2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines

Improving the Robustness of a Softcore Processor against SEUs by using TMR and Partial Reconfiguration

Yoshihiro Ichinomiya*, Shiro Tanoue*, Motoki Amagasaki†, Masahiro Iida†, Morihiro Kuga† and Toshinori Sueyoshi†
*†*Graduate School of Science and Technology*
Kumamoto University
2-39-1 Kurokami, Kumamoto 860-8555, Japan
Email:{ichinomiya, tanoue}@arch.cs.kumamoto-u.ac.jp*
†{amagasaki, iida, kuga, sueyoshi}@cs.kumamoto-u.ac.jp*

***Abstract*—SRAM-based field programmable gate arrays (FPGAs) are vulnerable to a single event upset (SEU), which is induced by radiation effect. This paper presents a technique for ensuring reliable softcore processor implementation on SRAM-based FPGAs. Although an FPGA is susceptible to SEUs, these faults can be corrected as a result of its reconfigurability. We propose techniques for SEU mitigation and recovery of a softcore processor using triple modular redundancy (TMR) and partial reconfiguration (PR) with state synchronization. By carrying out an experiment, we confirm that a faulty softcore processor can be recovered and synchronized with other softcore processors. The proposed technique requires 4.315 times the resource usage and 62.491% of the operating frequency of the base processor. However, the proposed recovery process only takes 6 μs under TMR and PR. As a result of reliability estimation, the proposed system achieved about 2.713 times longer MTBF comparing with the previous system.**

***Keywords*-SEU, TMR, Partial Reconfiguration, Softcore Processor, ECC**

I. Introduction

Field programmable gate arrays (FPGAs) are widely used in numerous applications such as industrial electronic devices and embedded systems. However, an SRAM-based FPGA (hereinafter called "FPGA") is vulnerable to radiation effects known as single event effects (SEE)[1]. In particular, single event upset (SEU) which is one of the SEE impacts is a critical issue for FPGA. There are two reasons: SEU causes a bit flipping in memory elements, and FPGA consists of a large number of SRAM cells. These SRAM cells store configuration data so that SEU can seriously affect the performance of FPGA. The SEU is particularly significant in space because of the presence of very high doses of radiation. Moreover, the feature size of integrated circuits has reached the nano scale, and transistors are more sensitive to SEUs because of low threshold voltage. Therefore, we also consider the effects of SEU at the ground level. SEU is serious problems in reliable systems, such as automotive, infrastructure systems, etc..

The SEU sensitivity of the configuration memory can be mitigated by using a hardware redundancy technique such as triple modular redundancy (TMR)[1]. Previous studies have proposed various TMR schemes with reconfiguration techniques. In these studies, reconfiguration time can be hidden by using partial reconfiguration (PR) to recover an SEU fault[2]. However, these techniques are effective only for combinational circuits because sequential circuits involve a state condition. While one of the redundant modules is reconfigured, the other modules carry out their operations continuously. As a result, their states become unequally after PR. Pilotto[3] examined synchronization process after partial reconfiguration, which is effective in the finite state machine (FSM) but has investigated in a simple circuit only. Currently commercial FPGAs can be implemented large-scale circuit such as a softcore processor system. Therefore, we consider that SEU mitigation and recovery techniques are crucial for large-scale sequential circuits.

This paper presents SEU mitigation and recovery techniques for a softcore processor using TMR and PR. Although an FPGA is susceptible to SEUs, these faults can be corrected by using its reconfigurability. In this paper, to synchronize triplicated softcore processors, we design a memory sharing TMR which include an error detection circuit known as a detector. The memory reliability is also ensured by using an error correcting code (ECC). Therefore SEU is corrected by a novel synchronization technique, and this process performs on the fly.

The remainder of the present paper is organized as follows. Section II describes the effects of SEU and related research on the techniques to improve the robustness of FPGAs. Sections III and IV describe the proposed system and the synchronization technique, respectively. Section V describes the SEU recovery process and its verification. Section VI describes the evaluation of the proposed system and recovery processing. Section VII describes the estimation of the robustness in proposed system. Finally, the conclusions are presented in Section VIII.

II. Related Research

A. Effect of single event upset on FPGAs

An SEU is a soft error that is caused by radiation sources such as alpha particles or neutrons. If these radiation noises

978-0-7695-4056-6/10 $26.00 © 2010 IEEE
DOI 10.1109/FCCM.2010.16

exceed the critical charge of a memory cell or a flip-flop (FF), the noise result in an SEU. The SEU occurs once a week in the space environment or once in several months at the ground level.

SEUs are classified into "transient errors" and "permanent errors"[4]. A transient error is an error that occurs on an FF or a latch. Therefore, transient errors only instantaneously affect the output. On the other hand, a permanent error is an error that occurs on configuration memory or storage memory. A permanent error may cause faulty operation of the FPGA. Permanent errors are more crucial because an FPGA consists of a large number of configuration memory. Therefore, reliable implementation techniques that can mitigate and repair these SEU effects are required.

