A Cascade Realization of Multiple-Output Function for Reconfigurable Hardware

Tsutomu SASAO[1,2], Munehiro MATSUURA[1], and Yukihiro IGUCHI[3]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology
[3]Department of Computer Science, Meiji University

May 1, 2001

Abstract

A realization of multiple-output logic functions using a RAM and a sequencer is presented. First, a multiple-output function is represented by an encoded characteristic function for non-zeros (ECFN). Then, it is represented by a cascade of look-up tables (LUTs). And finally, the cascade is simulated by a RAM and a sequencer. Multiple-output functions for benchmark functions are realized by cascades of LUTs, and the number of LUTs and levels of cascades are shown. A partition method of outputs for parallel evaluation is also presented. A prototype has been developed by using RAM and FPGA.

Key words: Reconfigurable hardware, Multiple-output logic function, Functional decomposition, Cascade realization, FPGA synthesis, Time domain multiplexing (TDM)

1 Introduction

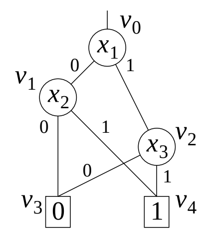
Two of the most crucial problems in system LSIs are their long design time and short life cycles. A solution to these problems may be reconfigurable architecture. Reconfigurable LSIs will reduce the hardware development time drastically, since one LSI can be used for various applications.

In this paper, we consider a realization of combinational logic functions by reconfigurable architecture. Various methods exist to realize multiple-output logic functions by reconfigurable architecture. Among them, random access memories (RAMs) and programmable logic arrays (PLAs) directly implement logic functions. However, when the number of input variables n is large, the necessary hardware becomes too large. Thus, field programmable logic arrays (FPGAs) are often used. Unfortunately, FPGAs require layout and routing in addition to logic design. Also, the area for programming and interconnections are much larger than the logic area. Thus, FPGAs require large chip area.

When speed of the operation is not so important, a general-purpose microprocessor can be used to implement logic functions. However, the microprocessor implementation is often 100 to 1000 times slower than the direct circuit realizations. Also, the power dissipation is rather high.

Here, we assume the following applications:

- The system need not be so fast as custom logic circuits, but must be faster than the software realization.



(a)

v_0: if($x_1 == 0$) goto v_1;
else goto v_2;
v_1: if($x_2 == 0$) goto v_3;
else goto v_4;
v_2: if($x_3 == 0$) goto v_3;
else goto v_4;
v_3: return(0);
v_4: return(1);

(b)

Figure 1.1: Branching program method.

- The system is too large to implement by a PLA or a RAM directly.
- System must be reconfigurable.

In this paper, we consider a method to implement logic functions by using sequential network, where the speed must be faster than the conventional software realization. First, to make the problem simple, let us consider the realization of single-output logic function of n variables.

The branching program method realizes a logic function by a sequential network as follows [1]:

1) Represent the given logic function by a binary decision diagram (BDD) [11, 4] (Fig. 1.1(a)).
2) Replace each non-terminal node of the BDD with an ”If then else” statement, and derive the branching program to represent f (Fig. 1.1(b)).
3) Implement the program by a general-purpose microprocessor.

Note that the branching program method requires $O(n)$ computation time. To reduce the instruction fetch time, special sequential machines that traverse the BDD structure are proposed [20, 6]. In this case, the necessary memory is proportional to the number of nodes in the BDD.

If k variables are evaluated at the same time, then the evaluation speed will be k times faster than the branching program method. This corresponds to using a multiple-valued decision diagram (MDD) instead of a BDD [12].

If we partition the BDD into several pages, and operate each page in parallel, then we have the pipelined architecture [7], which is several times faster than the naive realization.

Table 1.1: Comparison of Reconfigurable Realizations for Multiple-output Functions.

	Performance	Design time	Chip area
FPGA method	High	Long	Large
LUT cascade method	Medium	Medium	Medium
Branching program method	Low	Short	Small

In this paper, we propose *the LUT cascade method*, which uses lookup tables (LUTs) as basic logic elements. In this method, a cascade of LUTs is used to implement logic functions, which makes the sequencer simple enough to be implemented by a reconfigurable network. Also, with more memory, we can design a faster system.

