Implementation of Multiple-Output Functions using PQMDDs

Yukihiro IGUCHI[†] Tsutomu SASAO [‡] Munehiro MATSUURA[‡]

[†]*Dept. of Computer Science*
Meiji University
Kawasaki 214-8571, JAPAN
e-mail: iguchi@cs.meiji.ac.jp

[‡]*Dept. of Electronics and Computer Science*
Kyushu Institute of Technology
Iizuka 820-8502, JAPAN
e-mail: {sasao, matsuura}@cse. kyutech.ac.jp

Abstract

A sequential realization of multiple-output logic functions is presented. A conventional sequential realization is based on SBDDs (Shared reduced ordered Binary Decision Diagrams). In this paper, we propose PQMDD (Paged Quasi-reduced ordered Multi-valued Decision Diagram) as a new data structure. A function is represented by a PQMDD, and stored in memory. Dedicated control circuits traverse the PQMDD in parallel. We represent multiple-output function for benchmark functions by SBDDs and PQMDDs and compare the size of memory and computation time.

Keywords – multiple-output logic function, BDD, MDD, PMDD, PQMDD.

1. Introduction

Multiple-output logic functions can be implemented by three methods: The first method uses a dedicated multi-level combinational network. However, developing dedicated LSI is expensive and time consuming. Also, the modification of the function is quite expensive. The second method uses ROMs (Read Only Memories) or PLAs (Programmable Logic Arrays) to realize combinational network[14]. This method often requires too large network, when n, the number of the input variables, is large. The third method uses a general-purpose microprocessor or a special sequential circuit and ROMs to store the program. This is the most flexible method and quite useful when the speed is not so important. However, the sequential implementation is 100 to 1000 times slower than the combinational implementation.

This paper presents a method to implement multiple-output logic functions by sequential networks: They are faster than the conventional microprocessor realizations.

To explain the idea, for simplicity, consider an n-variable two-valued logic function $f : B^n \to B$.

First, represent f by a BDD (Binary Decision Diagram) [8, 3], and then replace each non-terminal node by an "*if then else*" statement, and we have a branching program to realize f[1]. A general-purpose microprocessor can be used to evaluate the logic function f. In this case, the evaluation time is $O(n)$.

To reduce the instruction fetch time, we can use a dedicated sequential circuit[15, 5] that traverses the BDD data structure stored in ROM. In this case, the size of ROM is proportional to the number of nodes in the BDD. Next, by grouping k two-valued variables to make multiple-valued variables, we can make an MDD (Multi-valued Decision Diagram) [15, 11, 2, 13]. The hardware that traverses MDD data structure is k times faster than BDD based one.

Most applications require multiple-output functions. Let the number of outputs be m. To evaluate m-output functions, we have to traverse SMDD (Shared Multi-valued Decision Diagram) m times. In this paper, we propose PMDDs (Paged reduced ordered Multi-valued Decision Diagrams), where, an SMDD is partitioned into r parts to speed up the evaluation time for multiple-output functions. In the PMDD, r modules evaluate the function in parallel, and the evaluation is r times faster than the one that is based on a SMDD. Furthermore, to reduce the memory access, we also propose PQMDD (Paged Quasi-reduced ordered Multi-valued Decision Diagram). In this case, we need not refer the indices, and can reduce the memory access into a half, which is also useful for faster implementation.

The rest of the paper is organized as follows: Section 2 surveys the realization methods for multiple-output logic functions, and presents PMDDs and PQMDDs. Section 3 proposes a realization of multiple-output logic functions by PQMDDs. Section 4 shows the experi-

mental results.

2. PQMDD and multiple-output logic function

From here, an SBDD (Shared reduced ordered Binary Decision Diagram) [10] is simply denoted by a BDD [3]. In a BDD, each non-terminal node corresponds to a variable, and the number 0 attached to an edge denotes $low(v)$, and the number 1 denotes $high(v)$. We only consider the decision diagrams where the orders of the input variables from the root nodes to the terminal nodes are the same in all paths.

