A Design Algorithm for Sequential Circuits using LUT Rings

Hiroki Nakahara† Tsutomu Sasao‡ Munehiro Matsuura‡

Department of Computer Science and Electronics
Kyushu Institute of Technology, Iizuka 820-8502, Japan
†nakahara@aries02.cse.kyutech.ac.jp, ‡{sasao,matsuura}@cse.kyutech.ac.jp

This paper shows a design method for a sequential circuit by using a Look-Up Table (LUT) ring. An LUT ring consists of memories, a programmable interconnection network, a feed-back register, an output register, and a control circuit. It sequentially emulates an LUT cascade that represents the state transition functions and the output functions. We present two algorithms for synthesizing a sequential circuit by an LUT ring: The first one partitions the outputs into groups, and realize them by LUT cascades. The second one reduces the evaluation time by using unused memories. We also compare the LUT ring with other methods to realize sequential circuits.

I. Introduction

The degree of integration of LSIs is constantly increasing with the Moore's law: Now we can integrate a hundred millions transistors into a chip. As a result, the design of LSIs require long time. In addition, the LSI design have deep-submicron (DSM) effects such as cross-talk noise and inductive effects that require electro-magnetic design.

To solve these problems, regular and reconfigurable architectures have been considered. Regular architectures have repeated structures, hence the overall structure at the global level is uniform. Such a structure is more predictable in its delays. A repeated pattern can be hand-designed and extensively analyzed to avoid internal DSM problems, since its scale is relatively small and needs to be designed only once [1]. Reconfigurable architectures is rewritable, and can reduce the hardware development time drastically.

Memory is the most important device that is regular and configurable. Several methods exist to implement a sequential circuit by using memory. They include:

1. **Direct Method**
   This method directly implements the combinational part by a memory. It is simple, but to implement an n-input m-output function, we need a memory with $m2^n$bits, which is impractical when n is large.
2. **Memory and Multiplexers** [2, 3, 4]
   This method uses a memory and multiplexers to implement the combinational part. It uses a property that in a sequential circuit, in many cases, the state transitions and output functions depend on proper subsets of the input variables. By using this property, we can reduce size of the memory. However, if any function depends on n variables, then the method requires a memory with n-bit address. For example, it would be impractical to use the memory with a size $n = 40$.
3. **Logic Simulation**
   This method uses a microprocessor and a memory. Given a logic circuit, it replace each gate with a fragment of program code. Then, it evaluates the code by a general-purpose microprocessor. This method stores both the data and the program in the memory. The evaluation time is proportional to the number of gates. The cost for the development is low, but the power dissipation is high compared with its performance.
4. **Look-Up Table (LUT) Ring** [8, 9]
   This method uses a control circuit, memories, registers and a programmable interconnection to emulate the sequential circuit. This method first represents the logic function by a BDD (Binary Decision Diagram), then transforms it into an LUT cascade. And, finally it emulates the cascade by an LUT ring. The LUT data are stored in a large memory. This method is faster than the logic simulator, since the number of memory references can be reduced.

In this paper, we will show a design method for a sequential circuit by using a Look-Up Table (LUT) ring. This paper is organized as follows: Section 2 introduces the cascade realization of logic functions. Section 3 shows the structure of LUT cascades and LUT rings. Section 4 describes a method to estimate the performance. Section 5 shows design algorithms for LUT ring. Section 6 shows experimental results. Finally, Section 7 concludes the paper.

II. Cascade Realization of Logic Functions

In this section, we will show a method to realize logic functions by a cascade of LUTs.

A. Representation of Multiple-output Logic Functions

Various decision diagrams (DDs) exist to represent multiple-output logic functions. Among them, MTBDD (multi-terminal BDD), BDD-for-ECFN (encoded characteristic function for non-zero outputs), and BDD-for-CF (characteristic function) are popular. In [12], we compared the evaluation time and the amount of memory of these DDs. They have the following features:

1. MTBDD: The width is too large to realize LUT cascades.
2. BDD-for-ECFN: The width is smaller than MTBDD, but its evaluation time is large when the number of outputs is large.

3. BDD-for-CF: The width of the BDD-for-CF is smaller than MTBDDs, and it can represent many outputs simultaneously.

Therefore, in this paper, we use a BDD-for-CF to represent a given multiple-output logic function.

B. Functional Decomposition using a BDD-for-CF [13]

Definition 2.1 *Let $\vec{f} = (f_1(X), f_2(X), \ldots, f_m(X))$ be a multiple-output function. Let $X = (x_1, x_2, \ldots, x_n)$ be the input variables, and $Y = (y_1, \ldots, y_m)$ be the output variables that denotes the outputs.* **The characteristic function of a multiple-output function** *is defined as*

$$\chi(X,Y) = \bigwedge_{j=1}^{m} (y_j \equiv f_j(X)).$$

The characteristic function of an n-input m-output function is a two-valued logic function with (n+m) inputs. It has input variables $x_i (i = 1, 2, \ldots, n)$, and output variables y_j for each output f_j. Let $B = \{0,1\}$, $\vec{a} \in B^n$, $\vec{f}(\vec{a}) = (f_1(\vec{a}), f_2(\vec{a}), \ldots, f_m(\vec{a})) \in B^m$, and $\vec{b} \in B^m$. Then, the characteristic function satisfies the relation

$$\begin{aligned} \chi(\vec{a},\vec{b}) &= 1 \; (if \; \vec{b} = \vec{f}(\vec{a})) \\ &= 0 \; (Otherwise). \end{aligned}$$