In the branching program method, the number of memory references is proportional to the number of input variables. On the other hand, in the LUT cascade method, the number of memory references is equal to the number of levels of the cascade. Experimental results show that the number of levels of the cascade is about one tenth of the numbers of the input variables. Thus, the we can expect that the LUT cascade method will be about ten times faster than the branching program method.

As for the amount of memory, branching program requires memory that is proportional to the number of nodes in the BDD. On the other hand, the LUT cascade method requires more memory than the branching program method.

Table 1.1 compares these methods.

2 Functional Decomposition using BDDs

Definition 2.1 *Let $X = (x_1, x_2, \ldots, x_n)$ be input variables. A set of variables X is denoted by $\{X\}$. $X = (X_1, X_2)$ is a partition of X if $\{X_1\} \cup \{X_2\} = \{X\}$ and $\{X_1\} \cap \{X_2\} = \phi$. The number of variables in X is denoted by $|X|$.*

Definition 2.2 *For a logic function $f(X)$, let $X = (X_1, X_2)$ be a partition of X. The* **decomposition chart** *of f, denoted by $M(f : X_1, X_2)$, is the matrix having 2^{n_1} columns and 2^{n_2} rows. In $M(f : X_1, X_2)$, each row and column has label with binary number, and the corresponding element denotes the truth value of f, where $n_1 = |X_1|$ and $n_2 = |X_2|$. The columns and rows have all possible patterns of n_1 bits and n_2 bits, respectively.*

Example 2.1 *Let $f(X)$ be a 4-variable function, and $X = (X_1, X_2)$ be a partition of X, where $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4)$. Table 2.1 is an example of a decomposition chart. (End of Example)*

Definition 2.3 *The number of different column patterns in a decomposition chart is the* **column multiplicity***, and is denoted by μ.*

The column multiplicity of a decomposition chart depends on the partition $X = (X_1, X_2)$ of the input variables.

Lemma 2.1 *[2, 5] Let the partition of X be (X_1, X_2). If the column multiplicity of the decomposition chart $M(f : X_1, X_2)$ for a function f is μ, then f can be represented as $f(X) = g(h_1(X_1), h_2(X_1), \ldots, h_u(X_1), X_2)$, and f can be realizable with the network structure shown in Fig. 2.1, where $u = \lceil \log_2 \mu \rceil$. X_1 is the* **bound set***.*

Table 2.1: Decomposition chart.

$X_2 = (x_3, x_4)$ \ $X_1 = (x_1, x_2)$	0 0	0 1	1 0	1 1
0 0	0	1	1	0
0 1	1	1	1	1
1 0	0	1	1	0
1 1	0	0	0	0

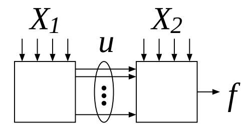


Figure 2.1: Functional Decomposition.

Lemma 2.2 *[10, 15] Let (X_1, X_2) be a partition of X, and let the BDD of the function f be partitioned as shown in Fig. 2.2. Suppose that k nodes in the lower block are adjacent to the upper block. Also, let the column multiplicity of the decomposition chart $M(f : X_1, X_2)$ for the function f be μ. Then, $\mu = k$.*

Definition 2.4 *[13] The* **width of the BDD at level** k *is the number of edges crossing the section of the graph between x_k and x_{k+1}, where the edges pointing to the same nodes are counted as one. The* **width** *of the BDD is the maximum width of the BDD among the levels.*

Theorem 2.1 *Consider a BDD for an n-variable logic function f. Let the width of the BDD be μ_{max}. If $u = \lceil \log_2 \mu_{max} \rceil \leq k - 1$, then f can be realized by a cascade of k-LUTs shown in Fig. 2.3. Let s be the numbers of levels of the cascade, and N be the number of LUTs, then we have*

$$\lceil \frac{n+u-2}{k-1} \rceil \leq \quad s \quad \leq 1 + \lceil \frac{n-u-1}{k-u} \rceil$$

$$\lceil \frac{n+u-2}{k-1} \rceil + u - 1 \leq \quad N \quad \leq \lceil \frac{n-u-1}{k-u} \rceil u + 1$$

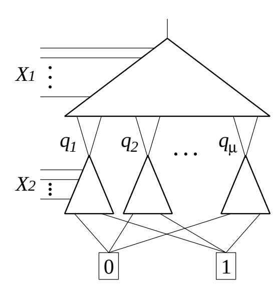


Figure 2.2: Functional decomposition using BDD.

