nodes $(s = 1,\ldots,r-1)$. The first page consists of the level 1 through the level t; the second page consists of the level $t+1$ through the level $2t$; and so on. A PMDD has the following properties:

1. *In a page, different nodes represent different functions. (Two nodes in different pages may represent the same function.)*
2. *Each edge from a node is connected to an another node in the same page, or to a node in the 1st level of the next page.*

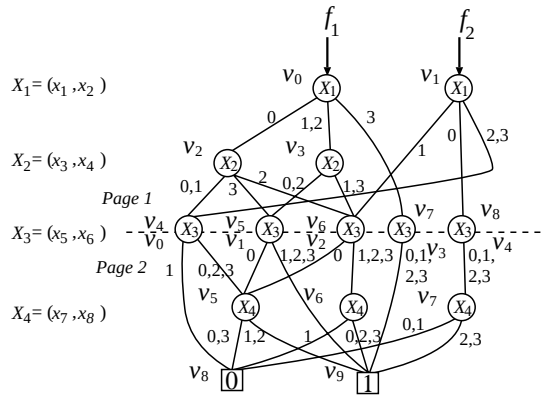

Fig. 5. PMDD(2, 2) for the function in Fig. 1(b).

A PMDD(k, r) denotes an MDD(k) partitioned into r pages. Note that a PMDD(1, 1) is an BDD, while a PMDD(1, n) is a QROBDD (Quasi-Reduced Ordered BDD). In the QROBDD, every variable appears along every path from the root node to the constant nodes [7]. A PMDD(k, r) is the MDD(k) consisting of r pages where nodes always exist in the 1st level of each page.

Let $size(DD, f)$ be the number of nodes in the DD for f. Then, we have the following:

Theorem 1
$size(PMDD(1,1), f) \leq size(PMDD(1,r), f)$
$\leq size(PMDD(1,n), f)$, *where* $1 \leq r \leq n$.

A PMDD(k, r) has the following merits:

1. It is a data structure suitable for pipelining.
2. Since the next addresses are limited to a page, the size for the next addresses and the index can be smaller.

Example 2 *Let us partition the the MDD(2) in Fig. 1(b) into two pages. Fig.5 is the PMDD(2, 2). In this case,* x_1 *through* x_4 *are in the first page, and* x_5 *through* x_8 *are in the second page. Note that some non-terminal nodes are appended to the boundary of the pages. The processor of the first page transfers the start address to the processor of the second page. Let width* (w_i) *be the number of nodes in the boundary, where* $w_1 = m$. *In Fig. 4, inputs* $S_1, \ldots, S_{l_{w_1}}$ *in the first unit specifies the function to be evaluated. In Fig. 5,* $w_1 = 2$ *and* $w_2 = 5$. *Fig. 6 shows the memory data for the PMDD(2, 2) in Fig. 5.* ∎

To construct a PMDD(k, r), we use the following strategies:

- Partition the MDD(k) into the pages so that the numbers of variables in the units are almost the same. This will reduce the cycle time of the pipelined engine.
- Partition the MDD(k) into the pages so that each page can be stored in the memory of the unit. For example, the boundary of the page should be the level with a small width. If necessary, the variable ordering of the MDD(k) should be changed.

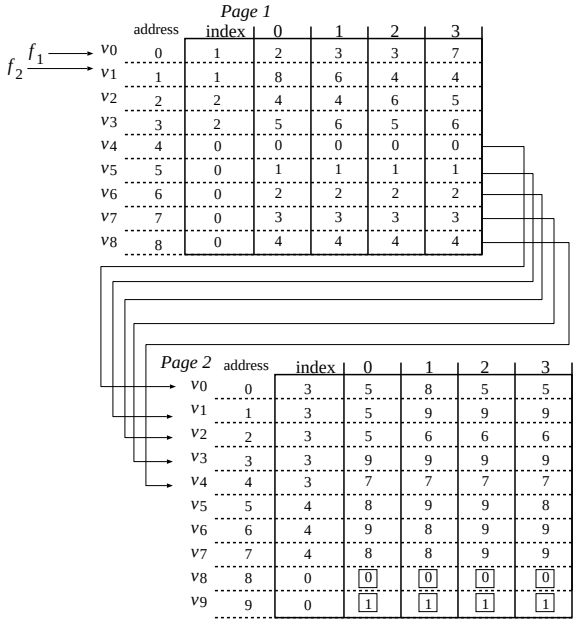

Fig. 6. Memory representation of PMDD(2, 2) shown in Fig. 5.