To represent a multiple-output function, we can use an SMDD (Shared reduced ordered Multi-valued Decision Diagram). From here an SMDD is simply denoted by an MDD.

Definition 2.1 *An MDD(k) is the multiple-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 two-valued variables at a time, MDD(k) based evaluation is k times faster than BDD based one. An MDD(k) is easily derived from the corresponding BDD [9].

Example 2.1 *Fig. 1(a) shows the MDD(1) for an 8-input 2-output logic function.*

Partition the input variables $\{x_1, x_2, \ldots, x_8\}$ 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 $X_4 = (x_7, x_8)$, and we have the MDD(2) shown in Fig. 1(b).

In MDD(2), the length of the path from the root node to the terminal node is a half of that of the corresponding BDD, so the evaluation time in MDD(2) is also a half of the BDD. Similarly, Fig. 1(c) shows the MDD(3). Note that MDD(3) has a dummy variable x_9. ∎

In the examples, two-valued vectors are denoted by the corresponding decimal numbers. For example, $(0,0)$ is denoted by 0, $(0,1)$ is denoted by 1, $(1,0)$ is denoted by 2, and $(1,1)$ is denoted by 3.

Definition 2.2 *A QMDD (Quasi-reduced ordered Multi-valued Decision Diagrams) is the MDD(K) satisfying the following conditions:*

1. *For any two different nodes v_1, v_2, the subgraphs rooted in v_1 and v_2 are not equivalent.*
2. *Every path from the root node to a terminal node involves all the variables.*

