A Soft Error Tolerant LUT Cascade Emulator

Hiroki Nakahara and Tsutomu Sasao
Department of Computer Science and Electronics,
Kyushu Institute of Technology, Iizuka 820-8502, Japan

Abstract

An LUT cascade emulator realizes an arbitrary sequential circuit. Given a sequential circuit, we convert the combinational part into one or more LUT cascades, and store LUT(cell) data into a memory in the LUT cascade emulator. The emulator evaluates multi-output logic functions by reading cell data sequentially. To improve the tolerance to soft errors, cell data in the memory are encoded by error correcting codes. Also, error-correcting circuits and checking circuits that periodically scan the memories are appended. When a soft error is detected, it removes the error by rewriting the correct data into the memory. To mask soft errors in flip-flops, a TMR (Triple Module Redundancy) technique is employed. Our system detects a soft error in a single bit. Also, the mission time of the system is more than 1000x of time of an ordinary LUT cascade emulator.

1 Introduction

With the decrease of the feature size of LSIs, the decrease of the reliability that comes from the variation of devices or random failure, becomes the problem [8]. Soft errors induced by thermal neutrons, cosmic rays, or alpha particles hitting the surface of silicon devices at random, are irreproducible [7]. Soft errors are the major causes of random failure. With the reduction of supply voltage and the size of transistors, the electric charge stored in a semiconductor element also decreases. The amount of critical electric charge that flips the data also decreases. As a results, soft errors occur not only in the outer space, but also on the ground. Furthermore, the frequently of soft errors is not negligible [14].

Many methods to prevent soft errors exist [15]. One method is to use radiation-hardened devices. However, such devices are much more expensive than ordinary devices. Furthermore, the performance of radiation-hardened devices is several generations behind state-of-the-art devices. A realistic, low-cost and high-performance solution is to add redundancy to the circuits. SRAMs, where each word is encoded by an ECC (Error Correcting Code), can detect and correct soft errors. Since a soft error does not destroy the semiconductor device, we can remove the soft error by rewriting the correct data to it. For SRAM-based FPGAs, a technique that periodically checks the configuration data has been developed [5]. When it detects a soft error in the FPGA, it rewrites the configuration bits.

We have proposed a look-up table (LUT) cascade emulator. An LUT cascade, where multiple-output LUTs are connected in series, realizes a multiple-output function [19]. An LUT cascade emulator consists of a control part, memories, and registers. Each register is connected to the programmable interconnection circuit, and the LUT cascade emulator evaluates the logic circuit stored in the memory [19]. The LUT cascade emulator realizes an arbitrary sequential circuit by storing logic data and its interconnection data in memories. Compared with an ordinary microprocessor, the LUT cascade emulator is about 10 times faster, while its power consumption is smaller. Compared with FPGAs, the LUT cascade emulator is several times slower, and power dissipation is larger. However, since the place and routing design in the LUT cascade emulator is unnecessary, the design time for the LUT cascade emulator is shorter than FPGAs, and the delay estimation for the LUT cascade emulator is also more accurate than that of FPGAs. By applying the method described later in this paper, the mission time of the LUT cascade emulator can be improved. For soft errors, our LUT cascade emulator only need to rewrite a part of the memory, while an FPGA needs to rewrite all the configurations. In this paper, we will present a soft error tolerant LUT cascade emulator.

The rest of the paper is organized as follows: Section 2 presents the LUT cascade emulator. Section 3 presents error models of the LUT cascade emulator. Section 4 presents the soft error tolerant LUT cascade emulator. Section 5 evaluates the reliability of the proposal method. Finally, Section 6 concludes the paper.

2 LUT Cascade Emulator

2.1 LUT Cascade

An LUT cascade is shown in Fig. 1, where multiple-output LUTs (**cells**) are connected in series to realize a

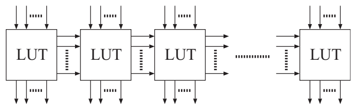


Figure 1. LUT cascade.