Definition 2.2 **The BDD-for-CF** *of a multiple-output function $\vec{f} = (f_1, f_2, \ldots, f_m)$ is the ROBDD for the characteristic function χ. In this case, we assume that the root node is in the top of the BDD, and variable y_j is below the support of f_j, where y_j is the variable representing f_j [5, 6, 7].*

Definition 2.3 **The width of the BDD-for-CF** *at height k is the number of edges crossing the section of the graph between x_k and x_{k+1}, where the edges incident to the same nodes are counted as one.*

Lemma 2.1 *[13] Let (X_1, Y_1, X_2, Y_2) be the variable ordering of the BDD-for-CF, representing a multiple-output function $\vec{f}$, and let W be the width of the BDD at the height $n_2 + m_2$. Then, $\vec{f}$ can be represented as*

$$\vec{f}(X) = g(h_1(X_1), h_2(X_1), \ldots, h_u(X_1), X_2). \qquad (1)$$

In this case, $\vec{f}$ can be realized by the network shown in Fig. 1, where the number of lines between H and G is $\lceil log_2 W \rceil$.

The representation of $\vec{f}$ in the form of Expression (1) is reffered as **functional decomposition**.

Theorem 2.1 *[8] Let μ_{max} be the maximum width of the BDD-for-CF that represents an n-input logic function $\vec{f}$. If $u = \lceil \log_2 \mu_{max} \rceil \leq k - 1$, then $\vec{f}$ can be realized by a cascade of k-LUTs as shown in Fig. 2. By applying functional decompositions $s - 1$ times, we have the circuit having the structure of Fig. 2.*

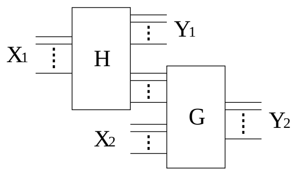

Fig. 1. Functional Decomposition with Intermediate Outputs.

III. LUT Cascade and LUT Ring

A. LUT Cascade [8]

An LUT cascade is shown in Fig. 2, where multiple-output LUTs (*cells*) are connected in series to realize a multiple-output function. The wires connecting adjacent cells are called **rails**. Let k_i be the number of inputs to the i-th cell, and let u_i be *the number of rail outputs* of the i-th cell, *i.e.*, the number of the rails between i-th cell and (*i+1*)-th cell. Let $|Y_i|$ be *the number of the external outputs* of the i-th cell, *i.e.*, the outputs that are connected to the primary output terminals. Let *s* be the number of cells in a cascade.

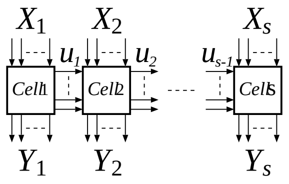

Fig. 2. LUT Cascade.

Lemma 3.2 *The size of the i-th cell is $2^{k_i} \cdot (u_i + |Y_i|)$. The amount of memory necessary to implement the cascade is given by*

$$M(X_1, Y_1, X_2, Y_2, \ldots, X_s, Y_s) = w \cdot \sum_{i=1}^{s} 2^{k_i}, \qquad (2)$$

where $k_i = |X_i| + u_{i-1}$, and $u_i + |Y_i| \leq w$.

The LUT cascade is simple and fast, but the restricted nature of its interconnections means it is not so flexible. Once the number of rails, inputs and outputs of cells, and the number of the cells are fixed, the number of functions realizable in the cascade is limited.

B. LUT Ring for Sequential Circuit [9]

By adding feedback lines between outputs and inputs of the LUT cascade shown in Fig. 2, we have an LUT ring. An LUT ring for sequential circuit is shown in Fig. 3.

It sequentially emulates an LUT cascade. Although it is slower than the LUT cascade, it has much more flexibility. In the LUT ring, the numbers of rails, input and outputs of cells, and the number of cells are flexible. We can consider an LUT ring with multiple units. However, for simplicity, in this paper, we will consider only the LUT ring with a single unit.

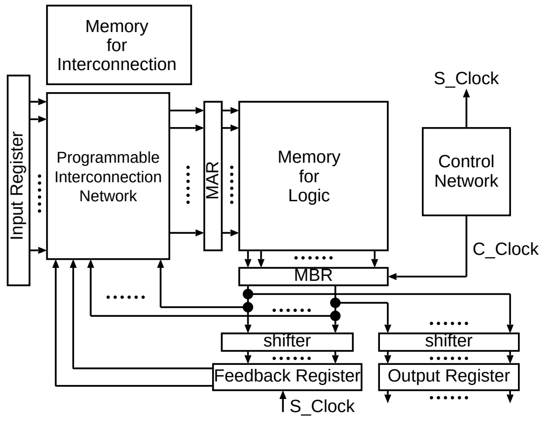

Fig. 3. LUT Ring for Sequential Circuit.

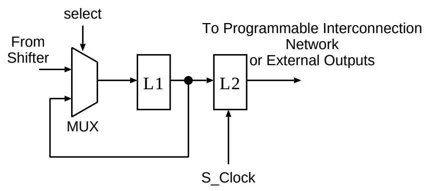

Fig. 4. Double-rank flip-flop.

In the LUT ring, all data for the cells are stored in a memory. The **Input Register** stores the values of the primary inputs; the **Feedback Register** stores the values of the state variables; the **Output Register** stores the values of external outputs; the **MAR** (Memory Address Register) stores the address of the memory; the **MBR** (Memory Buffer Register) stores the values of the outputs of the memory; the **Memory for Logic** stores the content for cells of the cascades; the **Programmable Interconnection Network** connects the Input register, the Feedback register, and the MAR, also it connects the MBR and the MAR; the **Memory for Interconnections** stores data for the interconnections; and the **Control Network** generates necessary signals to obtain functional values.