B. Previous Studies

Many studies on some of the dependable implementation techniques such as time redundancy, dual modular redundancy (DMR) and triple modular redundancy (TMR) have been conducted[1]. However, time redundancy cannot mitigate the permanent error effect. DMR cannot identify SEU affected module. Hence, full reconfiguration is required in order to eliminate a fault in time redundancy and DMR. This is undesirable for the system by which continuous operation is required, because system has been stopped and initialized by full reconfiguration. Consequently, we focus on the TMR implementation technique and partial reconfiguration (PR).

The basic concept of the TMR scheme is that the robustness of a circuit against SEUs. The TMR scheme can recover a transient error and mitigate a permanent error. However, SEUs must be corrected to prevent accumulation because multiple SEUs cannot be mitigated by the TMR scheme alone. In order to eliminate an SEU from the configuration memory, TMR is often coupled with reconfiguration techniques such as full reconfiguration, partial reconfiguration, or scrubbing. Most of the previous studies have used TMR with dynamic partial reconfiguration[2]. This helps in hiding the configuration time and is effective to a combinational circuits. Scrubbing is a technique that wherein the entire content of the configuration memory is reloaded periodically. However, these techniques cannot be applied to a sequential circuit because of its state information[5].

Although many studies focus on trade-off between TMR design and reconfiguration. However, it is not referred to state synchronization. Pilotto[3] considered synchronization by using a checkpoint state in the FSM. Once the reconfigured module reaches the checkpoint state, it is set on hold until the other two modules reach the checkpoint state. However, this method is not realistic for complicated systems. Some of studies have considered the implementation of a reliable softcore processor[6]. These studies only suggested a method of implementation, and they did not focus on recovery from SEU accumulation.

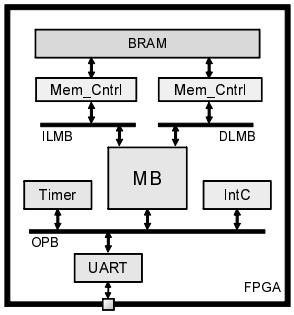

Figure 1. Base processor.

Table I
COMPONENTS OF THE BASE PROCESSOR.

Module	Used IP Core
Softcore Processor	MicroBlaze v6.00.b (MB)
Peripheral Bus	On-chip peripheral bus (OPB)
External Interface	Universal asynchronous receiver-transmitter (UART)
FPGA Embedded Memory	Block RAM (BRAM)
Memory Bus	Local memory bus (LMB)
Memory Controller	LMB BRAM interface controller (Mem_Cntrl)
Timer	OPB timer (Timer)
Interrupt Controller	OPB interrupt controller (IntC)

III. SYSTEM IMPLEMENTATION

A. Target system

Fig. 1 shows the target processor unit, which is called "base processor," and Table I shows the components of its architecture. In the present study, we use the Toyohashi Open Platform for Embedded Real-time Systems/Just Standard Profile (TOPPERS/JSP) kernel, which is an open-source real-time OS (RTOS) kernel [7]. This kernel is compliant with micro industrial the real-time operating system nucleus 4.0 (μITRON4.0). To operate the TOPPERS/JSP kernel, the system requires 64 KB of BRAM. In addition, two types of LMBs, namely, instruction LMB (ILMB) and data LMB (DLMB), are used, and a barrel shifter and divider are provided as an MB design option. MB area optimization is not enabled, which means MB is performance-optimized. In this study, the MB cache is not integrated into the base processor because the base processor assigns the BRAMs to the main memory and operates in synchronization with system clock. Furthermore, a translation lookaside buffer (TLB) is not implemented on MicroBlaze v6.00.b. The JSP kernel does not use a TLB, and therefore, the base processor does not use it.

B. Proposed TMR scheme

In the present paper, we investigate reliable softcore processor design using the TMR scheme [8]. Fig. 2 shows the proposed TMR softcore processor, which includes three redundant MBs, timers, and IntCs. The voter selects the correct signal. In this system, the BRAM is not triplicated because the BRAM can be handled by using a technique that

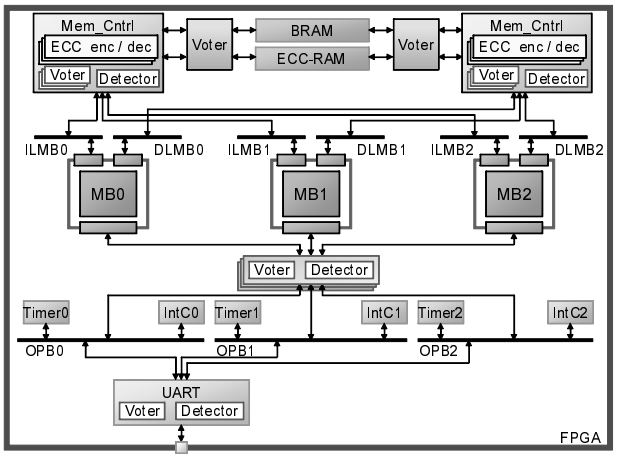

Figure 2. Proposed TMR softcore processor.