A non-pipelined engine based on a PMDD(1, 1) requires $n \cdot p$ steps to simulate p input vectors. A pipelined engine based on a PMDD(k, r) requires $(n \cdot p/k)$ steps to simulate for p input vectors, while the throughput is r times of PMDD$(k, 1)$: In each $n/(k \cdot r)$ step, we obtain the simulation result. Therefore, the speedup rate for PMDD(k, r) over PMDD$(1, 1)$ is $k \cdot r$.

IV. Performance Evaluation

A. Memory Requirement for PMDD(k, r)

Table 1 compares the number of nodes in PMDD$(k, 1)$, $(k = 1, 2, 3, 4, 5)$ for ISCAS benchmark circuits. For example, C3540 has 50 inputs and 22 outputs, and the PMDD$(k, 1)$ $(k = 1, 2, 3, 4, 5)$ has 34689, 24478, 19122, 16841, and 12089 non-terminal nodes, respectively. Table 1 shows that the number of nodes decrease when k is increased. Note that each node requires 2^k addresses. We used the heuristic algorithm [4] to obtain the ordering of the input variables for PMDD$(k, 1)$s. However, the ordering that minimizes the BDD does not always minimize the PMDD$(k, 1)$ $(k \geq 2)$. For C6288, which is the 16-bit multiplier, we could not build the PMDD, so it is not shown in the table. For other benchmark functions, we could build the PMDDs, and their widths (i.e., the number of nodes in the boundary) were not so large.

Definition 1 *Let* $MT(k, r)[Words]$ *be the total amount of memory for the PMDD(k, r). Let #max_nodes be the maximum number nodes in one page among all the pages in the PMDD(k, r). We assume that the memory size of the engine is a multiple of 16. That is, one word consists of 16 bits. Let* $\lceil a \rceil$ *denote the smallest integer greater than or equal to a. We also assume that each page has the same amount of memory.*

$MT(k, r)$ is derived as follows:

- For an *index*, we need $\lceil \lceil \log_2 \lceil n/k \rceil \rceil / 16 \rceil$ words.

TABLE 1
NUMBER OF NODES IN PMDD(k, 1).

Func	In	Out	PMDD(k,1)				
			$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
C432	36	7	1069	619	542	326	380
C499	41	32	27845	13543	9029	6319	5797
C880	60	26	4141	3030	2544	1924	1728
C1908	33	25	7432	4426	2869	2379	1589
C2670	233	140	2712	2345	1932	1779	1635
C3540	50	22	34689	24478	19122	16841	12089
C5315	178	123	2446	1968	1620	1462	1346
C7552	207	108	2841	2301	1913	1655	1462

TABLE 2
TOTAL MEMORY SIZE $MT(1,r)$.

Func	In	Out	$MT(1,r)$					
			1	2	4	8	16	32
C432	36	7	6	6	6	6	12	12
C499	41	32	96	192	192	384	384	384
C880	60	26	24	24	48	48	96	96
C1908	33	25	24	48	48	48	96	96
C2670	233	140	12	24	24	48	48	96
C3540	50	22	192	192	384	384	768	768
C5315	178	123	12	12	24	24	48	48
C7552	207	108	12	12	12	24	24	48

[kilo words]

- For a pointer addressing the next node, we need $\lceil \lceil \log_2(\#max\_nodes) \rceil / 16 \rceil$ words.
- For each node, we need 2^k pointers.
- The total number of nodes in one page is $2^{\lceil log_2(\#max\_nodes) \rceil}$.
- The total number of pages is r.