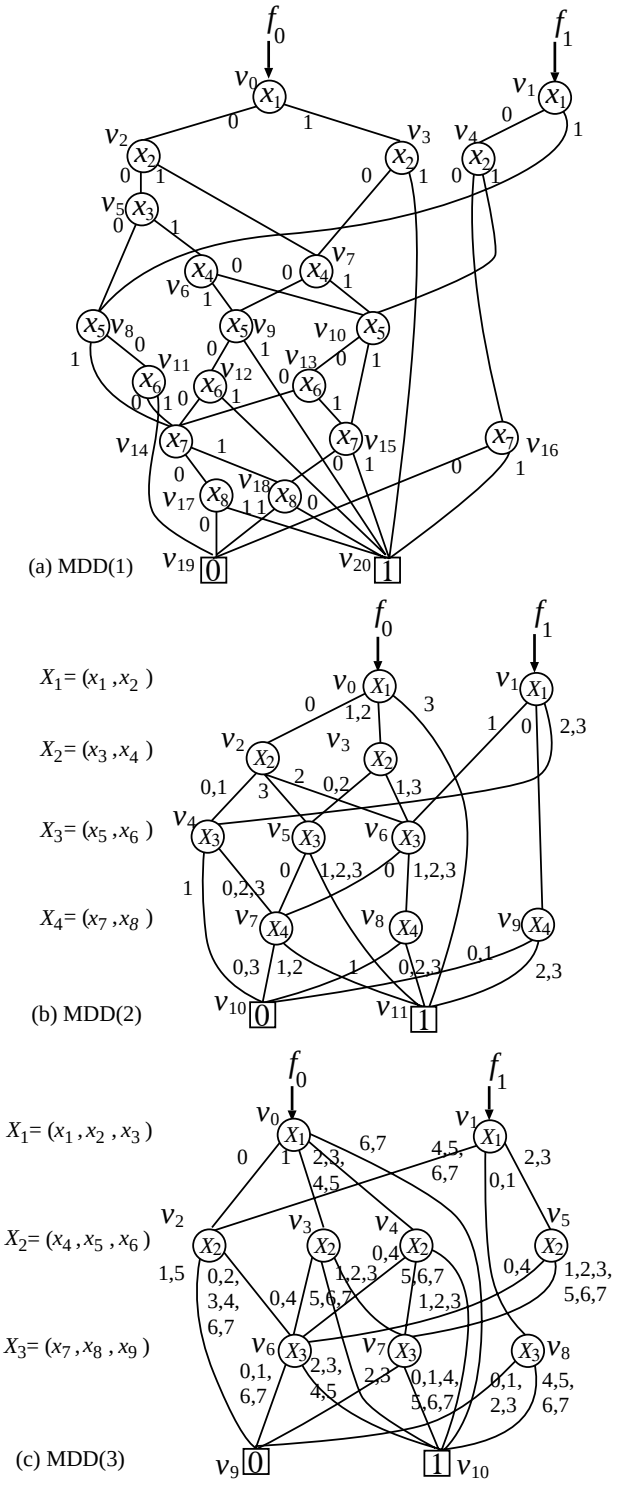

Figure 1. Example of MDD(k).

Definition 2.3 *The* **width** *of a QMDD is the maximum number of nodes that has the same index of variable.*

A QMDD(k) requires more nodes than MDD(k): It requires at most n times more nodes than MDD(k). However, in the evaluation of logic functions, QMDD(k) requires no index, since the index increases one by one.

Example 2.2 *Fig. 2(a) and (b) are the QMDD(k)s corresponding to the MDD(k)s in Fig. 1(b) and (c), respectively. Note that * denotes all the edges.* ∎

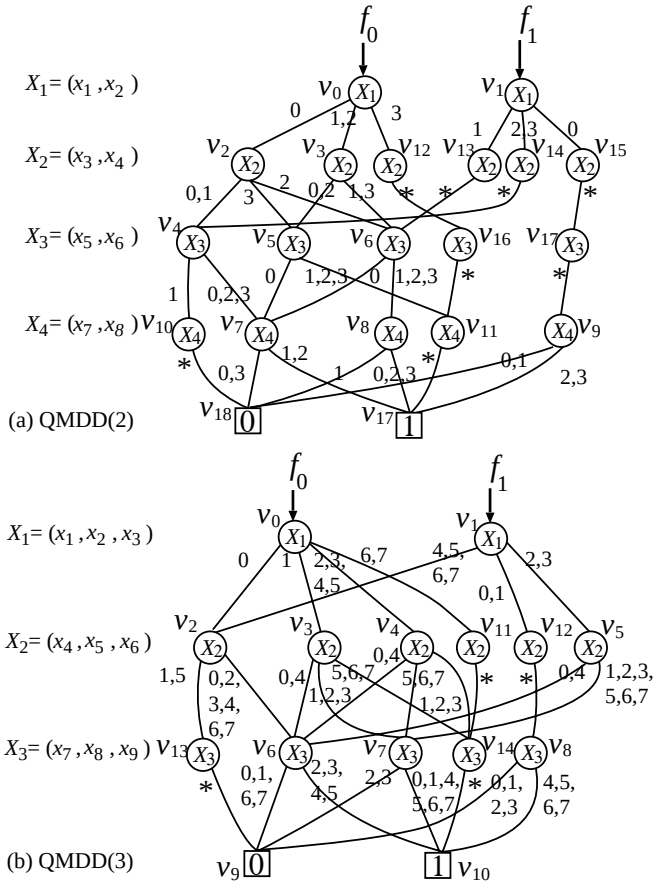

Figure 2. QMDD(k) corresponding to the MDD in Example 2.1.

Next, we will propose a PMDD (Paged reduced ordered Multi-valued Decision Diagram), which is the MDD(k) partitioned into r parts.

Definition 2.4 *A Paged reduced ordered MDD: PMDD(k, r) satisfies the followings:*
Let the input variables be $X = (X_1, X_2, \ldots, X_n)$, where $n = t \cdot r$. Consider the MDD, where in any path from the root node to a terminal node, there exists a non-terminal node in the $(t \cdot s+1)$th level $(s = 1, \ldots, r-1)$. The first page consists of levels 1 to t; the second page consists of levels $t+1$ to $2t$; and so on.