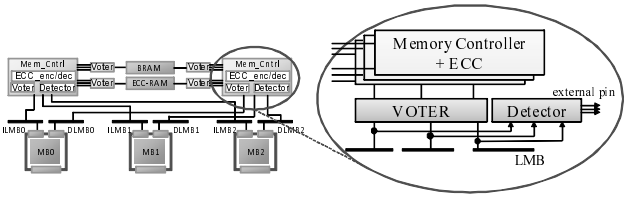

Figure 3. Memory Controller with voter and detector.

helps in improving the reliability of memory, such as the use of an error correcting code (ECC). In the present study, the fault recoverability of the softcore processor and the fault-tolerance capability of the system are discussed. Therefore, the partially reconfigurable region (PRR) is defined for MBs only. We achieve partial reconfiguration using the early access partial reconfiguration (EA PR) flow [9].

Fig. 3 shows a schematic diagram of the voter and the detector, which are integrated into Mem_Cntlr. The Mem_Cntlr outputs the ready signal to LMB when the MB accesses the memory. In order not to output the undesired ready signal, the inputs to Mem_Cntlr is voted. The detector is integrated into the voter, and it checks the outputs of the three modules. If one of the redundant modules includes an error, the detector detects the faulty module and outputs the error signal to external pins. Fig. 4 shows the circuit structure of the voter and detector. In the detector shown in this figure, R0, R1, and R2 are the outputs from the triplicated modules, and E0, E1, and E2 are error signals that indicate the module in which the error occurs. When the detector outputs the error signal, the faulty module is reconfigured. Voters and detectors are also integrated between MBs and the OPB and at a UART. The voter between the OPB and the MB stops the SEUs at MBs from affecting peripherals such as timers and IntCs. In order to prevent an output error, a voter is implemented at the UART. In the present study, only the effect of SEUs at MBs is considered.

Each softcore processor shares the BRAM in order to recover from the abnormal state and allow synchronization. The reason for this is described in greater detail in IV-B.

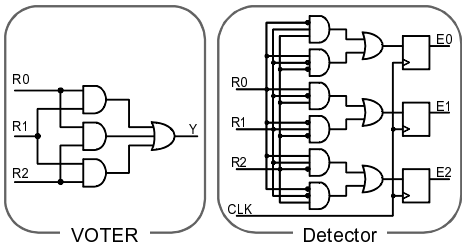

Figure 4. Voter and detector.

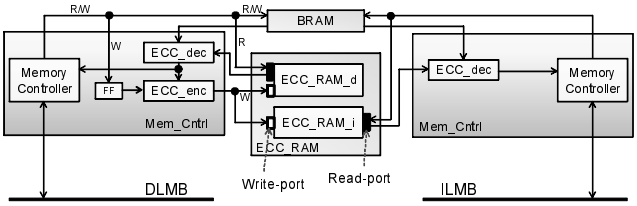

Figure 5. ECC implementation to BRAM

C. ECC implementation

As mentioned above, the BRAM is not triplicated, but its reliability is ensured by implementing the ECC. Fig. 5 shows the ECC-implemented BRAM structure. The ECC scheme used in the present study can handle single error correction and double error detection by using a Hamming code to encode the bit data. At this point, it does not handle the read-modify-write operation. As a result of the encoding process, a 7-bit code word is created; because the BRAM word width is 32 bits in our system. In order to store the code word, we add two 16 KB dual-port BRAMs, which have an 8-bit word size. We call them ECC_RAM. The ECC_RAM connected to the ILMB is represented as ECC_RAM_i, and the one connected to DLMB is represented as ECC_RAM_d. When we only describe BRAM, it means main memory.

In our ECC scheme, the BRAM is used in the "read first" mode. When the BRAM is used in the "write first" mode, the written data is output transparently. At this time, the ECC code word that corresponds to the output does not exist in ECC_RAM, and therefore, the decode process cannot take place. Additionally, MB can write the half-word and byte-unit data by controlling byte enable. If the bit error in BRAM is not overwritten with write data, the error data is inputted to the ECC encoder. As a result of encoding, both BRAM and ECC_RAM turn out to have the error data. The "write first" BRAM with ECC carries the risk of error propagation. Hence, we use the BRAM in the "read first" mode.

ECC_RAM initialization is performed after booting the operating system. It is because the BRAM initial entry is downloaded using configuration line. During this time, the ECC encoder does not operate. The ECC encoder operates only during the writing process from the MB to the BRAM. Therefore, ECC_RAM initialization is performed by repeating the load and store processes. The ECC decoder does not operate accurately unless the code word is created. In order to prevent faulty decode operation, we assign the unused

Table II
SPECIAL-PURPOSE REGISTERS ON MB

Special-purpose register	
register name	purpose
RPC	Program counter
RMSR	Machine status register
REAR	Exception address register
RESR	Exception status register
RFSR	Floating point status register
RBTR	Branch target register
RPVR0∼RPVR11	Processor version register

8th bit in ECC_RAM to check for the bit that can provide information on the creation of the code word.