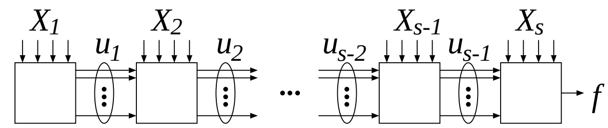


Figure 2.3: The network obtained by applying functional decompositions $s-1$ times.

Theorem 2.2 *Let s be the number of levels of the cascade in Fig. 2.3. Then, we have*

$$\lceil \frac{n-\hat{u}}{k-\hat{u}} \rceil \leq s \leq 1 + \lceil \frac{n-\hat{u}-1}{k-\hat{u}} \rceil, \text{ where } \hat{u} = \frac{1}{s-1}\sum_{i=1}^{s-1} u_i.$$

$u_i = \lceil \log_2 \mu_i \rceil$ and μ_i is the column multiplicity for the decomposition of f, where $X_1 \cup X_2 \cup \cdots \cup X_i$ is the bound set.

Theorem 2.2 gives tighter bounds on s than Theorem 2.1. Since $\hat{u}$ is hard to obtain, we approximate it by the average value of the logarithm of the widths of all the levels in the BDD. From these relations, we can easily estimate the number of LUTs and the level of the cascade.

3 Representation of Multiple-output Function ECFN

Although the method described in the previous section is useful for a single-output function, it is hard to apply to multiple-output functions. In the case of an m-output function, the number of terminal nodes of the MTBDD [16] can be as much as 2^m, which may be too large to construct. Also, the representations using characteristic function (CF) of multiple-output function have been developed [1]. However, in many cases, BDDs for CFs are too large to construct. From this, we use the following method to represent a multiple-output function.

Definition 3.1 *[19] Let m functions be f_j $(j = 0, 1, \ldots, m-1)$. The encoded characteristic function for non-zero outputs (ECFN) is*

$$ECFN = \bigvee_{j=0}^{w-1} z_{w-1}^{b_{w-1}} z_{w-2}^{b_{w-2}} \cdots z_0^{b_0} f_j,$$

where $\vec{b} = (b_{w-1}, b_{w-2}, \ldots, b_0)$ is the binary representation of the integer j, and $w = \lceil \log_2 m \rceil$.

Note that $z_0, z_1, \ldots, z_{w-1}$ are auxiliary variables that represent the outputs.

Example 3.1 *Consider the case of $m = 8$. Let z_0, z_1, and z_2 be the auxiliary variables that represent output groups. In this case, the ECFN of the 8-output function $(f_0, f_1, \ldots, f_7)$ is represented by*

$$\begin{aligned} ECFN = {} & \bar{z}_2\bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_2\bar{z}_1 z_0 f_1 \vee \bar{z}_2 z_1 \bar{z}_0 f_2 \vee \bar{z}_2 z_1 z_0 f_3 \vee \\ & z_2\bar{z}_1\bar{z}_0 f_4 \vee z_2\bar{z}_1 z_0 f_5 \vee z_2 z_1 \bar{z}_0 f_6 \vee z_2 z_1 z_0 f_7 \end{aligned}$$

(End of Example)

When constructing a BDD for an ECFN, we can reduce the width of the BDD by mixing the auxiliary variables and ordinary input variables. As will be shown in the experimental results, the widths of the BDDs obtained in this way are, in most cases, smaller than those of SBDDs, BDDs for CFs, and MTBDDs for the corresponding functions. When all the auxiliary variables are adjacent to the root node, the BDD is equivalent to an ordinary SBDD. On the other hand, when all the auxiliary variables are adjacent to the terminal nodes, the BDD is equivalent to the MTBDD. We can also reduce the sizes of BDDs by considering the encoding methods [19].