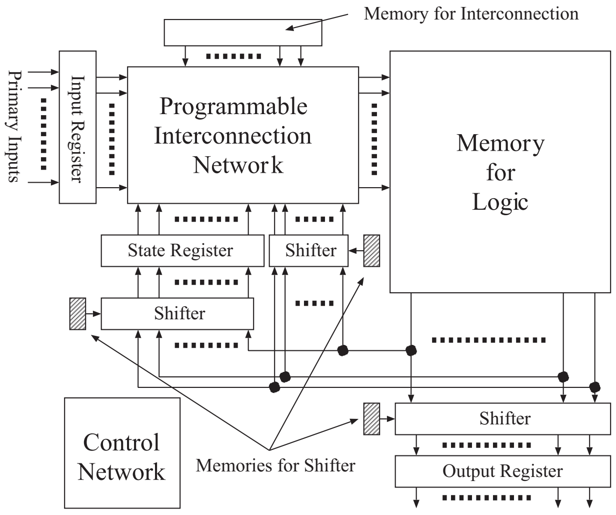


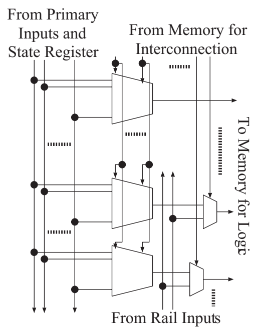
Figure 2. LUT cascade emulator.

multiple-output function. The wires connecting adjacent cells are called **rails**. Also, each cell may have external outputs in addition to the rail outputs. Estimation of the delay time is easier than FPGAs and ASICs, since the interconnections are limited to adjacent cells in the LUT cascade. We can obtain an LUT cascade by applying **functional decompositions** repeatedly to the BDD (BDD_for_CF) [12] that represents the multiple-output function [18]. Thus, logic synthesis is relatively easy for the LUT cascade.

2.2 LUT Cascade Emulator

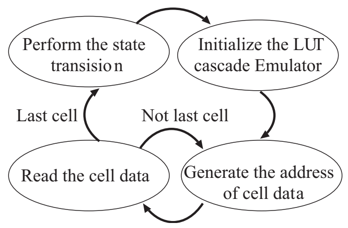
Fig. 2 shows an **LUT cascade emulator** for a sequential circuit. Although it is slower than the LUT cascade, it is much more logically flexible than the LUT cascade [19].

The LUT cascade emulator stores the cell data of LUT cascades in the **Memory for Logic**; the address lines of cell are connected from inputs, state variables, and rail outputs of the preceding cell through the **Programmable Interconnection Network**; and the **Memory for Interconnection** stores data for the interconnections. The LUT cascade emulator reads the cell outputs from the memory for logic, and send them to the **State Register** and the **Output Register** through **Shifters**; the **Memories for Shifter** store data for the shifters; the **Input Register** stores the values of the primary inputs; and the **Control Network** generates necessary control signals to obtain function values. The following example shows an emulation of a sequential circuit on an LUT cascade emulator.



From Primary Inputs and State Register
From Memory for Interconnection
To Memory for Logic
From Rail Inputs

Figure 3. Programmable interconnection network.



Perform the state transision
Initialize the LUT cascade Emulator
Last cell
Not last cell
Read the cell data
Generate the address of cell data

Figure 4. State diagram for LUT cascade emulator.

Example 2.1 *1. Initialize the LUT cascade emulator:*
Set the input register, and initialize the state register, and let cell number$\leftarrow 0$.

2. *Generate the address of cell data:*
   The programmable interconnection network connects the inputs, state variables, and rail outputs of the preceding cell according to the connection data from the memory for interconnection.
3. *Read the cell data:*
   Read the content of the memory for logic using address generated in Step 2. Then, send the values obtained from the memory for logic to the output register and the state register through the shifters. If (cell number $<$ *total number of cells), then (cell number*$\leftarrow$ *cell number* $+1$*), and goto Step 2.*
4. *Compute the state transition:*
   Update the values of the state register. Also, send the values of the output register to the output terminal, and go to Step 1. *(End of Example)*

Fig. 4 shows the state diagram described in Example 2.1. In an LUT cascade emulator, let c be the number of cells, n be the number of primary inputs, m be the number of primary outputs, s be the number of state variables, and r be the maximum number of rails. In Fig. 3, the number of control signals for MUXs that selects the primary inputs and state inputs is $\lceil log_2 n + s\rceil$. Also, r MUXs select the primary inputs or the rail inputs. Thus, the amount of the memory for interconnection is

$$M_{ic} \quad = \quad c(\lceil log_2(n+s)\rceil + r)[bit]. \qquad (1)$$

The total amount of memory for shifters is

$$M_{sift} \quad = \quad c(\lceil log_2 m\rceil + \lceil log_2 s\rceil + \lceil log_2 r\rceil)[bit]. \quad (2)$$

Table 1. Sizes of LUT cascade emulator.