The ECC_RAM is only written during the BRAM write process. Since the MB does not have a write port for the ILMB, the code word is written from the DLMB port to both ECC_RAM_i and ECC_RAM_d. Additionally, in order to match the code word to the BRAM contents, the encoder inputs must be equivalent to the BRAM contents after the writing operation. We determine the BRAM contents from "decoder output," "write data," and byte enable. If the read BRAM data has an error, the decoder can repair it. Therefore, the encoder input can be calculated accurately.

In the decode process, read access to BRAM and ECC_RAM are operated simultaneously, and this process takes one clock cycle. However, the write access to BRAM and ECC_RAM are not operated at the same time. Since the ECC encoder uses the decoder output, a lag of one cycle exists between the write access to ECC_RAM and the write access to BRAM. To prevent the access collision between the read and write processes of ECC_RAM, we assign the read and write ports, respectively, by using a dual port RAM.

IV. SYNCHRONIZATION TECHNIQUE

A. Recovery target registers

The effect of SEU on MB is mitigated by reconfiguration; however, reconfiguration alone is insufficient because the states of the MB become unequal after PR. To synchronize the MBs, we have to figure out the MB registers. The MB is a RISC softcore processor with Harvard architecture. The MB has 32 general-purpose registers and 18 special-purpose registers[10]. Table II shows the special-purpose registers in the MB. The number of special-purpose registers in use is decided by the MB settings. In this study, the MBs copy all the registers, except the processor version register (RPVR). RPVR is read-only register that can distinguish between multiple MBs in a multiprocessor system. The proposed system uses 3 MBs; however, they perform the same operation. Consequently, RPVR is negligible in the synchronization process.

B. Synchronization process

The synchronization process is described in detail in this paragraph. We assume that one of the MBs is reconfigured, and we call it "reconfigured MB." Firstly, the interrupt signal is inputted to all MBs, and the MBs start the synchronization process. Secondly, the MBs fetch the register synchronization instruction from a shared BRAM. However, the reconfigured MB cannot accept the interrupt signal because the reconfigured MB tries to execute a start-up process. To access the memory, the operation must proceed through the voter. Therefore, the voter rejects the memory access request from a reconfigured MB, and the reconfigured MB waits for the acknowledge signal. On the other hand, the voter accepts the requests from other MBs, which operate normally, and allows the instruction fetch operation. As a result, all MBs fetch the register synchronization instruction. Finally, all the MBs perform the register store and restore process to synchronize their inner states. At this time, the inner states of the reconfigured MB are not consistent with the other MB states. However, in the same way as instruction fetch, the register values of the reconfigured MB are rejected and the register values of the other MBs are accepted by the voter and stored in the BRAM. Subsequently, the stored values are written back to each of the registers of the MBs. As a result, the inner states of all the MBs are synchronized, and all MBs start to operate synchronously.

In the present study, we do not consider the case of the system that uses the cache or TLB. We assume that it can be handled by the following process. To handle the cache, we make it unusable and flush its contents. In the case of a write-back cache, the duty-line has to be written back to the main memory. If the determined cache content is incorrect because of the effect of an SEU, incorrect data is written to the main memory. In order to prevent such writing error, hardware redundancy technique is used. In the same way, it is assumed that TLB can be handled by making it unusable and resetting it during the synchronization process. These processes require a speed overhead because the cache and TLB are made unusable and are flushed/reset. However, this overhead can be ignored when considering the SEU occurrence rate.

V. SEU RECOVERY PROCESS

A. Recovery sequence

The proposed recovery mechanism incorporates TMR implementation, partial reconfiguration, and synchronization technique. Fig.6 shows the proposed recovery mechanism, and Table III shows the recovery procedure.

When an SEU occurs on one of the MBs, the effect of the SEU detected by the detector and mitigated by the voter. Then, the detector outputs an error signal to an external pin, and the faulty MB is reconfigured immediately. While the faulty MB is reconfigured, the other MBs continue to run. Therefore, the reconfiguration process is performed on the fly. After reconfiguration, the interrupt process of the RTOS triggers the synchronization process. This is why the registers of the reconfigured MB are different from those of the other MBs. The synchronization process is performed as mentioned in section IV-B. After these recovery processes,

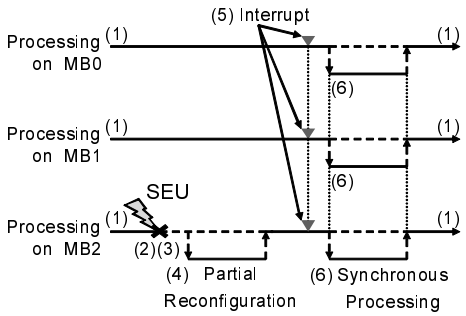

Figure 6. Recovery mechanism

Table III
RECOVERY PROCESS

(1)	The three MBs are operating normally
(2)	An SEU occurs at an MB
(3)	The voter mitigates the error, and the detector outputs the error signal
(4)	The faulty MB is reconfigured
(5)	The interrupt process of the RTOS triggers the synchronization process
(6)	Synchronization is performed

the SEU effect is removed from the faulty MB, and the registers of all the MBs are synchronized. As a result, the operation of all MBs are synchronized, and the operation returns to the normal state.