To emulate the sequential machine, the LUT ring uses two types of clock pulses: C_Clock to evaluate each cell of the LUT cascade, and S_Clock for state transitions. We use **Double-Rank Filp-Flops** in Fig.4 for the feedback register and the output register. Set the select signals to high when all the cells in a cascade are evaluated, and store the values into the latches L_1. When all the cascades are evaluated, the values of the state variables are sent to the latches L_2. This can be done by adding S_Clock.

In an LUT ring, all the least significant k bits of the starting address for k-input cells should be zeros. For example, the starting address of a 10-input cell in a 32-kilo-word memory should have the form xxxxx0000000000. Thus, the amount of memory actually needed to implement the LUT ring may be larger than the value obtained by Equation (2). We can reduce the number of levels of the cascade and/or total amount of memory by using cells with different numbers of inputs and/or by memory packing.

C. Emulation of a Sequential Circuit by an LUT Ring

A method to emulate a sequential circuit by the LUT ring is as follows:

1. Partition the outputs of the combinational part, and then, realize them by a set of LUT cascade. Since the combinational part of sequential circuit usually has many inputs and outputs, a direct implementation by a single memory is often impractical.
2. Emulate of the LUT cascade by the LUT ring. For example, Fig.5(b) shows a realization of the LUT cascade with three cells (Fig.5(a)) by an LUT ring.
3. In evaluating the outputs, for the outputs that becomes primary outputs, store them in the output register, while for the outputs that becomes state variables, store them in the feedback register.
4. When all the cascades are evaluated, transfer the values of the feedback register into the programmable interconnection network. Also, transfer the values of the output register to the primary outputs.

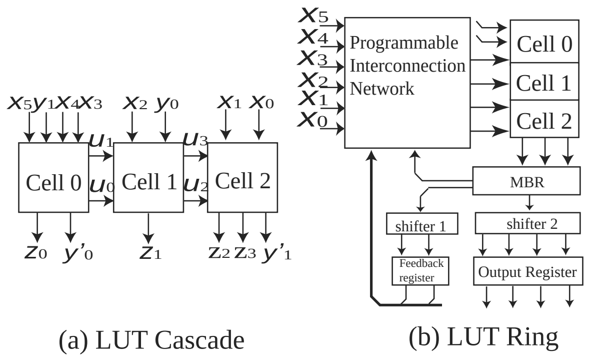

Fig. 5. LUT Cascade and corresponding an LUT Ring.

Example 3.1 *Fig.6 illustrates the emulation of the sequential circuit whose combinational part realizes the LUT cascade in Fig.5(a). In Fig.6, x_i denotes the input variables, y_i denotes the state variables, z_i denotes the primary output variables, and u_i denotes the rail variables.*

time = 1 *To evaluate the Cell 0, the two most significant bits of the address are set to (0,0) to specify page 0. Also, the input variables of Cell 0 (x_5, y_1, x_4, x_3) are set to the lower address bits through the programmable interconnection network as shown in Fig.6(a). By reading the contents of the page 0, the outputs of Cell 0 are stored in MBR. For the outputs that becomes the primary outputs, store them in the output register, while for the outputs that becomes state variables, store them in the feedback register.*

time = 2 *To evaluate Cell 1, the two most significant bits of the address are set to (0,1) to specify page 1. Also, the outputs of Cell 0 (u_1, u_0) are connected to the middle address bits, and the input variables of Cell 1 (x_2, y_0) are set to the least significant bits through the programmable interconnection*

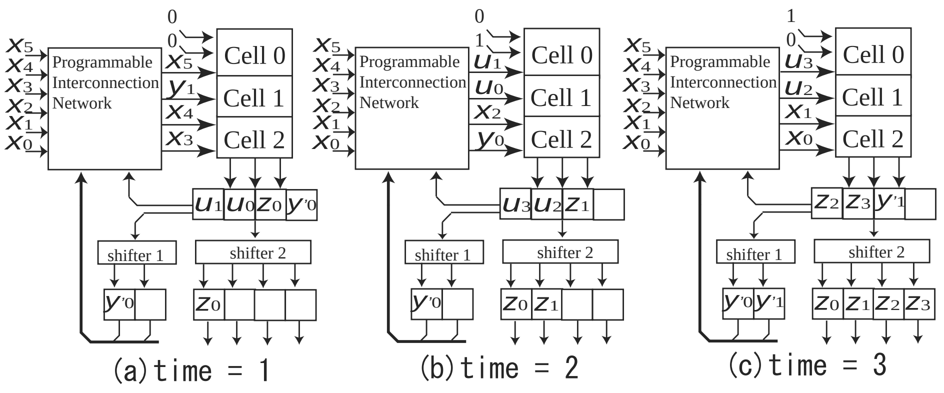

Fig. 6. Emulation of sequential circuit.

network, as shown in Fig.6(b). By reading the contents of page 1, the outputs of the Cell 1 are stored in MBR. For the outputs that becomes the primary outputs, store them in the output register, while for the outputs that becomes state variables, store them in the feedback register.