A PMDD has the following properties:

1. In a page, each node presents distinct function (two nodes in different pages may represent the same function).

2. An edge emerging from a node, is connected to other node in the same page, or a node in the first level of the next page.

Definition 2.5 *A Paged Quasi-reduced ordered Multi-valued Decision Diagrams: PQMDD(k, r) is a PMDD that has non-terminal nodes in every level.*

A PMDD(k, r) is an MDD(k) partitioned into r pages. Note that a PMDD$(1, 1)$ is a BDD, and a PMDD$(1, n)$ is a QROBDD (Quasi-reduced Ordered Binary Decision Diagram).

In a QROBDD, in every path from the root node to a constant, all the variables appear [12]. A PMDD(k, r) is an MDD(k) partitioned into r pages, and in the first level of each page, node exists in every path.

Definition 2.6 *Suppose that a function f is represented by a decision diagram (DD). The number of nodes in the DD is denoted by $size(DD, f)$.*

Theorem 2.1 $size(PMDD(1,1), f) \leq size(PMDD(1,r), f) \leq size(PMDD(1,n), f)$, $size(PMDD(k,r), f) \leq size(PQMDD(k,r), f)$, $size(QMDD(k), f) \leq n\ size(MDD(k), f)$, *where* $1 \leq r \leq n$.

When a QMDD(k, r) is partitioned into r pages, the subgraph from the $\{t(i-1)+1\}$th level to the $(t \cdot i)$th level forms the i-th page, we have the PQMDD(k, r). In this case, the structures of the graphs are the same except for the partition of the pages. So, the total numbers of the nodes in QMDD(k, r) and PQMDD(k, r) are the same. Thus, we have the following:

Theorem 2.2 $size(PQMDD(k,r), f)$ *does not depend on the value of* r $(1 \leq r \leq n)$.

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

1. It is a data structure suitable for pipelined implementation.

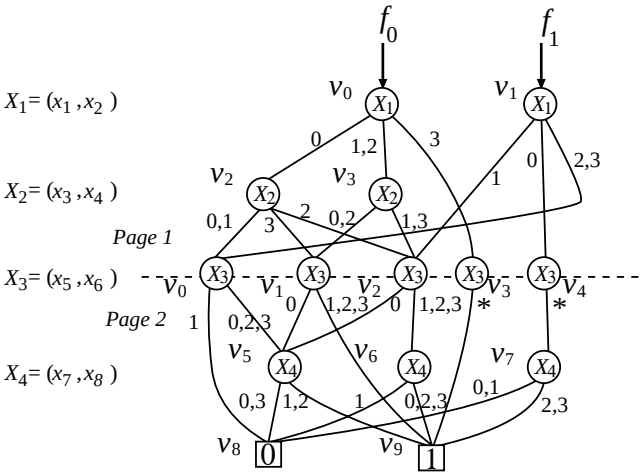

Figure 3. PMDD(2, 2) for the function in Example 2.1.

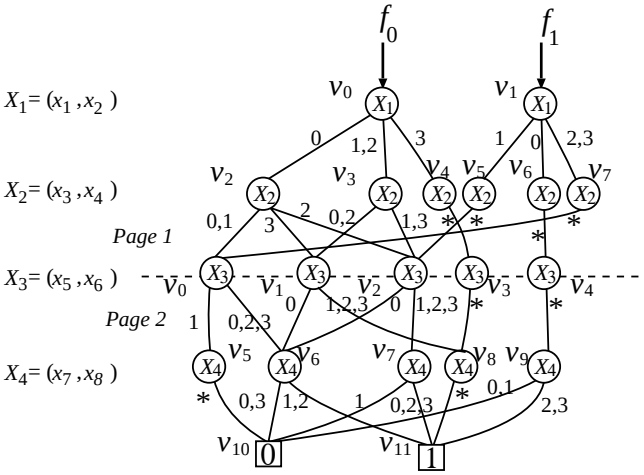

Figure 4. PQMDD(2, 2) for the function in Example 2.1.