4 Level Reduction by Output Partition

To evaluate an m-output function by using the network for the ECFN, we have to iterate logic evaluation m times by changing the values of the auxiliary variables. Thus, when m is large, the evaluation time tends to be long. To solve the this difficulty, we use a parallel process to make it faster:

To represent a multiple-output logic function $\mathcal{F} = \{f_0, f_1, \ldots, f_{m-1}\}$, partition the output set $\mathcal{F}$ into $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_r$, where $\mathcal{F}_1 \cup \mathcal{F}_2 \cup \cdots \cup \mathcal{F}_r = \mathcal{F}$, and $\mathcal{F}_i \cap \mathcal{F}_j = \phi$ $(i \neq j)$. And, we can reduce the levels of the cascade. Let μ_i be the width of the BDD representing the ECFN for $\mathcal{F}_i$. Clearly, $\mu_i \leq \mu$. Partitioning the outputs $\mathcal{F}$ will often reduce the number of input variables and μ_i. Thus, by Theorem 2.2, n and $\hat{u}$ are also decreased. So, in many cases, the levels of the network are also reduced.

We partition the output set into $\mathcal{F}_1, \mathcal{F}_2, \ldots, \mathcal{F}_r$, so that each group has nearly the same number of elements. If we evaluate them in parallel, then the evaluation speed-up will be r times. Furthermore, in many cases, since levels of the network will be decreased, the evaluation speed will be more than r times. The output partition can be done as follows:

Algorithm 4.1 *(Partitioning of the outputs)*

1. *Let Th be the threshold on the number of levels of LUTs and let $l = 1$.*
2. *Among the BDDs for the individual output function, find f_i that requires the maximum number of levels.*
3. *Let $\mathcal{F}_l \leftarrow f_i$.*
4. *Let f_j be an output function that is not selected. While the number of levels does not exceeds Th, let $\mathcal{F}_l \leftarrow \mathcal{F}_l \cup \{f_j\}$.*
5. *If there exist an unselected output, then let $l \leftarrow l+1$ and go to 2. If all the output functions are selected, then stop.*

5 Architecture for Reconfigurable Hardware

The cascade of LUTs shown in Fig. 2.3 can be simulated by the architecture shown in Fig. 5.1. In this architecture, the memory stores the data for LUTs, while the control part (a sequencer) stores the information of the interconnections among LUTs. Since the network structure is very simple, the control part is also simple. We can make the operation

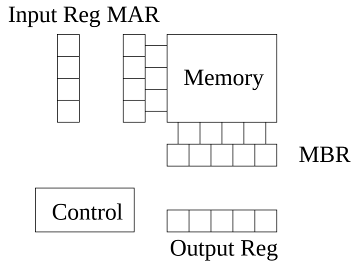


Figure 5.1: Architecture for Logic Simulator.

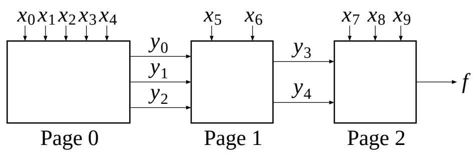


Figure 5.2: Cascade.

fast by using a special hardware tailored to the given logic function.

Example 5.1 *Given a 10-variable function $f(x_0, x_1, \ldots, x_9)$, and the cascade of 5-input LUTs shown in Fig. 5.2, realize f by the memory shown in Fig. 5.3 and a sequencer. Fig. 5.4 shows the map of the LUT data in the memory, where the hatched part shows the unused area.*

Algorithm 5.1 *(Function Evaluation using a Memory and a Sequencer)*

1. *Let* $(A_0, A_1, A_2, A_3, A_4, A_5, A_6) \leftarrow (0, 0, x_0, x_1, x_2, x_3, x_4)$.
2. *Read* (D_0, D_1, D_2, D_3)*, and let* $(y_0, y_1, y_2) \leftarrow (D_1, D_2, D_3)$.
3. *Let* $(A_0, A_1, A_2, A_3, A_4, A_5, A_6) \leftarrow (0, 1, y_0, y_1, y_2, x_5, x_6)$.
4. *Read* (D_0, D_1, D_2, D_3)*, and let* $(y_3, y_4) \leftarrow (D_2, D_3)$.
5. *Let* $(A_0, A_1, A_2, A_3, A_4, A_5, A_6) \leftarrow (1, 0, y_3, y_4, x_7, x_8, x_9)$.
6. *Read* (D_0, D_1, D_2, D_3)*, and let* $(f) \leftarrow (D_3)$.