time = 3 *To evaluate Cell 2, the two most significant bits of the address are set to (1,0) to specify page 2. Also, the outputs of Cell 1 (u_3, u_2) are connected to the middle address bits, and the input variables of Cell 2 (x_1, x_0) are set to the least significant bits through the programmable interconnection network, as shown in Fig.6(c). By reading the contents of page 2, the outputs of Cell 2 are stored in MBR. For the outputs that becomes the primary outputs, store them in the output register, while for the outputs that becomes state variables, store them in the feedback register.*

Control Network sends the S_Clock to the feedback register and output register, and the values of feedback register are transfered into the programmable interconnection network, also the values of the output register are transfered to the primary outputs. *(End of Example)*

IV. Estimation of the Performance and the Amount of Memory

A. Lower Bounds on the Number of Levels of LUT Cascade

Theorem 4.2 *Let μ_{max} be the maximum width of the BDD-for-CF that represents an n-input m-output function $\vec{f}$. If $u_i \leq k-1$ and $u_i + |Y_i| \leq w$, then $\vec{f}$ can be realized by a LUT cascade with k-input w-output shown in Fig. 2. Let s be the number of levels of the LUT cascade in Fig. 2. Then, we have the following relation:*

$$\max\left\{\left\lceil\frac{n+u-2}{k-1}\right\rceil, \left\lceil\frac{m+u-2}{w-1}\right\rceil\right\} \leq s, \tag{3}$$

where $u = \max\{u_i\}$, and $u_i \leq \lceil\log_2 \mu_{max}\rceil$. When $n \leq k$, $\vec{f}$ can be realized by one cell.

Corollary 4.1 *Let $\vec{f}(X)$ represent an n-input m-output function. Suppose that $\vec{f}(X)$ cannot be represented as $\vec{f}(X) = g(h(X_1), X_2)$, where (X_1, X_2) is a partition of X. Let s be the number of k-input w-output cells to realize $\vec{f}(X)$. Then, we have the following relation: $l_{low} \leq s$, where*

$$l_{low} = \max\left\{\left\lceil\frac{n-2}{k-2}\right\rceil, \left\lceil\frac{m-2}{w-2}\right\rceil\right\}. \tag{4}$$

We can estimate the number of levels of the LUT cascade by using expression (4).

B. Relation Between the Amount of Memory and the Number of Cell Inputs

Theorem 4.3 *Suppose that an n-variable logic function $\vec{f}$ is realized by the cascade of k-inputs shown in Fig. 2. Let L[bit] be the total amount of available memory; μ be the width of the BDD; k be the maximum inputs of cells; and w be the number of output bits of the Memory for Logic. If $\vec{f}$ has no non-trivial simple disjoint decomposition, then we have*

$$\frac{2^k}{k-2} \leq \frac{L}{w(n-2)}, \tag{5}$$

where $n \geq k$.

Example 4.2 *The benchmark function s208 has $n = 18$ inputs. Let us realize it on a memory with $L = 2^{20}(= 1Mbits)$, and $w = 16$. From Theorem 4.3, it is sufficient to consider k with values for $k \leq 16$.* *(End of Example)*

V. Realization of Sequential Circuits by an LUT Ring

A. Formulation of the Design Problem in an LUT Ring

For many benchmark functions, BDD-for-CFs are too large to construct. Also, even if the BDD-for-CF is stored in a memory of a computer, it can be too large to be realized by an LUT cascade [13].

Definition 5.4 *Assume that the output functions are partitioned into r groups, and for each groups of outputs, and each LUT cascade realize the functions in the group. Let l_i $(i = 1, 2, \ldots, r)$ be the number of levels in each LUT cascade.*

Note that, the total number of cells is $l_1 + l_2 + \cdots + l_r$.

We partition the outputs so that each set of outputs depends on as small number of input variables as possible. Since the computation time of an LUT ring is proportional to the total number of cells in the cascades, we can formulate the design problem for an LUT ring as follows:

Problem 5.1 *Given a multiple-output function $\vec{f} = (f_1(X), f_2(X), \ldots, f_m(X))$ and an ordering of the input variables X, obtain the partition of X that satisfies the following conditions:*

1 . *The total amount of memory is at most L_0.*
2 . *The number of cells of the cascade is the minimum subject to Condition 1.*
3 . *The total amount of memory is the minimum subject to Condition 2.*

B. Estimation of the Maximum Number of Outputs in Each Group

Partitioning the outputs into groups after representing a large single BDD-for-CF is inefficient: The number of nodes of the BDD-for-CF is so large that the optimization of the BDD-for-CF is very time consuming. Previous approach [13] reorders the output functions so that the support will increase as slowly as possible. Then, it finds an ordering of the input and output variables to construct a minimum BDD-for-CF. Finally, it generates cascades form the BDD-for-CF. However, this approach requires much computation time for variables ordering. Also, this approach generates cascades with large number of levels, since the BDD-for-CFs with many outputs tend to have large widths, and produce cascades with large number of levels. In this paper, we use the strategy to make many short cascades rather than to make a single long cascade. This strategy requires much shorter computation time than the previous approach, and generates cascades with smaller total number of levels. In the following algorithm, the user must specify the maximum number of cell inputs.

We can estimate the total number of external outputs of cells, except for the last cell as follows:

$$(the\ average\ number\ of\ external\ outputs\ of\ a\ cell)$$
$$\times(the\ number\ of\ levels\ of\ LUT\ cascades - 1).$$

We designed LUT cascades, and obtained the statistical data of the number of levels and external outputs for each cell for 25 MCNC89 benchmark functions. To obtain the statistical data, we omitted the cascades consisting of only a single cell. We assumed that the distributions of average number of cells and levels are normal distributions, and calculated the 95% confidence interval.