In the proposed recovery mechanism, system reconfiguration can be performed on the fly. In addition, the overhead of the synchronization process is only the time required to store and restore the registers. In the present study, the completion signal of the reconfiguration triggers the interrupt process. However, we also consider the use of the periodic interrupt. In this case, we assume that the cycle of periodic interrupt is set to be 10 times the frequency at which SEUs occur; this is because Xilinx suggested that the mitigation rate be at least 10 times the expected upset rate in order to minimize upset accumulation. Since SEUs occur less frequently, about once in several months on the ground level, the cycle of periodic interrupt is set longer.

B. Verification of recovery process

1) Verification process: We verified the recovery processing using a 16802A 68-channel portable logic analyzer (Agilent Technologies) and observed the error signal output at the detector. Table IV shows the verification procedure. In Step 1, we inject the fault by editing a bit corresponding to the MB in an original bitstream. In Steps 2, 4, and 6, the FPGA outputs the logical addition of the error signals of the detectors to the external pins. The signal is then checked using a logic analyzer.

2) Verification result: Fig. 7 shows the error signal of the detector after partial reconfiguration. As shown in Fig. 7, the error signal was output after reconfiguration because the values of the reconfigured MB registers differ from those of the registers of the other MBs. Therefore, the calculation results of MBs are not in agreement. After synchronization process, the error is corrected. The detector checks all output

Table IV
VERIFICATION PROCEDURE

(1)	Download the fault-injected bitstream onto the FPGA
(2)	Check the error signal of the detector
(3)	The faulty MB is reconfigured to correct the fault
(4)	Check the error signal after reconfiguration
(5)	Perform synchronization
(6)	Check whether the MBs are synchronized

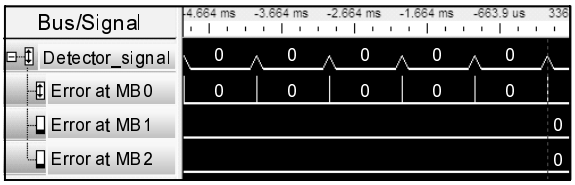

Figure 7. Error detection signal after partial reconfiguration

signals of the MB, but the detector does not indicate error occurrence. As a result, the proposed technique is found to have a beneficial effect on a softcore processor with regard to recovery from an SEU.

VI. EVALUATION

A. Evaluation environment

Table V shows the system implementation environment, design tools, and evaluation tools. We evaluate the system implementation results, such as resource usage, operating frequency, and recovery time of the proposed recovery technique. In order to evaluate the implementation result, we use a placement-and-routing (PAR) tool and a timing analyzer, which are the applications of integrated software environment 9.1.02i_PR2 (ISE 9.1.02i_PR2). Their input is the design file that is integrated on EA PR flow.

Firstly, in order to evaluate the resource usage, we use the PAR and create "placement and routing report". Secondly, in order to analyze the critical path delay, we use the timing analyzer. The entire operating frequency is calculated from this critical path delay. Finally, we evaluate the execution time of proposed recovery technique and compare it with the execution time for full reconfiguration. The time of the proposed recovery technique is evaluated by using the timer function of the TOPPERS/JSP kernel.

B. Implementation result

Table VI shows the system implementation result. In the table, "base" refers to the base processor structure, and "TMR" refers to the memory-shared TMR structure. "PR" and "ECC" refer to partial reconfiguration and ECC, respectively. The system with ECC requires 48 blocks of BRAM, and the one without ECC requires 32 blocks.

1) Resource Evaluation: "Base + ECC" requires more than 10% of the slices when compared to the "base processor," and "TMR + ECC" requires 3.05 times the slices when compared to "base + ECC." However, when compared to the "base processor" and "TMR," "TMR" requires 3.167

Table V
DEVELOPMENT ENVIRONMENT

Development Board	Xilinx ML410 Embedded development board
FPGA	Virtex-4 XC4VFX60FF1152-11
Design tools	Embedded development kit 9.1i(EDK9.1i)
	Integrated software environment (ISE) 9.1.02i_PR2
	PlanAhead 9.1.4
Evaluation tools	Placement and routing (PAR) 9.1.02i_PR
	Timing analyzer 9.1.02i_PR

Table VI
IMPLEMENTATION RESULT

	Slices	Critical path delay (ns)	Maximum operating frequency (MHz)
base processor	1,738	11.382	87.858
base + PR	2,266	13.627	73.384
base + ECC	1,919	16.742	59.730
base + PR + ECC	2,457	18.404	54.336
TMR	5,504	15.404	64.918
TMR + PR[8]	7,134	18.166	55.048
TMR + ECC	5,844	17.448	57.313
TMR + PR + ECC (Proposed system)	7,499	18.214	54.903

times the slices. The differences in the resource rates can be attributed to area optimization on synthesis flows. The inputs of the triplicated ECC_decoder are equivalent in the "TMR + ECC," because their inputs comprise of the BRAM output. Since all the ECC_encoders/decoders operate in a same manner and use the same input, the redundant ECC_encoder/decoder may be removed in optimize step. Further research in this area is thus required.