In this way, we can evaluate f by accessing the memory three times. *(End of Example)*

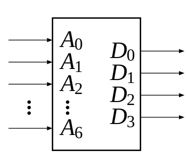


Figure 5.3: Memory for Logic Simulation.

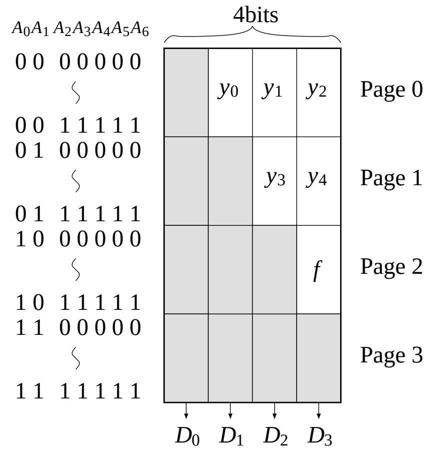


Figure 5.4: Mapping of LUT Data into the Memory.

Memory-Packing

In Example 5.1, only the data for the LUTs in the same stage are stored in a page. By this restriction, more than half of the memory area is unused in Fig. 5.4. By embedding the LUT data for the final stage of the cascade into the D_0 area in Page 0, we can save memory. In general, the LUT data of the same stage of the cascade must be read at the same time, and so must be stored in the same page. However, if there is unused area in the same page, we can store the LUT data for the different stage. This is called *memory-packing*. By memory packing, the necessary memory can be reduced up to [number of LUTs$\times 2^k$ bit]. The memory-packing can be done as follow:

Algorithm 5.2 *(Memory-packing)*

1. *Let u_i be the number of outputs in the i-th stage $(i = 1, 2, \ldots, s)$ of the cascade. Note that $u_s = 1$, since in the final stage of the cascade, the number of outputs is 1.*
2. *Prepare memory having $u = \max_i\{u_i\}$-bit outputs.*
3. *Sort $u_1, u_2, \ldots, u_s$ in increasing order, and let them to be $v_1, v_2, \ldots, v_s$.*
4. *Let $i \leftarrow 0$.*
5. *Let $i \leftarrow i + 1$ and $k \leftarrow 0$.*
6. *Check if v_i outputs can be assigned in the k-th page. If possible, assign it, and return to 5, otherwise let $k \leftarrow k + 1$, and go to 6.*
7. *If $i = s$ is assigned, stop.*

Theorem 5.1 *Suppose that an n-variable logic function is realized by the cascade of k-LUTs shown in Fig. 2.3. Let $L(bits)$ be the size of memory available, and let μ_{max} be the width of the BDD. Then, we have*

$$
\begin{aligned}
&k \geq u + 1, \\
&u = \lceil \log_2 \mu_{max} \rceil, \\
&2^k \left(\frac{n + u - 2}{k - 1} + u - 1\right) \leq L
\end{aligned}
$$

Example 5.2 *The benchmark function C499 has $n = 41$ inputs. Let us realize it on a memory with $L = 2^{20}$(1 Mega) bits. Since $\mu_{max} = 2048$, we have $u = \lceil \log_2 \mu_{max} \rceil = 11$. From Theorem 5.1, it is sufficient to consider k with values for $12 \leq k \leq 16$.* *(End of Example)*

6 Experimental Results

6.1 Realization of Cascades

Table 6.1 shows the results for the cascade realization of benchmark functions. This table shows that the size of BDDs for ECFNs are, in most cases, smaller than corresponding MTBDDs and BDDs for CFs. Blank entries show that the BDDs were too large to construct.

We optimized the BDD for ECFN by mixing the input variables and auxiliary variables. We find the ordering of the variables by using a heuristics that reduces the total number of nodes in the QROBDD [16]. Note that this heuristic will reduce $\hat{u}$ in Theorem 2.2. For some functions, the number of nodes in the BDDs increases, in spite of the reduction of $\hat{u}$. In the table, μ_{max1} denotes the width of the shared BDDs (SBDDs), and μ_{max2} denotes the width of the BDD for the ECFN. In most cases, our heuristics reduced the widths of BDDs: Exceptions are C3540 and rot. s_1 shows the lower and upper bounds on the number of levels obtained from Theorem 2.1. Similarly, s_2 shows the lower and upper bounds on the number of levels obtained from Theorem 2.2. We can see that s_2 is tighter than s_1. Also, s denotes the number of levels in a cascade, and N denotes the number of LUTs. In this experiment, encodings of outputs [19] are not optimized. Currently, we do not have a good algorithm that works for large BDDs.