Fig.7 plots the average number of levels in an LUT cascade. The vertical axis denotes the number of cells; the horizontal axis denotes the number of cell inputs; *arithmetic mean* denotes that of the number of cells; *confidence interval* denotes the 95% confidence interval of the average number of cells; and *alpha* denotes the estimation for the number of levels Expression (4) multiplied by $alpha$, where $\alpha = 1.5$. From Fig.7, the number of levels per single cascade, that most inputs of cells are occupied by external outputs, approaches the bound of Expression (7) in the case of total number of levels are minimum. Also, the number of LUT cascades increases compared with the previous approach. Hence, the total number of levels of LUT cascades decrease even if the number of LUT cascades increase. From this experiment,

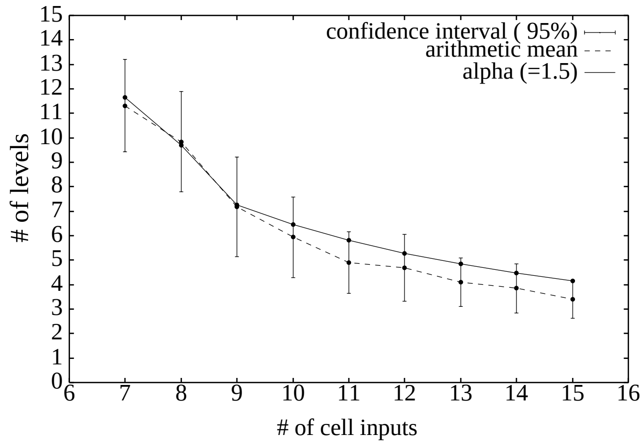

Fig. 7. Statistic of Levels.

we have an estimation of number of levels of an LUT cascade, i.e., α multiplied by l_{low}, where l_{low} is given by Expression (4).

Fig.8 plots the average numbers of external outputs for each cell. The vertical axis denotes the number of external outputs for each cell; the horizontal axis denotes the number of cell inputs; *arithmetic mean* denotes that of the number of external outputs; *confidence interval* denotes the 95% confidence interval of the average number of external outputs of cells; and *beta* denotes the maximum number of rails $k-1$ multiplied by 0.25, i.e.,$\beta(k-1)$, where $\beta = 0.25$

From these results, we can introduce the parameter T that estimates the number of outputs in a group.

Definition 5.5 *Given an n-input m-outputs logic function, let k be the number of maximum inputs of cells, let w be the number of outputs of a cell, let α, and β be the positive constants. Define the parameter T that estimates the number of outputs in a group:*

$$T = \beta(k-1)(\lceil l_{low} \rceil - 1) + w, \qquad (6)$$

where l_{low} is given by the Expression(4).

The first term of Expression 6 estimates the total number of external outputs of cells except for the last cell. The second term of Expression 6 estimates the number of outputs in the last cell.

C. Partition of the Outputs

Definition 5.6 *The* **support** *of a function f is the set of variables on which f actually depends.*

Definition 5.7 *Let $F = \{f_0, f_1, \ldots, f_{m-1}\}$ be the set of the output functions, let G be a subset of F, and $f_i \in F - G$. Then, the* **similarity** *of the supports of f_i with G is defined as follows:*

$$Similarity(i, G, F) = |Sup(f_i) \cap Sup(G)|, \qquad (7)$$

where Sup(F) denotes the set of supports of the functions in F.

We partition the outputs into groups using this similarity so that each group of outputs depends on as small number of inputs as possible.

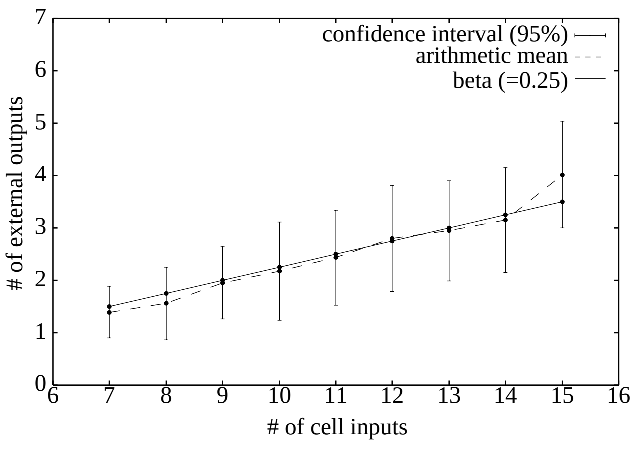

Fig. 8. Statistic of External Outputs of Cells.