Name	#In	#Out	c	r	M_{ic} [bit]	M_{sift} [bit]	M_{logic} [Mbit]
C432	36	7	4	14	1197	28	0.968
C1908	33	25	63	13	80	207	1.000
apex7	49	37	13	14	260	130	1.000
too_large	38	3	9	13	171	54	0.500
rot	135	107	201	10	3618	2211	1.000

c:# of cells r:Max. # of rails

Let the amount of the memory for logic (M_{logic}) be 1[Mega bit](32$k\times$ 32). We implement several MCNC benchmark functions [16] on the LUT cascade emulator using our synthesis method [6]. Table 1 shows M_{ic}, M_{sift}, and M_{logic}. It shows that the memory for logic occupies the most chip area in the LUT cascade emulator.

3 Error Model of the LUT Cascade Emulator

In this section, we will consider the error model of the LUT cascade emulator.

3.1 Failure Period

We assume that a system consists of multiple elements. The abnormalities of the output of the system denotes a **failure**. Also, the abnormalities of the output of the elements denotes an **error**. We assume that failures are caused by one or more errors. Errors are roughly divided into **hard errors** and **soft errors**. The hard error denotes an element (i.e. transistor, capacitor, interconnection, etc.) destroyed physically. The soft error[1] also called **SEU (Single Event Upset)**, is a bit-flip fault. It occurs when the radiation energy forms an electric charge on a semiconductor storage element. The failure period is divided into three as shown in Fig. 5 according to the time of the fault. In the **early failure period**, failures occur at the manufacture by defects or inferior materials. We assume that both hard and soft errors occur in this period. In the **intrinsic failure period**, failures occur with a fixed probability. We assume that only soft errors occur in this period. In the **wear-out failure period**, failures occur by age and fatigue. In this period, both hard and soft error occur. In this paper, we only consider the reliability in the intrinsic failure period. To make this assumption realistic, we detect all the early failures by the burn-in test.

[1]In a wider sense, it includes defects of operation that are caused by the drift of the supply voltage or the temperature. In this paper, we only consider the influence of radiations.

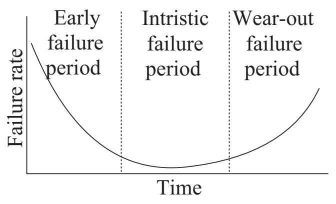


Figure 5. Failure rates.

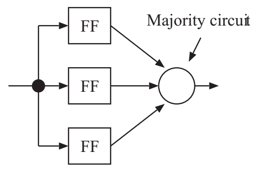


Figure 6. TMR flip-flop.

3.2 Assumption of the Failures

We assume that most failure occur in the memories, since most chip area for the LUT cascade emulator is occupied by memories. The early failures are detected by the standard test for memories [9, 1]. Then, we can mask the failures by exchanging spare-cells or memory-packing that skips the failure part of memory cells [17]. Since chips with the early failures are detected by the burn-in test, we do not consider the early failures in the intrinsic failure period. We also assume that, except for the memories, devices have enough margin. Thus, we can assume that the failure probability of the devices other than memory is negligibly small. Thus, in this paper, we only consider soft errors for memories and flip-flops in the intrinsic failure period.

4 Soft Error Tolerance Technique for LUT Cascade Emulator

4.1 Assumption of Soft Errors

We assume that soft errors only occur at memory cells and flip-flops that store electric charge. Also, we only consider a single-bit soft errors in a word.

4.2 A Soft Error Tolerance for Flip-flops

In the advanced design rules of VLSIs, such as 90nm, 65nm, and 45nm, soft errors also occur in flip-flops [10]. Many methods exist to protect against soft errors in flip-flops: Append a mask latch at inputs and inside of the circuit [10]; construct a soft error tolerant latch inside the circuit [11]; and insert a low-pass filter in the latch in order to mitigate a soft error influence [2]. In this paper, we implement the registers using **TMR (Triple Modular Redundancy)** flip-flops. Fig. 6 illustrates a TMR flip-flop. To improve the reliability, three flip-flops are used to store a bit, and the majority for the outputs of three flip-flops is produced at the output.