Murgai-Hirose-Fujita [14] have developed a logic simulation system which realizes given function by using k-LUT $(k = 15)$. In their paper [14], no level of the networks are shown. So, we did similar experiment by using MIS-FPGA, and obtained N, the number of LUTs, and s, the number of levels. In this experiment, we used the following script:

```
> xl_imp -n 2
> xl_partition -n 15
> simplify
> xl_partition -n 15
```

The results are shown in the last two columns of Table 6.1. In most cases, MIS-FPGA produced networks with more LUTs, but fewer levels.

Since the evaluation time of the Algorithm 5.1 is proportional to $s \cdot m$, when m is large we have to reduce the number of levels by partitioning the output set. Table 6.2 compares the numbers of LUTs and levels of cascades when the outputs are partitioned into four and eight groups. By partitioning the outputs into four groups, the number of levels can be reduced to half. In this case, the parallel evaluation is more than eight times faster than the original one.

6.2 Prototype of Reconfigurable Hardware

In order to investigate the performance of the architecture shown in Section 5, we developed reconfigurable hardware using a commercially available FPGA board.

Table 6.2: Results of Output Partition.

Name	Without partition		4 partition		8 partition	
	s	LUTs	s	LUTs	s	LUTs
C2670	28	170	12	178	11	183
C5315	23	142	13	257	10	295
C7552	25	156	17	216	17	203
des	34	235	15	319	9	293
rot	18	125	9	179	7	203

The specification of the FPGA board is as follows:

- FPGA: Altera EPF10K200S
- Clock frequency :33MHz
- RAM: Static 8MBytes
- Interface: PCI

In this prototype, memory-packing is not implemented.

We did experiments for a large number of benchmark functions. When m, the number of outputs is large, the number of levels s tends to be large. So, we partition the output set into groups to make the simulator faster.

Murgai-Hirose-Fujita [14] use an event-driven method, so the evaluation time is at least proportional to the number of LUTs.

7 Conclusions

In this paper, we have shown a method to represent a multiple-output logic function by a cascade of k-LUTs. We also developed a reconfigurable hardware consisting of a memory and a sequencer.

The features of the method include:

1. The system uses a cascade of LUTs: The hardware is simple to implement. The design consists of iterative decompositions of BDDs for ECFNs.
2. The system uses multiple-output LUTs: It is faster than Murgai-Hirose-Fujita's simulator.
3. The system users BDDs for ECFNs, which are smaller than the corresponding SBDDs: The input variables and the auxiliary variables are mixed to reduce the BDDs.
4. Given the size of memory, we can find the best value of k to optimize the hardware.
5. By partitioning the outputs into r groups, the hardware becomes at least r times faster.

To the architecture shown in Fig. 5.1, we can map the network in various ways. For example, when the size of the network is small, we can map the multiple-output function directly without using ECFN.

In this paper, we only considered the case where the values of k are the same for all the stages of a cascade. However, in general, the value of k can be different for different stages. By using this technique, we can implement larger function on a smaller memory.

Table 6.1: Experimental Results $(k = 15)$.