Algorithm 5.1 *(Partition of Outputs and Realization of Cascades) Let $F = \{f_0, f_1, \ldots, f_{m-1}\}$ be the set of the output functions to be realized. Let T be the maximum number of outputs in a group, given by Expression (6). Let G be a subset of F; let Non_Cas be the set of outputs not realized by cascade; let $n_{Non\_Cas\_group}$ be the number of groups that are not realized by cascade; and let max_{n_group} be the maximal number of outputs in a group. In the following algorithm, the user must specify the maximum number of cell inputs k.*

```
1. Read the data and parameters such as $k$. Compute $T$ by Expression (6).
2. $F \leftarrow \{f_0, f_1, \ldots, f_{m-1}\}$, $max_{n\_group} \leftarrow T$.
3. While $F \neq \phi$ do steps (a)(b), and (c)
    (a) $Non\_Cas \leftarrow \phi$, $n_{Non\_Cas\_group} \leftarrow 0$.
    (b) While $F \neq \phi$ do steps $(i)$ through $(v)$
        i. $G \leftarrow f_t$, where $f_t \in F$, and $t$ is the suffix of the function with the minimum $|Sup(f_j)|$.
        ii. While $|G| \leq max_{n\_group} - 1$ do
            A. Find $f_i$ with the maximum Similarity(i, G, F), where $f_i \in F - G$.
            B. Let $G \leftarrow G \cup \{f_i\}$, and $F \leftarrow F - \{f_i\}$.
        iii. (Try to realize a cascade for the functions in G.)
        iv. If (the cascade is not realized) then
            if $|G| = 1$ then (write ‘Impossible to realize‘ and Terminate.)
            else ($n_{Non\_Cas\_group} \leftarrow n_{Non\_Cas\_group} + 1$, $Non\_Cas \leftarrow Non\_Cas \cup G$. )
        v. $F \leftarrow F - G$.
    (c) $F \leftarrow Non\_Cas$,
        $max_{n\_group} \leftarrow |F|/(n_{Non\_Cas\_group} + 1)$.
4. Terminate.
```

D. Reduction of the Number of Levels Using Unused Memory

Although Algorithm 5.1 gives fairly good solutions, it does not always produce LUT cascades with the minimum levels, since it is a heuristic method. If there is any spare memory, we can decrease the number of levels by using LUTs with more inputs. In this section, we show an algorithm to reduce the number of levels using unused memory.

Algorithm 5.2 *(Reduction of Levels) By Algorithm 5.1, obtain the cascade for a given multiple-output function. Let r be the number of LUT cascades, and let mem be the total amount of necessary memory. Let $\mathcal{GRPS} = \{G_1, G_2, \ldots, G_r\}$ be the set of groups of the outputs, and lv_G_i be the number of levels of the LUT cascade for G_i. In the following ,the user must specify the number of cell inputs k_{lim}, and the amount of available memory mem_{avail}.*

```
1. Read the parameters $k_{lim}$ and $mem_{avail}$.
2. While $\mathcal{GRPS} \neq \phi$ do steps (a)(b), and (c).
    (a) For each $G_i \in \mathcal{GRPS}$. Let $bdd_{cf}$ be the BDD-for-CF, representing $G_i$. Let, $\mathcal{GRPS} \leftarrow \mathcal{GRPS} - \{G_i\}$.
    (b) Obtain the maximum number of cell inputs $k_{max}$ by Expression (5).
    (c) For $k \leftarrow k_{lim}$ until $k_{max}$ do steps (i) through (iv)
        i. Try to realize a cascade for the $bdd_{cf}$ with the maximum number of inputs $k$, let $j$ be the levels of cascade.
        ii. Let $mem_{pack}$ be the amount of necessary memory after memory packing.
        iii. If ($j < lv_{G_i}$ and $mem_{pack} \leq mem_{avail}$) then ($lv_{G_i} \leftarrow j$, and $mem \leftarrow mem_{pack}$)
             else if ($j = lv_{G_i}$ and $mem_{pack} < mem$) then $mem \leftarrow mem_{pack}$.
        iv. $k \leftarrow k + 1$.
3. Terminate.
```

VI. Experimental Results

A. Implementation of ISCAS'89 Benchmark Functions with an LUT Ring

We implemented Algorithms 5.1 and 5.2 in the C programming language, and designed LUT rings for ISCAS'89 benchmark functions [14]. In the LUT ring, we assumed the following conditions: The number of bits in a word of the memory for logic is 16; and the number of the maximum cell inputs is 13.

Table I shows the number of levels and the amount of memory for the LUT rings. *Name* denotes the name of benchmark function; *In* denotes the number of inputs; *Out* denotes the number of outputs; *FF* denotes the number of flip-flops; Sup denotes the maximum number of supports in the output groups; r denotes the number of LUT cascades; s denotes the number of levels; and *Mem* denotes the amount of memory (Mega Bits).

First, we obtained partitions without memory limitation (*Lower Bound*) using relation (2). Second, we obtained the partitions of outputs by Algorithm 5.1, and we reduced the number of levels by Algorithm 5.2 with the memory limitation of 1 Mega Bytes (*Limit 1MByte*). Finally, we obtained the partitions by Algorithm 5.1, and we reduced the number of levels by Algorithm 5.2 with the memory limitation of 4 Mega Bytes (*Limit*

TABLE I

THE AMOUNT OF MEMORY AND THE NUMBER OF CELLS FOR LUT RINGS.

Name	In	Out	FF	Sup	r	Lower Bound			Limit 1MByte			Limit 4MByte		
						s	Mem[Mbit]		s	Mem[Mbit]		s	Mem[Mbit]	
							Non-Pack	Pack		Non-Pack	Pack		Non-Pack	Pack
s208	10	1	8	18	1	2	0.625	0.312	1	4.000	4.000	1	4.000	4.000
s344	9	11	15	21	1	3	0.312	0.312	3	0.312	0.312	3	0.312	0.312
s386	7	7	6	13	1	1	0.125	0.125	1	0.125	0.125	1	0.125	0.125
s420	18	1	16	34	1	5	0.281	0.156	3	2.500	1.500	2	36.000	32.000
s510	19	7	6	22	1	3	0.062	0.062	2	0.375	0.250	2	0.375	0.250
s641	36	23	19	37	2	11	1.188	0.813	7	5.250	4.750	6	14.000	13.500
s713	36	23	19	35	2	11	0.812	0.562	7	4.750	4.250	6	9.750	9.500
s820	18	19	5	23	1	3	0.093	0.062	3	0.093	0.062	3	0.093	0.062
s838	34	1	32	66	2	14	1.390	0.937	8	7.500	5.500	7	19.000	15.000
s1196	13	13	19	26	3	10	0.688	0.531	5	7.531	7.000	5	7.531	7.000
s1423	17	5	74	65	5	43	3.968	2.438	32	12.453	7.563	25	42.937	30.750
s5378	35	49	164	72	22	76	4.891	2.332	61	15.055	7.609	52	65.359	30.828
s9234	36	39	211	94	41	136	6.998	2.102	121	17.271	7.875	95	101.859	31.546
ratio						1.00			0.73			0.66		