2. The number of bits for the next address can be reduced, since the next address is limited to the same page or to the first level of the next page.

3. The index need not to be stored.

Example 2.3 *By partitioning the MDD(2) in Fig. 1(b) into two pages, we have the PMDD(2, 2) shown in Fig. 3. Similarly, by partitioning the QMDD(2) in Fig. 2(a) into two pages, we have the PQMDD(2, 2) shown in Fig. 4. In this case, variables x_1 to x_4 are assigned to the first page, and variables x_5 to x_8 are assigned to the second page. Note that several non-terminal nodes are attached on the boundary of the pages.* ∎

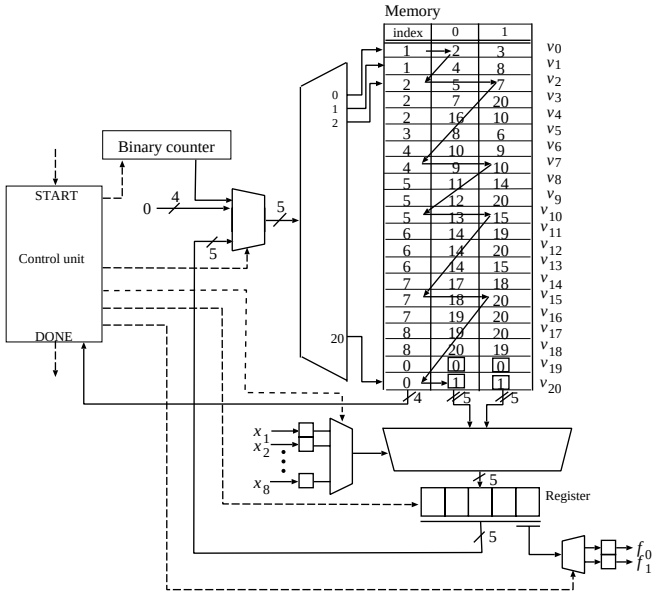

Figure 5. Sequential Network based on SBDD.

3. Sequential realization of a PQMDD

Sequential realization of multiple-output functions using SBDDs is well known [15, 5, 4].

3.1. The operation of sequential network

Fig. 5 shows a model of a sequential network. It consists of memory and control circuits. First, represent the given logic function by an MDD(k), and store the MDD data in the memory. By using sequential network, traverse the MDD(k) according to the input vectors, and obtain the function value.

Example 3.4 *Fig. 5 realizes the MDD(1), or SBDD in Fig. 1. The datum for the root nodes of f_0 and f_1 are stored in v_0 and v_1, respectively. Let us evaluate $f_0(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = f_0(0, 1, 1, 1, 1, 1, 1, 0)$ by traversing the memory from v_0 to v_{20}.*

First, to evaluate f_0, the binary counter is set to 0. This will be the starting address of the memory. The value of index in v_0 is 1, so read the value of x_1. Since the value is 0, which means the 0-edge, read the 0-edge. Since the value is 2, go to v_2. In a similar way, repeat the operation until the condition index $= 0$ is satisfied, which shows the terminal node. When the condition

that index = 0 is satisfied, read the value of either 0-edge or 1-edge, and evaluate $f_0(0,1,1,1,1,1,1,0) = 1$.

Next, to evaluate f_1*, the binary counter is set to 1, which is the starting address of* f_1. $f_1(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$ *are also evaluated in similar ways.* ∎

Traversing an edge requires two memory accesses: Once for the index part, and once for either 0-edge or 1-edge. For m-output function, the DD is traversed m times. Therefore, the n-input m-output logic function by using the network in Fig. 5 requires at most $2 \cdot n \cdot m$ memory access.