Name	In	Out	MTBDD	BDD for CF	SBDD	BDD for ECFN	μ_{max1}	μ_{max2}	Lower and Upper bounds s_1		Lower and Upper bounds s_2		This method s	This method N	MIS-FPGA s	MIS-FPGA N
C432	36	7	1198	885	1075	859	101	83	4	5	4	5	4	20	9	22
C499	41	32			27876	24944	2176	2048	4	10	6	7	7	60	4	104
C880	60	26			4166	4567	466	464	6	11	8	9	8	51	9	66
C1908	33	25			7456	7548	620	620	4	7	5	6	5	35	7	104
C2670	233	140			2847	3688	411	284	18	40	28	29	28	170	5	186
C3540	50	22	9564117		34710	39320	5420	5482	5	22	11	11	11	107	13	154
C5315	178	123			2564	3256	258	238	14	27	22	23	23	142	7	188
C7552	207	108			2945	3648	193	160	16	31	25	26	25	156	7	254
apex3	54	50	537	1786	986	1207	204	165	5	9	6	7	6	28	4	86
apex7	49	37			300	437	55	49	5	7	5	6	5	21	2	38
b9	41	21	32720	1582	177	219	42	40	4	6	4	5	4	14	2	22
dalu	75	16	522749	9042	1178	1218	244	149	7	11	8	9	8	40	12	119
des	256	245			3975	3740	608	285	20	44	34	35	34	235	2	309
duke2	22	29	638	755	366	452	78	48	3	4	3	4	3	10	3	38
e64	65	65	131	2277	194	675	66	36	6	9	7	8	7	25	5	69
ex4*	128	28	4722244		540	602	83	38	7	11	8	9	8	32	2	33
k2	45	45	913	2860	1321	1640	251	245	5	7	5	6	6	28	6	161
rot	135	107			8501	9658	1196	1204	11	34	18	19	18	125	7	141
spla	16	46	11100	2121	628	626	101	69	2	3	2	3	2	8	6	144

μ_{max1}: The width of SBDD. μ_{max2}: The width of the BDD for the ECFN.

s_1: Lower and upper bounds on the number of levels obtained by Theorem 2.1.

s_2: Lower and upper bounds on the number of levels obtained by Theorem 2.2.

s: Number of levels. N: Number of LUTs.

*: Contains redundant variables. We used the number of dependent variables to obtain the bounds.

Acknowledgments

This research is partly supported by the grant in Aid for Scientific Research of the Ministry of Education, Culture, Sports, Science and Technology of Japan. Dr. R. Murgai of Fujitsu Laboratory of America showed us the method to use MIS-FPGA. Prof. Jon T. Butler's comments improved the English presentation. Prof. Qingjian Yu's comments improved Theorems 2.1 and 2.2.

References

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

[2] R. L. Ashenhurst, "The decomposition of switching functions," *In Proceedings of an International Symposium on the Theory of Switching*, pp. 74-116, April 1957.

[3] J. Babb, R. Tessier, M. Dahl, S. Hanono, D. Hoki, and A. Agarwal, "Logic emulation with virtual wires," *IEEE Transactions on Computer Aided Design*, Vol. 16, No. 6, pp. 609-626, June 1997.

[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE TC*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[5] H. A. Curtis, *A New Approach to The Design of Switching Circuits*, D. Van Nostrand Co., Princeton, NJ, 1962.

[6] M. Davio, J-P Deschamps, and A. Thayse, *Digital Systems with Algorithm Implementation*, John Wiley and Sons, New York, 1983.

[7] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference (ASP-DAC'2000)*, Jan. 26-28, Yokohama, Japan.

[8] Y. Iguchi, T. Sasao, M. Matsuura, "Implementation of multiple-output functions using PQMDDs," *IEEE International Symposium on Multiple-Valued Logic*, pp.199-205, May 2000.

[9] J.-H. R. Jian, J.-Y. Jou, and J.-D. Huang, "Compatible class encoding in hyper-function decomposition for FPGA synthesis," *Design Automation Conference*, pp. 712-717, June 1998.

[10] Y-T. Lai, M. Pedram, and S. B. K. Vrudhula, "EVBDD-based algorithm for integer linear programming, spectral transformation, and functional decomposition," *IEEE Trans. CAD*, Vol. 13, No. 8, pp. 959-975, Aug. 1994.

[11] C. Lee, "Representation of switching circuits by binary-decision programs," *Bell System Technical Journal*, Vol. 19, pp. 985-999, July 1959.

[12] 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.

[13] S. Minato, "Minimum-width method of variable ordering for binary decision diagrams," *IEICE Trans. Fundamentals*, Vol. E75-A, No. 3, pp. 392-399, March 1992.

[14] R. Murgai, F. Hirose, and M. Fujita, "Logic synthesis for a single large look-up table," *Proc. International Conference on Computer Design*, pp. 415-424, Oct. 1995.

[15] T. Sasao, "FPGA design by generalized functional decomposition," *(Sasao ed.) Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

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

[17] 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.

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

[19] T. Sasao, "Compact SOP representations for multiple-output functions: An encoding method using multiple-valued logic," *International Symposium on Multiple-Valued Logic*, Warsaw, Poland, 2001 (to be published).

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