For applying the PR to "TMR" and "TMR + ECC," about 1,600 additional slices are required. This increase in resources is induced by the bus macro and the area optimization problem. The bus macro is the interface between the PRR and a fixed region. To define the PRR for each MB, we use 50 bus macros. Since each bus macro consists of 8 slices, a total of 400 slices are used for each MB. Thus, the total number of slices for the bus macros in the proposed system is 1,200. Area optimization was not accurate because the PRR was dedicated for partial reconfiguration. The same can be assumed for the base processor and "base + PR" too.

Owing to TMR implementation and partial reconfiguration, the proposed system requires 7,499 slices, which is about 3.907 times the number of slices in "base + ECC" and about 4.315 times the number of slices in the base processor. The proposed system can then be used on a medium-sized FPGAs such as XC4VFX60. We thus confirm that the proposed system can be put to practical use.

2) Evaluation of operating frequency: The evaluated systems satisfied a clock-net timing constraint of 20 ns, which corresponds to a frequency of 50 MHz. Table VI shows that the operating frequency of "base + ECC" decreased to about 67.985% of the operating frequency of the base processor. The critical path of "base + ECC" is from BRAM to ECC_RAM. Since this path goes through the ECC-decoder and ECC-encoder, the signals have to pass many XOR gates in order to determine the ECC code word. Moreover, it is difficult to optimize routing delay since both the critical path source and destination are hard macros. Consequently, the data path delay of "base + ECC" increases.

In the case of "TMR + PR," the operating frequency decreased to about 62.656% of the operating frequency of the base processor. In the EA PR design flow, the PRR is defined after designing the system framework. Therefore, the increase in delay due to TMR implementation is about 4.022 ns (15.404 - 11.382), and the delay increased by PR is about 2.762 ns (18.166 - 15.404). About 59.287% of the increases in delay is induced by TMR, and the remainder by PR. The critical path of "TMR + PR" is from the LMB_Cntlr (the ready signal for LMB) to the BRAM via the MB. In addition, the increase in delay by TMR and PR only affects the path between the MB and BRAM.

"TMR + PR + ECC" requires 62.491% of the operating frequency of the base processor. The difference in the operating frequency between "TMR + PR + ECC" and "TMR + PR" is small. It is because the increase in delay due to ECC implementation and that due to the implementation of "TMR + PR" do not affect each other. Consequently, the proposed system can increase the BRAM reliability with only a small performance degradation when compared to the "TMR + PR" scheme.

C. Evaluation of recovery time

We evaluate the performance of the fault recovery process in the case of the proposed system operating at 50 MHz. We compare the time required for the proposed recovery technique with that required for recovery by full reconfiguration.

The time required for recovery in the proposed system is equivalent to the synchronization process time, because the proposed system hides the time required for partial reconfiguration in the TMR scheme. We evaluate the synchronization process time using the system time reference function of the JSP kernel. The obtained synchronization process time was 6 μs, which indicates that the proposed technique can be adopted if a system allows a recovery time of 6 μs.

On the other hand, the time required for full reconfiguration depends heavily on the size of configuration data and data transfer rate. The FPGA used in this evaluation is Virtex-4 XC4VFX60-FF1152 having 2,625,439 bytes (= 21,003,512 bits) of configuration data, and the default transfer rate of Platform Cable USB is 6 Mbit/s. Consequently, the time required for full reconfiguration is 3,500.585 ms. In the case of minimum transfer rate (0.75 Mbit/s) or maximum transfer rate (24 Mbit/s) of Platform Cable USB, the time required is 28,004.683 ms and 875.146 ms, respectively. In order to restart the system from the point of start of full reconfiguration, register store/load time is required

Hence, the proposed recovery technique helps in reducing the recovery time when compared to full reconfiguration.

$$FIT/system = \frac{265}{1024^2}[FIT/bit] \times NC[bit] + \frac{427}{1024^2}[FIT/bit] \times NB[bit] \quad (1)$$

$$P(t,n)_{/target} = \left\{ \frac{2 \times X_t}{NT} \times \frac{(n-1)!}{p_1!p_2!p_3! \cdots p_t! \cdots p_i!} \times \frac{(X_1)^{p_1}(X_2)^{p_2}(X_3)^{p_3} \cdots (X_t)^{p_t} \cdots (X_i)^{p_i}}{(NT)^{n-1}} \right\} \times 3^l \quad (2)$$

$$FIT_{/TMR} = \sum_{k=2}^{N} \left\{ \frac{265}{k \times 1024^2} \times \sum_{t=1}^{i} \left\{ P(t,k)_{/target} \times X_t \right\} \right\} \quad (3)$$

$$P(n)_{/ECC} = TW \times (n-1)! \times \frac{(NW)^{n-1} \times (NW-1)}{(NB)^n} \quad (4)$$