3.2. Speed up of the sequential network

This section shows two methods to speed up the evaluation. First, use MDD(k) to make the evaluation k times faster. Second, use QMDD to eliminate the memory access for index part: This makes the evaluation two times faster. Third, use r processing units to make the throughput r times larger.

3.2.1. MDD(k) and QMDD(k)

The datum for an MDD(k) are stored in the memory similar to the case of a BDD. The use of an MDD(k) will reduce the memory access into $1/k$, and make evaluation k times faster than BDD. An MDD(k) requires to read the index parts and to determine the input value to apply. However, if a QMDD is used instead of an MDD, input values can be applied without referring the indices. So, the index part is eliminated and memory access will be reduced into a half. Therefore, the evaluation will be two times faster. The demerit is that the non-terminal nodes increase up to n times.

Example 3.5 *Fig. 6 shows the sequential network based on QMDD(2) shown in Fig. 2. The function is evaluated as* $f_0(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) = f_0(0,1,1,1,1,1,1,0) = 1$ *by traversing memory from* v_0 *to* v_8 *along the arrows.*

The input values are $X_1 = (x_1, x_2) = (0,1) = 1$, $X_2 = (x_3, x_4) = (1,1) = 3$, $X_3 = (x_5, x_6) = (1,1) = 3$, *and* $X_4 = (x_7, x_8) = (1,0) = 2$.

Traverse the edge from v_0*, and in the fourth memory reference, we arrive at the second edge of* v_8*. Since the value is* 1*, we can evaluate* f_0 *as 1.*

In a similar way, f_1 *is evaluated by traversing the memory from* v_1*. For any input vector, the function is evaluated by four memory access. Note that the reference to the index is unnecessary.*

As shown in Fig. 2(b), the QMDD(3) has 13 non-terminal nodes. and each non-terminal node of a QMDD(k) requires 2^k *next addresses.* ∎

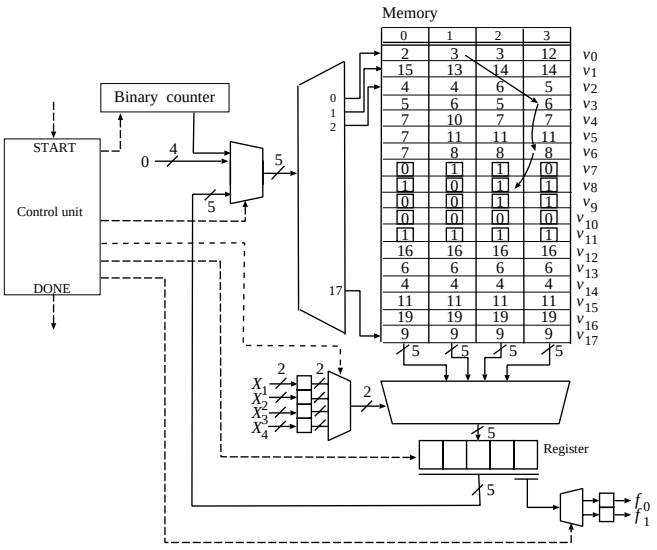

Figure 6. Sequential Network based on QMDD(2).

3.2.2. Sequential network based on PQMDD(k, r)

When the network has only one memory system, only one evaluation can be done at a time.

By preparing r memory systems for a PQMDD(k, r), and r copies of sequential networks that traverse the memory, and by pipelining these systems, we can increase the throughput.

Example 3.6 *Fig. 7 shows an example of sequential network for PQMDD(2,2) in Fig. 4.*
Time 1: $f_1(X_1, X_2, X_3, X_4) = f_1(1,3,3,2)$ *and* $f_2(X_1, X_2, X_3, X_4) = f_2(1,3,3,2)$. *First, in the first page, the input is* $(X_1, X_2) = (1,3)$*: From* v_0*, traverse the 1-edge, then go to* v_3 *to read the third entry. Then, start the search of the next page. The starting address of the second page is determined to be* v_2*, and the control is passed to the second unit.*
Time 2: *In the second unit, traverse from* v_2*. Since the input is* $(X_3, X_4) = (3,2)$*, read the third entry in* v_2*, and go to* v_7 *to read the second entry, and finally arrive at the terminal node* 1*, which is the output.*