Thus, the total amount of memory is obtained as:

$$\begin{aligned} MT(k,r) = & (\lceil \lceil \log_2 \lceil n/k \rceil \rceil / 16 \rceil + 2^k \\ & \times \lceil \lceil \log_2(\#max\_nodes) \rceil / 16 \rceil) \\ & \times 2^{\lceil \log_2(\#max\_nodes) \rceil} \times r[words] \end{aligned} \quad (1)$$

By using an PMDD(k,r), we can make the simulation $r \cdot k$ times faster than an PMDD(1, 1).

B. Prototype of the Simulation Engine

We developed a prototype of an engine based on PMDD(1, 2)s. Each unit consists of 160 kilo bytes of SRAM and control circuits implemented by XILINX CPLDs (XC95108-10PC84). The benchmark functions are represented by PMDD(1, 2)s. Input vectors are random patterns generated by an LFSR (linear feedback shift register). The prototype engine works at 18MHz. Table 4 compares performance of the engine with the software simulator [3]. By the limitation of the memory, we could not perform the simulation for C499 and C3540.

TABLE 3
TOTAL MEMORY SIZE $MT(k,r)$ FOR C2670.

		r					
		1	2	4	8	16	32
k	1	12	24	24	48	48	96
	2	20	40	40	40	80	160
	3	18	36	72	72	144	144
	4	34	68	136	136	136	272
	5	66	132	264	264	528	528

[kilo words]

TABLE 4
COMPARISON OF PMDD(1,2)-BASED ENGINE WITH SOFTWARE SIMULATION.

Function	In	Out	Prestissimo	Engine
C432	36	7	3.2	7.0
C880	60	26	19.2	43.4
C1908	33	25	7.6	23.5
C2670	233	140	55.6	910.0
C5315	178	123	112.8	607.6
C7552	207	108	46.5	624.2

[μsec/vector]

C. Observation

A preliminary experiment by using PMDD(1, 2) demonstrates the usefullness of the approach. If we use PMDD(k,r)s with $r = 8 \sim 16$, and $k = 4$, then we can make it $16 \sim 32$ times faster. If we use the DDs that have multiple terminals $(0 \cdots 00)$, $(0 \cdots 01)$,..., and $(1 \cdots 11)$ instead of two terminals, then several functions can be evaluated at the same time [5]. Another method to speed up is to use characteristic functions [1, 3]. Such reconfiguration is possible by just changing the program for CPLDs. By using these methods, we can build a high performance engine without increasing the clock frequency.

V. CONCLUSION

In this paper, we proposed a hardware logic simulation engine based on PMDDs. It is a pipelined MDDs with control circuits, and can be faster than the corresponding software implementation. The preliminary experiment using a prototype showed promising results. With these results, we are building a larger scale engine. Experimental results for larger configurations will be reported in the final paper.

ACKNOWLEDGMENTS

The authors thank Dr. Yusuke Matsunaga of Fujitsu Laboratory, and Dr. Kiyoshi Oguri of NTT Network Innovation Labs for their constructive comments. This work was supported in part by the Grant in Aid for Scientific Research of the Ministry of Education, Science, Culture and Sports of Japan.

REFERENCES

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *ICCAD'95*, pp. 408–412, Nov. 1995.

[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] P. C.McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD'95*, pp. 402–407, Nov. 1995.

[4] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD-93*, pp. 42–47, 1993.

[5] T. Sasao and J. T. Butler, "A method to represent multiple-output switching functions by using multi-valued decision diagrams," *IEEE International Symposium on Multiple-Valued Logic*, pp. 248–254, Santiago de Compostela, Spain, May 29-31, 1996.

[6] A. Thayse, M. Davio, and J. P. Deschamps, "Optimization of multiple-valued decision algorithms," *ISMVL-79*, Rosemont, IL. pp. 171–177, May 1978.

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