Table 2. Number of bits for SEC code

information part (bits)	check part (bits)
16	6
32	7
64	8
128	9

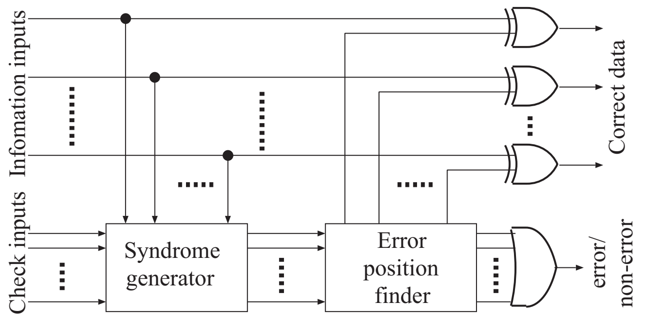


Figure 7. A single-error correcting circuit for the memory using the SEC code.

4.3 Masking Soft Errors in Memory-Cells using Error Correcting Codes

Error Correcting Codes (ECCs) are used to correct errors in memories. A SEC (Single Error Correcting) code is a kind of ECCs. Let SEC(k,q) code be a Hamming code (k+q bits) that consists of the information part (k bits) and the check part (q bits). Let $\vec{H}$ be a check matrix. Then, the SEC code $\vec{w}$ satisfies the relation $\vec{H}\vec{w}^T = 0$, where $\vec{w}$ consists of k+q bits. Let $\vec{k}$ be the information part, and $\vec{G}$ be the generator matrix that satisfies $\vec{H}\vec{G}^T = 0$. The SEC code is obtained from the relation $\vec{w} = \vec{k}\vec{G}$. By checking the **syndrome** $\vec{s}$, we can detect a single-bit error, and specify the position of the error bit. For the SEC code, since the minimum weight of the row vectors in the generator matrix $\vec{G}$ is three, it can perform single error correction. Table 2 shows the numbers of bits in the information part and the check part.

Fig. 7 illustrates a single-error correcting circuit for the memory using the SEC code. To correct a single-bit error, the syndrome generator first computes the syndrome from the inputs. This part is implemented by EXOR gates. Then, it checks whether the syndrome is zero vector or not. The error-position finder specifies the position of the error from the non-zero syndrome vector.

Soft errors in memories are not discovered until they are read by the LUT cascade emulator. Furthermore, the error correcting circuit masks the soft error of memory outputs, while soft errors remain in the memory. If an another soft error occurs in the word that contains a soft error, then the error correcting circuit cannot correct two soft errors. Thus, to avoid such vulnerability, we must restore the words that contains a soft error before the next soft error occurs. In this paper, we show a technique that *scans* the memory periodically to discover the latent soft errors. When a soft error is discovered, our method *rewrites* the correct data into the word that contains the soft error.

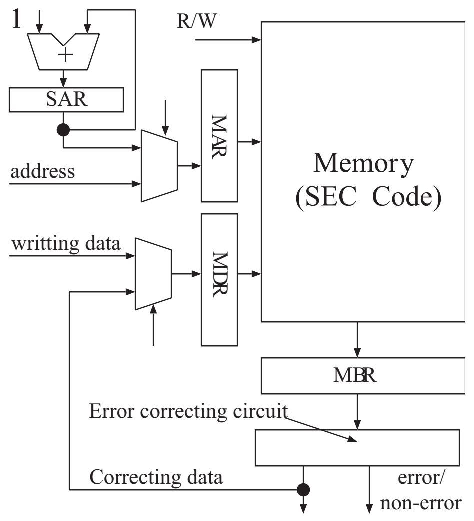


Figure 8. Memory with a scan circuit.

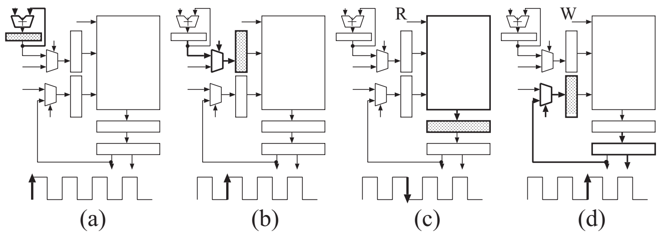


Figure 9. Scanning a word.

4.4 Correction of Latent Soft Errors

Fig. 8 illustrates a memory with a scan circuit: The **MAR (Memory Address Register)** retains the address of memory; the **MBR (Memory Buffer Register)** retains the output of memory; the **MDR (Memory Data Register)** retains the write data of memory; and the **SAR (Scan Address Register)** retains the address for scanning words. The error correcting circuit was described in Section. 4.3. In the memory for logic, the memory for interconnection, and memories for shifters, data are encoded in the SEC codes, and the scan circuits are attached.