At the same time, in the first page, the input is $(X_1, X_2) = (1,3)$*: From* v_1*, read the first entry, then go to* v_5 *to read third entry, and go to* v_3*, to find the staring address and pass the control to the second unit.*
Time 3: *In the second unit, start the traversal from* v_3*. For* $(X_3, X_4) = (3,2)$*. Read the third entry in* v_3*, and go to* v_9 *to read the second entry, and finally go to the terminal node* 1*, which is the output value.* ∎

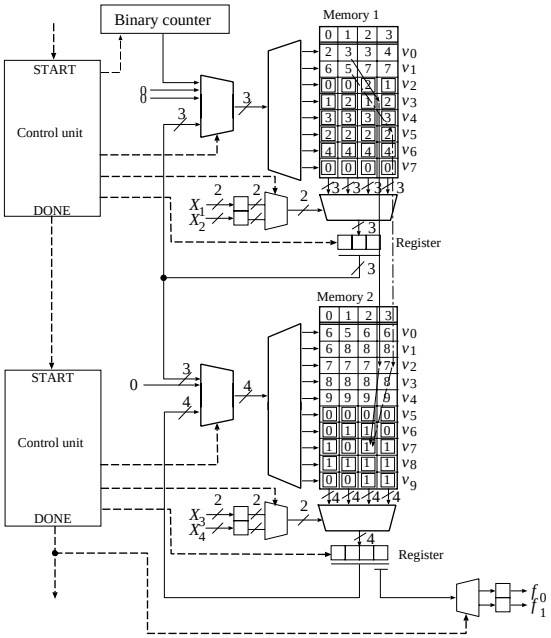

Figure 7. Sequential Network based on PQMDD(2, 2).

The above example shows the evaluation for one input vector. The example has two outputs. For m-output function where $m >> r$, if we ignore the overhead, the throughput will be r times larger than QMDD based one.

4. Experimental results

We developed a C program to generate PQMDD(k, r)s, and represented many functions.

Table 1 compares the number of nodes in PQMDD(k, r), (k = 1,2,3) for various benchmark circuits. For example, accpla has 50 inputs, 69 outputs, and the SBDD requires 1589 nodes. The PQMDD(1, 1) has 5876 nodes. In the PQMDD(1, 2), the first page requires 3904 nodes and the second page requires 1972 nodes, and the total number of nodes is 5876. Note that the total number of nodes in a PQMDD(k, r) does not depend on r for given k (Theorem 2.2). When k is increased from 1 to 3, the number of nodes decreases from 5876 to 1980. Since PQMDD(k, r) requires 2^k next addresses, the size of total memory is given by:

$$Mem = \lceil (\log_2 Nodes)/8 \rceil \times 8 \times 2^k \times Nodes[bits]$$

The total memory sizes are, 5876, 2961, and 1980, for $k = 1$, 2, and 3, respectively.

Function evaluation based on PQMDD(2, r) is two times faster than one based on PQMDD(1, r). However, the necessary memory size is almost the same for most benchmark functions. Thus, the value of the parameter k may be selected as $k = 2$.

For some functions, the value of k can be increased to three with moderate increase of memory.

5. Summary

In this paper, we have presented a method to implement multiple-output function by sequential networks. The conventional method is based on SBDD. The method based on a PQMDD(k,r) is $2 \times k \times r$ times faster than ones based on SBDD. However, the size of memory will be increased.

We can speed up the evaluation by pipelining, that is, we can activate many RAM blocks at the same time without increasing the clock frequency.

Acknowledgments

A part of this research is supported 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] B. Becker and R. Drechsler, "Efficient graph based representation of multivalued functions with an application to genetic algorithms," *Proc. of International Symposium on Multiple Valued Logic*, pp. 40-45, May 1994.