4MByte). In this experiment, we used two algorithms: the algorithm to find a partition of inputs that minimizes the number of levels [10], and the algorithm for memory packing [10].

In the columns of Mem, *Non-Pack* denote the sizes of memory without memory packing, and the *Pack* denote the sizes of memory with memory packing. In the case of *Limit 1MByte*, we achieved a reduction in the number of levels to 73%. Furthermore, in the case of *Limit 4MByte*, we achieved a reduction in the number of levels to 66%. Therefore, with the enough amount of memory, we can considerably reduce the number of levels.

B. Comparison with Other Method

B.1 Memory Size

Implementation using Memory and Multiplexers [4] Consider the sequential circuit, where n is the number of the external input variables, m is the number of the external output variables, and s is the number of the state variables. The straightforward implementation of the transition functions and the output functions by memories requires $s \cdot 2^{n+s}$ bits and $m \cdot 2^{n+s}$ bits, respectively.

However, we can often reduce the necessary amount of memory by using properties of the given sequential circuits. In many case, the transition functions and output functions of sequential circuit depend on the proper subset of the input variables. Let q be the maximal number of the input variables for which the next state depends. Then, we can use q qualifier variables instead of the external input variables. In this case, we use current state to select the external input variables. Also, we need 2^s:1 multiplexers to select the qualifier variables from the external inputs.

Example 6.3 *Consider the state transition table shown in Table 1. In this table, the number of external inputs is three, but each state depends on at most two external inputs. Thus, by using the circuit in Fig. 9, we can implement Table II. In Fig. 9, two 8:1 multiplexer select qualifier variables. Note that with this implementation, the size of memory is reduced from* $3 \cdot 2^{3+3}$ *to* $3 \cdot 2^{2+3}$ *bits.* *(End of Example)*

Logic Simulation using Microprocessor The LCC [15] is a kind of logic simulator that assigns a fragment of program code to each gate of logic circuits. It evaluates codes from the inputs

TABLE II

EXAMPLE OF STATE TRANSITION TABLE.

$\vec{x}$			$\vec{y}_{current}$			$\vec{y}_{next}$		
x_1	x_2	x_3	y_1	y_2	y_3	y'_1	y'_2	y'_3
1	-	-	0	0	0	0	0	1
0	0	-	0	0	0	1	0	1
0	1	-	0	0	0	1	0	0
-	-	1	0	0	1	1	0	1
-	-	0	0	0	1	0	1	1
0	-	-	1	0	1	0	1	1
1	1	-	1	0	1	1	0	1
1	0	-	1	0	1	0	0	1
-	1	0	1	0	0	1	0	1
-	0	0	1	0	0	0	0	1
-	-	1	1	0	0	0	1	0
-	-	-	0	1	1	0	0	1
-	0	-	0	1	0	0	1	0
-	1	-	0	1	0	0	0	0

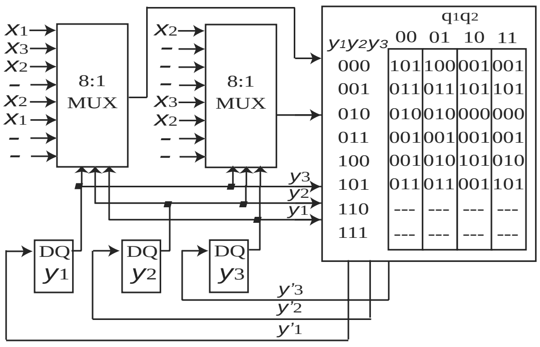

Fig. 9. Implementation of Table II using memory and multiplexers.

to the outputs in a topological order. To produce the executable code, we converted the ISCAS'89 benchmark circuits into the program code, and compiled it by gcc compiler with option -O2.

Table III compares LUT rings and other three method. In Table III, *Name* denotes the name of benchmark function; *In* denotes the number of inputs; *Out* denotes the number of outputs; *FF* denotes the number of state variables; *Direct* denotes the size of memory that is directly implemented by a single memory. *Mem+Mux* denotes the amount of memory by using multiplexers [4]. *LCC* denotes that the size of executive code. *Ring* denotes the amount of memory for the memory for logic in Fig.3. We found the partition of the inputs that minimizes the number of levels, and did packing [10].

TABLE III
COMPARISON OF MEMORY SIZES[MBIT] TO EMULATE SEQUENTIAL CIRCUITS.