Example 4.2 *Fig. 9 illustrates the operations of scanning a word. While the LUT cascade emulator is distributing the cell outputs to registers, it increments the address of the scanning word (Fig. 9(a)). At the rising edge of the clock, it sends the address of the scanning word to the MAR (Fig. 9(b)). At the next falling edge of the* *clock, it reads the memory, and contents are sent to the MBR (Fig. 9(c)). Then, the error correcting circuit checks whether an error exists or not. When an error exists, it sends the correct data to the MDR (Fig. 9(d)). By rewriting, the correct data is sent to the memory, and the soft error in the memory is corrected. (End of Example)*

The proposed method does not induce any overhead time, since it scans and rewrites memories when the LUT cascade emulator is idle[2]. The hardware overhead for each memory is the scan circuit. It consists of multiplexers to select data for the MAR and the MDR, a scan address generator (one adder and the SAR), a control circuit for scan and write-backing, and control signals. Since we use five memory units (i.e., the memory for logic, the memory for interconnection, and three memories for shifters), total hardware overhead is five scan circuits for memories.

5 Evaluation of Reliability

In this section, we evaluate the reliability of the proposed system in the LUT cascade emulator. We assume that the soft error rate for memory cells and flip-flops are the same.

5.1 Analysis of Reliability

The **reliability** at time t is the probability of proper operation at time t. Let $R_t(t)$ be the reliability of the circuit for a 1-bit memory cell or a flip-flop. Then, it can be written as

$$R_t(t) = e^{-\lambda t}. \tag{3}$$

Let W be the number of words of the memory, and m be the number of bits in a word, and $R_{NonSEC}(t,m,W)$ be the reliability of the memory without the ECC. Then, we have

$$\begin{aligned} R_{NonSEC}(t,m,W) &= \prod_{W}^{i=1}(\prod_{m}^{j=1} R_t(t)) \\ &= e^{-Wm\lambda t}. \end{aligned} \tag{4}$$

Let $R_W(t,m)$ be the reliability for one word of the memory. Then, we have

$$R_W(t,m) = e^{-m\lambda t} + m(1-e^{-m\lambda t})e^{-(m-1)\lambda t}, \tag{5}$$

where the first term denotes the probability that no soft error occurs in a word, and second term denotes the probability that only one soft error occurs in a word. Let $R_{SEC}(t,m,W)$ be the reliability of the memory with the ECC. Then, we have

$$\begin{aligned} R_{SEC}(t,m,W) &= [R_W(t,m)]^W \\ &= [me^{(1-m)\lambda t} - (m-1)e^{-m\lambda t}]^W \end{aligned} \tag{6}$$

[2]This is true only when the rewriting latency of a memory is one clock. If the write-back of a memory needs several clocks, we must suspend the operation.

Table 3 shows the number of bits in a word and the number of words of the memory. In Table 3, m denotes the number of outputs of the memory for logic; W denotes the number of words of the memory for logic; $\#In$ denotes the number of bits of the input register; $\#Out$ denotes the number of bits of the output register; s denotes the number of bits in the state register; r denotes the maximum number of bits of the rails; and c denotes the maximum number of cells. Let $R_{SEC\_MEM\_1}(t)$ be the reliability for five memories (i.e., the memory for logic with the SEC code and the other four memories without the support for soft errors) of the LUT cascade emulator; $R_{SEC\_MEM\_2}(t)$ be the reliability for five memories (the memory for logic without the support for soft errors and the other four memories with the SEC code) of the LUT cascade emulator; $R_{SEC\_MEM\_3}(t)$ be the reliability for five memories with the SEC code of the LUT cascade emulator; and $R_{NonSEC\_MEM}(t)$ be the reliability for five memories without any support for soft errors. Then, we have the following:

$$\begin{aligned} R_{SEC\_MEM\_1}(t) &= R_{SEC}(t,m,W) \\ &\times R_{NonSEC}(t, \lceil\log_2(\#In+s)\rceil + r, c) \\ &\times R_{NonSEC}(t, \lceil\log_2 s\rceil, c) \\ &\times R_{NonSEC}(t, \lceil\log_2 r\rceil, c) \\ &\times R_{NonSEC}(t, \lceil\log_2 \#Out\rceil, c) \end{aligned} \tag{7}$$