[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] C. C. Clare, *Designing Logic Systems Using State Machines*, McGraw-Hill, New York, 1973.

[5] M. Davio, J. -P. Deschamps, and A. Thayse, *DIGITAL SYSTEMS with algorithm implementation*, JOHN WILEY & SONS, 1983.

[6] Y. Iguchi, T. Sasao, and M. Matsuura, "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference, Proc. ASP-DAC'2000,* pp. 73–76, Jan. 2000.

Table 1. Number of nodes and necessary memory in PQMDDs.

			BDD		PQMDD					
					$k=1$		$k=2$		$k=3$	
Name	In	Out	Nodes	Mem	Nodes	Mem	Nodes	Mem	Nodes	Mem[Byte]
accpla	50	69	1589	63560	5876	188032	2961	189504	1980	253696
apex1	45	45	1275	51000	4290	137280	2137	136896	1426	182528
apex3	54	50	935	37400	5541	177312	2775	177600	1851	236928
apex5	117	88	1078	43120	10672	341504	5402	345856	3553	454784
apex6	135	99	611	24440	10680	341760	5371	343872	3584	458752
apex7	49	37	262	10480	1770	56640	902	57856	600	77056
b3	32	20	359	14360	991	31712	500	32000	341	43904
bca	26	46	847	33880	1752	56064	888	56832	593	76160
bcb	26	39	716	28640	1117	35744	561	35904	392	50432
bcc	26	45	736	29440	919	29408	469	30016	323	41600
bcd	26	38	846	33840	1155	36960	582	37248	398	51200
cps	24	109	987	39480	2346	75072	1213	77632	813	104064
des	256	245	3726	149040	62417	1997344	31194	1996416	20909	2676608
ex4	128	28	522	20880	4458	142656	2234	142976	1502	192512
exep	30	63	601	24040	1547	49504	780	49920	535	68480
frg2	143	139	1400	56000	19564	626048	9812	628096	6511	833664
i5	133	66	133	3192	4803	153696	2416	154752	1625	208256
i6	138	67	209	5016	9684	309888	4844	310016	3225	412800
i7	199	67	334	13360	15870	507840	7953	509120	5291	677504
i8	133	81	2034	81360	24042	769344	12002	768256	7973	1020800
i9	88	63	922	36880	11711	374752	5876	376064	3929	503168
ibm	48	17	206	4944	1022	32704	505	32320	338	43264
jbp	36	57	413	16520	2021	64672	1015	64960	697	89216
k2	45	45	1275	51000	4290	137280	2137	136896	1426	182528
pdc	16	40	560	22400	1040	33280	548	35072	346	44544
seq	41	35	1248	49920	3614	115648	1784	114304	1259	161408
soar	83	94	535	21400	5802	185664	2917	186816	1946	249344
ti	47	72	665	26600	2965	94880	1507	96576	1013	129920
xparc	41	73	1875	75000	5059	161888	2530	162048	1735	222336

k: The number of variables in a group.
Nodes: The number of nodes.
Mem: Total memory size. [bits]

[7] T. Kam, T. Villa, R. K. Brayton, and A. L. Sagiovanni-Vincentelli, "Multi-valued decision diagrams: Theory and Applications," *Multiple-Valued Logic*, 1988, Vol. 4, No. 1-2, pp. 9–62, 1998.

[8] C. Lee, "Representation of switching circuits by binary decision programs," *Bell System T. J.*, 38, pp. 989–999, July 1959.

[9] 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.

[10] S. Minato, N. Ishiura, and S. Yajima, "Shared binary decision diagram with attributed edges for efficient Boolean function manipulation," *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 52–57, June 1990.

[11] D. M Miller, "Multiple-valued logic design tools," *Proc. of International Symposium on Multiple Valued Logic*, pp. 2–11, May 1993.

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

[13] 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.

[14] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

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