Name	In	Out	FF	Direct	Mem+Mux	LCC	Ring
s298	3	6	14	2.500	1.250	0.015	0.187
s344	9	11	15	416.000	208.000	0.015	0.187
s382	3	6	21	432.000	216.000	0.015	0.156
s386	7	7	6	0.102	0.025	0.016	0.125
s400	3	6	21	432.000	216.000	0.015	0.156
s820	18	19	5	192.000	0.188	0.019	0.503
s1494	8	19	6	0.391	0.049	0.025	0.500

Table III shows that the method using Memory and Multiplexer realizes the benchmark functions with small memory sizes, when the number of state variables or FFs is small. However, when the number of state variables is large, the necessary amount of memory becomes too large. LCC and Ring can realize the benchmark functions with smaller amount of memory than Direct and Mem+Mux methods.

B.2 Evaluation Time.

Table IV compares the LUT rings with the LCC logic simulator with respect to the evaluation time. In this table, we generated one million random test vectors on an IBM PC/AT compatible machine using a Pentium III 800MHz microprocessor with 256MBytes of memory. We obtained average evaluation time per one vector, and we considered it as the LCC evaluation time. Also, from the expression in [11], evaluation time for the LUT Ring was estimated as follows:

$$\text{Evaluation time [ns]} = 4.5 \times \text{Number of levels} + 5.9.$$

In Table IV, *Name*, *In*, *Out*, and *FF* denote the same things as Table III. And *LCC*, and *LUT Ring* denote evaluation time (ns) for correspond methods.

TABLE IV
COMPARISON OF EVALUATION TIME FOR LCC AND LUT RING.

Name	In	Out	FF	LCC[ns]	LUT Ring[ns]
s208	10	1	8	740	10.4
s349	9	11	15	1200	14.9
s420	18	1	16	1510	14.9
s510	19	7	6	2140	14.9
s641	36	23	19	1800	32.9
s713	36	23	19	2170	32.9
s820	18	19	5	4080	19.4
s1196	13	13	19	5430	28.4
s1494	8	20	6	7830	14.9
s5378	35	49	164	19860	239.9
s9234	36	39	211	38400	275.9

Table IV shows that the LUT ring is about 100 times faster than the LCC. Note that the LCC stores both the data and the program in a memory, and execute by a general-purpose processor. For these benchmark functions, LUT rings efficiently realize sequential circuits using the reasonable amount of memory in a reasonable evaluation time.

VII. CONCLUSION

In this paper, we presented a method to realize sequential circuits by using Look-Up Table (LUT) rings. This method partitions output functions into groups, represents them by BDD-for-CFs, and emulates by an LUT ring. The strategy presented in this paper reduced the design time drastically. And, we could reduce the number of levels for LUT rings by using unused memory. Also, the LUT ring is faster than microprocessors.

VIII. ACKNOWLEDGMENTS

This research is partly supported by Japan Society for the Promotion of Science (JSPS), MEXT, and Kitakyushu Innovative Cluster. Discussions with Prof. Iguchi were quite useful.

REFERENCES

[1] R. K. Brayton, ”The future of logic synthesis and verification,” *in S. Hassoun and T. Sasao (e.d.), Logic Synthesis and Verification, Kluwer Academic Publishers, 2001.*

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

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

[4] D. Green, *Modern Logic Design,* Addison-Wesley Publishing Company, 1986.

[5] P. Ashar and S. Malik, ” Fast functional simulation using branching programs,” *In Proc. of the Intl. Conf. on Computer-Aided Design*, pp. 408-412, Nov. 1995.

[6] P. McGeer, K. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, and P. Scaglia, ” Fast discrete function evaluation using decision diagrams,” *In Proc. of the Intl. Conf. on Computer-Aided Design,* pp. 402-407, Nov. 1995.

[7] Y. Iguchi, T. Sasao and M. Matsuura, ”Evaluation of multiple-output logic functions using decision diagrams,” ASP-DAC 2003, (Asia and South Pacific Design Automation Conference 2003), Kitakyushu, Jan. 21 - 24, 2003, pp. 312-315.

[8] T. Sasao, M. Matsuura, and Y. Iguchi, ”A cascade realization of multiple-output function for reconfigurable hardware,” *IWLS-2001*, Lake Tahoe , CA, June 12-15,2001, pp.225-300.

[9] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, ”Realization of sequential circuits by look-up table ring,” *MWSCAS2004*, Hiroshima, July 25-28, 2004, I517-I520.

[10] T. Sasao, M. Kusano, and M. Matsuura, ”Optimization methods in look-up table rings,” International Workshop on Logic and Synthesis (IWLS-2004), June 2-4, 2004, Temecula, California, USA, pp.431-437.

[11] H. Qin, T. Sasao, M. Matsuura, S. Nagayama, K. Nakamura, Y. Iguchi, ”Realization of multiple-output functions by sequential look-up table cascades,” *Technical Report of IEICE,* VLD2003-127, pp.13-18, Yokohama, Jan.2004.

[12] Y. Iguchi, T. Sasao and M. Matsuura, ”Evaluation of multiple-output logic functions using decision diagrams,” ASP-DAC 2003, (Asia and South Pacific Design Automation Conference 2003), Kitakyushu, Jan. 21 - 24, 2003, pp. 312-315.

[13] T. Sasao and M. Matsuura, ”A method to decompose multiple-output logic functions,” 41st Design Automation Conference, San Diego, CA, USA, June 2-6, 2004, pp.428-433.

[14] http://www.cbl.ncsu.edu/pub/Benchmark_dirs/ISCAS89/

[15] M. Abramovici, M. A. Breuer, and A. D. Friedman, ”Digital Systems Testing and Testable Design, ” Wiley-IEEE Press; Rev. Print edition, Sept. 1994.