$$\begin{aligned} R_{SEC\_MEM\_2}(t) &= R_{NonSEC}(t,m,W) \\ &\times R_{SEC}(t, \lceil\log_2(\#In+s)\rceil + r, c) \\ &\times R_{SEC}(t, \lceil\log_2 s\rceil, c) \\ &\times R_{SEC}(t, \lceil\log_2 r\rceil, c) \\ &\times R_{SEC}(t, \lceil\log_2 \#Out\rceil, c) \end{aligned} \tag{8}$$

$$\begin{aligned} R_{SEC\_MEM\_3}(t) &= R_{SEC}(t,m,W) \\ &\times R_{SEC}(t, \lceil\log_2(\#In+s)\rceil + r, c) \\ &\times R_{SEC}(t, \lceil\log_2 s\rceil, c) \\ &\times R_{SEC}(t, \lceil\log_2 r\rceil, c) \\ &\times R_{SEC}(t, \lceil\log_2 \#Out\rceil, c) \end{aligned} \tag{9}$$

$$\begin{aligned} R_{NonSEC\_MEM}(t) &= R_{NonSEC}(t,m,W) \\ &\times R_{NonSEC}(t, \lceil\log_2(\#In+s)\rceil + r, c) \\ &\times R_{NonSEC}(t, \lceil\log_2 s\rceil, c) \\ &\times R_{NonSEC}(t, \lceil\log_2 r\rceil, c) \\ &\times R_{NonSEC}(t, \lceil\log_2 \#Out\rceil, c) \end{aligned} \tag{10}$$

Let $R_{NonTMR}(t)$ be the reliability for the p-bit register realized by normal flip-flops. Then, we have

$$\begin{aligned} R_{NonTMR}(t) &= [R_t(t)]^p \\ &= e^{-p\lambda t}. \end{aligned} \tag{11}$$

Let $R_{FF}(t)$ be the reliability for a TMR flip-flop. Then, we have

$$R_{FF}(t) = R_t(t)^3 + 3R_t(t)^2(1-R_t(t)), \tag{12}$$

Table 3. Numbers of bits for five memories.

Memory type	#bits in a word	#words
Memory for Logic	m	W
Memory for Interconnection	$\lceil \log_2(\#In + s) \rceil + r$	c
Memory for State Shifter	$\lceil \log_2 s \rceil$	c
Memory for Rail Shifter	$\lceil \log_2 r \rceil$	c
Memory for Output Shifter	$\lceil \log_2 \#Out \rceil$	c

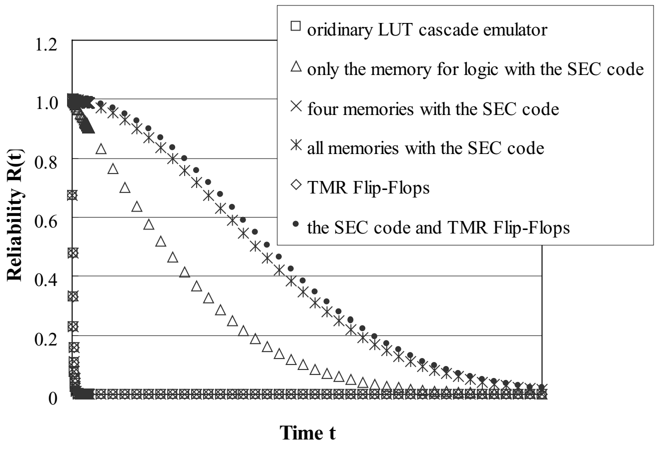


Figure 10. Reliabilities for six methods.

where the first term denotes the probability that no soft error occurs in three flop-flops, and the second term denotes the probability that only one soft error occurs in a flip-flop. Let $R_{TMR}(t)$ be the reliability for a p-bit register. Then, we have

$$\begin{aligned} R_{TMR}(t) &= [R_{FF}(t)]^p \\ &= [3e^{-2\lambda t} - 2e^{-3\lambda t}]^p. \end{aligned} \qquad (13)$$

Fig. 10 compares reliabilities for six methods: Without the support for any soft errors $(R_1(t) = R_{NonSEC\_MEM}(t)R_{NonTMR}(t))$; only support for soft errors in the memory for logic $(R_2(t) = R_{SEC\_MEM\_1}(t)R_{NonTMR}(t))$; without support for soft errors in the memory for logic but the support for other four memories $(R_3(t) = R_{SEC\_MEM\_2}(t)R_{NonTMR}(t))$; with support for soft errors for all memories $(R_4(t) = R_{SEC\_MEM\_3}(t)R_{NonTMR}(t))$; with support for soft errors by TMR flip-flops $(R_5(t) = R_{NonSEC\_MEM}(t)R_{TMR}(t))$; and with support for soft errors for all memories and by TMR flip-flops $(R_6(t) = R_{SEC\_MEM}(t)R_{TMR}(t))$.