Table VII
THE EXPLANATION OF THE PARAMETER

X_j	The number of the memories in module 'j' ($1 \leq j \leq i$)
i	The number of base modules before triplication. ($i \geq 1$) They include the bus macro and the buses such as OPB.
p_i	The number of the times of SEUs ($0 \leq p_i \leq n-2$, $\sum_{k=1}^{i} p_k = n-1$)
l	The number of the modules which is affected by SEUs ($1 \leq l \leq i$)
t	The selection number of the target module ($1 \leq t \leq i$, $1 \leq p_t \leq n-1$)
NT	The sum of configuration memories on triplicated modules
TW	The total number of the memory words
NW	The total word size of BRAM and ECC-RAM. In this estimation NW = 48

Table VIII
THE ESTIMATION TARGET SYSTEMS AND THEIR MEMORY USAGE

	Configuration memory (bit)	BRAM (bit)
base	1,143,569	524,288
base + ECC	1,262,663	786,432
TMR	3,621,522	524,288
TMR + ECC	3,845,234	786,432
TMR + ECC + PR	4,934,191	786,432

VII. ESTIMATION OF THE DEPENDABILITY

In this section, we estimate the failure in time (FIT) and the mean time between failure (MTBF) in order to demonstrate the effectivity of the proposed system. FIT means the number of the errors per billion hours of use. We refer to SEU information provided by Xilinx white paper[11] and user guide[12]. The FIT of the configuration memory and the BRAM in Virtex-4 are 265 FIT/Mbit and 427 FIT/Mbit, respectively. Table VIII shows the estimation target systems and the memory usage in their systems. The amount of configuration memory usage is calculated by multiplying the slice utilization ratio by total number of configuration memories in XC4VFX60.

In this estimation, we only consider the SEUs in the configuration memory and the BRAM. Equation (1) shows the FIT calculation formula for the base circuit which is normal processor without reliable technique. We assume that memory on FPGA consists of configuration memory and BRAM so that FIT of total system is defined following; Note that we transfer FIT/Mbit to FIT/bit. NC and NB: the total number of memories on configuration memory bit and BRAM. Table VII lists parameters in this evaluation.

Fig.8 shows the example of the SEU occurrence in the

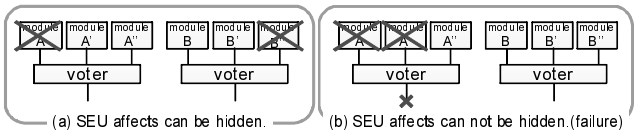

Figure 8. Example of the SEU occurrence in TMR scheme

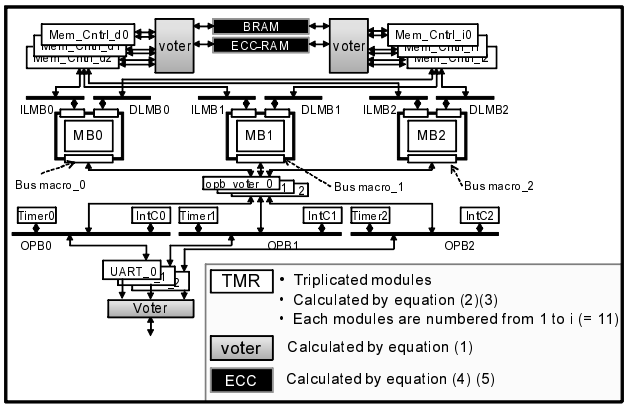

Figure 9. Classification of the proposed system for FIT calculation

TMR scheme. If the SEUs occur on the one of triplicated circuit such as diagram (a), the SEUs are mitigated by TMR scheme. However, the SEUs occur on the more than two of the triplicated circuit such as diagram (b), the SEUs induce the failure. In order to calculate the FIT of the triplicated circuit, we propose the equations (2) and (3). Equation(2) calculates the probability which causes failure in target module for n times SEU occurrence. This means that target module can mitigate until n-1 times SEU occurrence. The module combination influenced by SEU is 3^l. In the equation (3), the failure probability of the each modules are summed, and the FIT of total system is calculated. Fig. 9 shows the classification of the modules for FIT calculation. In this system, the number of triplicated modules are 33, and the number of base modules which means that i = 11. The base modules are numbered from 1 to i, and they correspond to $X_j (1 \leq j \leq i)$.

Equations (4) and (5) are FIT calculation formula for the ECC applied BRAM.

$$FIT_{/ECC} = \sum_{k=2}^{N} \frac{427 \times NW}{k \times 1024^2} \times P(k)_{/ECC} \quad (5)$$

The basic concepts of these equations are the same as

Table IX
ESTIMATION RESULT OF SYSTEM DEPENDABILITY (N=20)

	FIT	MTBF (hour)	MTBF (year)
base	502.506	1,990,022	227
base + ECC	319.105	3,133,766	358
TMR	286.502	3,490,375	398
TMR + ECC	68.776	14,539,936	1660
TMR + ECC + PR (Proposed system)	25.349	39,448,711	4503

equation (2) and (3). Firstly, the probability of the failure in a word is calculated. Next, the $FIT_{/ECC}$ and the $FIT_{/TMR}$ is calculated each 'n' value which vary from "n = 2" to "n = N". These results are summed. In this estimation, we assume that the value of 'N' is 20.