To compare these methods, we define the **Mission Time Improvement Degree (MTID)**. Let T_R be the time when the reliability of the system considering soft errors falls to 99%, and T_{NonR} be the time when the reliability of the system without considering soft errors falls to 99 %. Then, we have the following:

$$MTID = \frac{T_R}{T_{NonR}}. \qquad (14)$$

Table 4. MTIDs for six methods.

	Method	MTID
R_1	Ordinary LUT cascade emulator	1.0000
R_2	Only the memory for logic with SEC code	204.6196
R_3	Four memories (exclude the memory for logic) with SEC code	1.0053
R_4	All memories with SEC code	1540.7089
R_5	With TMR flip-flops	1.0004
R_6	With SEC code and TMR flip-flops	2608.9992

Table 4 compares MTIDs for six methods. From Table 4, when the SEC code is used only in the memory for logic, the MTID is 204.6196. When the SEC code is used in all memories, the MTID is 1540.7089. However, when the SEC code is used in four memories except for the memory for logic, the MTID is 1.0053. Since the reliability for memory for logic is dominant in the LUT cascade emulator, the improvement of the reliability only for four small memories hardly improves the total reliability. When TMR flip-flops are used in the registers but no SEC codes are used for memories, the MTID is almost the same as an ordinary LUT cascade emulator. Since the reliability for memories is dominant in the LUT cascade emulator, the improvement of the reliability only for flip-flops hardly improves the total reliability. On the other hand, when both the SEC code and TMR flip-flops are used, the MTID is 1.69 times larger than the case where only the SEC codes are used in five memories. When the reliability for memories is improved, the MTID greatly depends on the reliability of flip-flops. Thus, the improvement for the reliability of flip-flops considerably improves the MTID.

5.2 Memory Scan and Rewrite Time

Let the MTBF (Mean Time Between Failure) be the average time between two consecutive single-bit soft errors occur in a word. Then, we have [13],

$$MTBF = \int_0^\infty R_{SEC}(t, m, W)dt, \qquad (15)$$

where $R_{SEC}(t, m, W)$ is given in (6). Let $Time_{SC}$ be the scan-rewrite time that is the maximum time to restores a soft error occurred in the memory. To restore from a single-bit soft error by the scan circuit, after the first soft error, the scan and rewrite must be finished before the second soft error occurs. Thus, we have the relation:

$$Time_{SC} \leq MTBF. \qquad (16)$$

Let $Cell_{max}$ be the maximum number of cells in the LUT cascade emulator, $Clock$ be the clock frequency (Hz), M be the amount of memories (bits), and m be the number of bits for each word. As shown Fig.4, our LUT cascade emulator uses two clocks for evaluating a cell, and two clocks for setting the input registers, the output registers, and performing the state transition. Thus, total number of clocks for evaluating all cells is $2Cell_{max} + 2$. The number of words for a memory is $\frac{M}{m}$. Then, we have the relation:

$$Time_{SC} \quad = \quad (2Cell_{max} + 2) \times \frac{1}{Clock} \times \frac{M}{m}. \quad (17)$$

We designed a prototype LUT cascade emulator on an FPGA (Altera Stratix EP1S60F1020C5). When $Cell_{max} = 128$, $M = 32k \times 32\ (bits)$, and $m = 32\ (bits)$, we obtained the value $Clock = 40 \times 10^6\ (Hz)$. From the prototype, we confirmed that $Time_{SC}$ is at most 0.2113 (sec). Let 1000 FIT/Megabit be a soft error rate (neutron + alpha) for SRAMs[3][15]. Note that, n-*FIT* denotes that n failures occur in a system per 10^9 operating hours. Thus, we have $\lambda = 1.0 \times 10^{-12} error/bit \cdot hour$. Then, we obtained $MTBF = 0.2204 \times 10^{10}\ (sec)$. We can see that the proposed method can mask soft errors in memories. If we implement the LUT cascade emulator by an ASIC, then the proposed method also masks the errors since it operates at higher speed.

6 Conclusion

This paper shows a soft error tolerant LUT cascade emulator. The cell data in the memories are encoded by error correcting codes. Also, error-correcting circuits are appended. Scanning circuits periodically check the memories. When they detect soft errors, they remove the error by write-backing the correct data into the memory. To mask the soft error in flip-flops, a TMR (Triple Module Redundancy) technique is applied. Our method detects a single-bit soft error in a word. Also, the mission time of our method is more than 1000x of an ordinary LUT cascade emulator.

7 Acknowledgment

This research is partly supported by Japan Society for the Promotion of Science (JSPS), MEXT, and Kitakyushu Innovative Cluster. Discussions with Prof. Yukihiro Iguchi and Hideo Ito were quite useful. Discussions with Prof. Jon T. Butler improved English presentation.

References

[1] A. Iseno, Y. Iguchi, and T. Sasao, "Fault diagnosis for RAMs using Walsh spectrum," *IEICE Trans. Information and Systems*, Vol. E87-D, No.3, March 2004, pp. 592-600.

[2] A. Maheshwari, I. Koren, and W. Burleson, "Techniques for transient fault sensitivity analysis and reduction in VLSI circuits," *IEEE Intl. Symp. on Defect and Fault Tolerance in VLSI Systems*, 2003, pp. 597-604.

[3] C. Metra, M. Favalli, and B. Ricc'o, "Novel Berger code checker," *IEEE Intl. Symp. on Defect and Fault Tolerance in VLSI Systems*, 1995, pp. 287-295.

[4] C. V. Frieman, "Optimal error detection codes for completely asymmetric binary channels," *Inform. Contr.*, Vol. 5, March 1962, pp. 64-71.

[5] G. H. Asadi, and M. B. Tahoori, "Soft error mitigation for SRAM-based FPGAs," *IEEE VLSI Test Symp.*, 2005, pp.207-212.

[6] H. Nakahara, T. Sasao, and M. Matsuura, "A design algorithm for sequential circuits using LUT rings," *IEICE Trans. Fundamentals of Electronics*, Vol. E88-A, No.12, Dec. 2005, pp. 3342-3350.

[7] H. T. Nguyen, and Y. Yagil, "A systematic approach to SER estimation and solutions," *Proc. of the 41st Intl. Reliability Physical Symp.*, Dallas, Texas, 2003, pp. 60-70.

[8] J. M. Rabaey, "Design at the end of the silicon roadmap," *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, Shanghai, Jan., 2005, Volume 1, pp.18-21.

[9] J. T. Chen, J. Rajski, J. Khare, O. Kebichi, and W. Maly, "Enabling embedded memory diagnosis via test response compression," *IEEE International Test Conference*, 2002, pp.292-298.

[10] M. Nicolaidis, "Time redundancy-based soft-error tolerance to rescue nanometer technologies," *Proc. IEEE VLSI TEST Symp.*, 1999, pp. 86-94.

[11] M. Omana, "Novel transient fault hardened static latch," *IEEE Intl. Test Conf.*, 2003, pp. 88-892.

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

[13] P. K. Lala, *Fault Tolerant and Fault Testable Hardware Design,* Prentice-Hall International, New York, 1985.

[14] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," *Proc. of the Intl. Conf. on Dependable Systems and Networks (DSN'02)*, Washington D. C., June 2002.

[15] S. Mitra, N. Seifert, M. Zhang, Q. Sbi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *IEEE Compt.*, Vol. 38, No. 2, , Feb. 2005, pp. 43-52.

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

[17] T. Sasao, M. Kusano, and M. Matsuura, "Optimization methods in look-up table rings," *IWLS-2004*, June 2-4, 2004, Temecula, California, USA, pp.431-437.

[18] T. Sasao, and M. Matsuura, "A method to decompose multiple-output logic functions," *Proc. Design Automation Conference (DAC)*, San Diego, CA, USA, June 2-6, 2004, pp.428-433.

[19] T. Sasao, Y. Iguchi, and M. Matsuura, "LUT cascades and emulators for realizations of logic functions," *RM2005*, Tokyo, Japan, Sept. 5-6, 2005, pp.63-70.

[3] We can also implement the memory for logic by DRAMs.