The FIT of the whole system is calculated by equation (6). $FIT_{/system}$ and $FIT_{/ECC}$ are calculated by equation (2)$\sim$(5), and $FIT_{/voter}$ is calculated by equation (1).

$$FIT_{/system} = FIT_{/TMR} + FIT_{/ECC} + FIT_{/voter} \quad (6)$$

$$MTBF = \frac{10^9}{FIT_{/system}} \quad (7)$$

The MTBF, which is the metrics of the dependability, is calculated by equation(7).

Table IX shows the result of the dependability estimation. Table IX shows that MTBF of the proposed system is about 2.713 times longer than one of the "TMR + ECC." In the case of comparing with "base", proposed system achieve about 19.837 times of MTBF. The MTBF of the Proposed system is much longer than "TMR + ECC." There are two reasons for the increasing. First, the SEU on the MB can be recovered by using partial reconfiguration (PR) in novel system. Second, the area of the MB is larger than other modules. Although the SEU probability of MB is high due to its area, the SEU effect on the MB is eliminated by PR. Additionally, the FIT of the other modules are very small comparing with one of the MB. As a result, the FIT of the proposed system becomes smaller.

VIII. CONCLUSION AND FUTURE WORK

In the present paper, we have presented a technique for ensuring reliable softcore processor implementation on FPGAs. This technique involves the use of TMR schemes coupled with dynamic partial reconfiguration, which mitigates the effects of SEUs on the configuration memory of the FPGA. In addition, we consider synchronization after partial reconfiguration by using an interrupt process. Owing to these schemes, the proposed system requires about 4.315 times the resources, and 62.491% of the maximum operating frequency of the base processor. However, the proposed system can recover a faulty softcore processor on the fly. Furthermore, the recovered MB becomes synchronized with the other softcore processors after a recovery time of 6 μs. In conclusion, a softcore processor can recover from an SEU by dynamic partial reconfiguration and synchronization. In order to demonstrate the effectivity of proposed system, we estimate the FIT and MTBF. As a result, the proposed system achieved about 2.713 times longer MTBF comparing with the "TMR+ECC."

In the present system, we cannot repair the fault on the processor peripheral circuits. In addition, the present ECC circuit does not have the read-modify-write function. In our future studies, we intend to focus on the fault recovery technique for BRAM and peripheral circuits. After that, we intend to study more accurate metrics of reliability and estimate the reliability of whole system.

REFERENCES

[1] F.L. Kastensmidt, L. Carro, and R. Reis, "Fault-Tolerance Techniques for SRAM-based FPGAs," Vishwani D. Agrawal Springer, Netherlands, 183p., 2006.

[2] C. Bolchini, A. Miele, and M.D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," Proc. of 22th IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 87–95, Sept. 2007.

[3] C. Pilotto, J. Rodrigo Azambuja, and F. Lima Kastensmidt, "Synchronizing Triple Modular Redundant Designs in Dynamic Partial Reconfiguration Applications," In Proc. on SBCCI'08, pp. 199–204, Sept. 2008.

[4] G. Asadi and M. B. Tahoori, "Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs," Proc. of the 13th. ACM Int. Symp. on FPGA2005, pp. 149–160, Feb. 2005.

[5] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, KA. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," IEEE Trans. on Nuclear Science 55(4 Part 1):2259-2266, 2008.

[6] V. Vasudevan, P. Waldeck, H. Mehta, and N. Bergmann, "Implementation of a triple modular redundant FPGA based safety critical system for reliable software execution," In Proc. on SCS'06, pp. 113–119, 2006.

[7] TOPPERS Project (2010), http://www.toppers.jp/en/index.html

[8] S. Tanoue, T. Ishida, Y. Ichinomiya, M. Amagasaki, M. Kuga and T. Sueyoshi, "A Novel States Recovery technique for the TMR Softcore Processor" Proc. of the 19th Int'l Conference on Field Programmable Logic and Applications (FPL'09), pp.543–546, Aug. 31 - Sept. 2, 2009.

[9] Xilinx, Inc. "Early Access Partial Reconfiguration User Guide," Xilinx User Guide : UG208 (v1.2), 2007.

[10] Xilinx, Inc. "MicroBlaze Processor Reference Guide," Xilinx User Guide : UG081 (v7.1), 2007.

[11] A. Lesea, "Continuing Experiments of Atmospheric Neutron Effects on Deep Submicron Integrated Circuits," Xilinx White Paper : WP286 (v1.0.1), 2009.

[12] Xilinx, Inc. "Device Reliabiligy Report," Xilinx User Guide : UG116 (v5.7), 2009.