A NOVEL STATES RECOVERY TECHNIQUE FOR THE TMR SOFTCORE PROCESSOR

Shiro Tanoue, Tomoyuki Ishida, Yoshihiro Ichinomiya, Motoki Amagasaki, Morihiro Kuga, and Toshinori Sueyoshi

Graduate School of Science and Technology, Kumamoto University
2-39-1 Kurokami, Kumamoto, 860-8555, Japan
e-mail: {tanoue,ishida,ichinomiya}@arch.cs.kumamoto-u.ac.jp,
{amagasaki,kuga,sueyoshi}@cs.kumamoto-u.ac.jp

ABSTRACT

The present paper describes a technique for ensuring reliable softcore processor implementation on SRAM-based Field Programmable Gate Arrays (FPGAs), which can handle the effects of Single Event Upsets (SEUs). We propose the Triple Modular Redundancy (TMR) scheme coupled with dynamic partial reconfiguration to remove SEUs from the configuration memory of the FPGA. Although the FPGA is subject to SEUs, these errors can be corrected as a result of its reconfigurability. Furthermore, we consider the synchronization after a partial reconfiguration using an interrupt process of an RTOS. Experimental results reveal that one faulty softcore processor is recovered and synchronized with the other softcore processors. The present study demonstrates that a softcore processor can recover from an SEU using the proposed dynamic partial reconfiguration and the synchronization process.

1. INTRODUCTION

SRAM-based Field Programmable Gate Arrays (FPGAs) are widely used in numerous applications, such as industrial electronic devices and embedded systems. However, FPGAs are vulnerable to radiation effects called Single Event Upsets (SEUs). If a SEU changes the information of a memory element, the functionality on the FPGA is not ensured. In addition, the feature size of integrated circuits has reached the nanoscale, and nanoscale transistors are become more sensitive to SEUs. Therefore, the SEU is particularly significant for the environment in space, but we also consider the effects of SEUs at the ground level. To cope with this problem, fault detection and correction techniques must be introduced.

The SEU sensitivity in the configuration memory can be mitigated by techniques such as Triple Modular Redundancy (TMR). Therefore, previous study have proposed TMR schemes coupled with dynamic partial reconfiguration [1] because adopting only the TMR scheme is not robust [2]. However, such schemes did not address the problem of synchronization after reconfiguration. Furthermore, the above techniques are effective for combinational circuits only, because sequential circuits include a state condition. For instance, if we use reconfiguration on a softcore processor, the states of the processor are initialized. Therefore, the objective of the present study is to correct for the effects of SEUs in the case of softcore processor design for an FPGA.

The dynamic partial reconfiguration and synchronization process allow for the correction of the effects of SEU. However, there is little information available on synchronization after partial reconfiguration. Pilotto [3] examined synchronization after partial reconfiguration, which is effective in the Finite State Machine (FSM) but has not been investigated in a complex circuit. Therefore, we consider synchronization after partial reconfiguration in a softcore processor.

The present paper describes the recovery technique on a softcore processor using dynamic partial reconfiguration and synchronization. Although the FPGA is subject to SEUs, these errors can be corrected due to its reconfigurability. The faulty processor is detected at the voter and is corrected through dynamic partial reconfiguration. Then, an interrupt process of an RTOS triggers the synchronization process. The synchronization process synchronizes the softcore processors through storing and restoring the registers of the correct processor. As a result, this technique can correct the effects of SEUs and enable continued operations of the softcore processors. Therefore, the proposed technique has a beneficial effect on the recovery from SEUs. The present study introduces a reliable softcore processor design with a recovery function.

The remainder of the present paper is organized as follows. Section 2 describes the proposed technique. Section 3 describes the implementation of the proposed system and the verification of the recovery processing. Finally, conclusions presented in Section 4.

978-1-4244-3892-1/09/$25.00 ©2009 IEEE

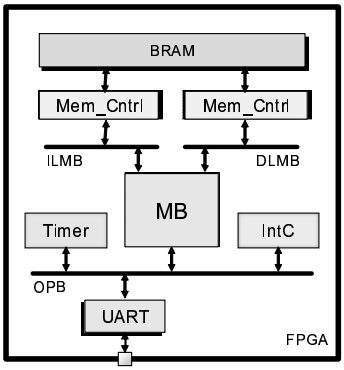

Fig. 1. Base processor.

Table 1. Components of the base processor.

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)

2. PROPOSED TECHNIQUE

2.1. TMR Softcore processor

In the present paper, we investigate reliable softcore processor design using the TMR scheme [4]. Fig. 1 shows the base processor, and Table 1 shows the components of its architecture. In the present study, we use the TOPPERS/JSP kernel, which is an open-source RTOS kernel [5]. This kernel is compliant with μITRON4.0 (Micro Industrial The Real-time Operating System Nucleus). To operate the RTOS, the system requires 64 KB of BRAM. In addition, two types of LMBs, namely, Instruction LMB (ILMB) and Data LMB (DLMB), are implemented.

Fig. 2 shows the proposed TMR softcore processor, which includes three redundant MBs, Timers, and IntCs. The voter selects the correct signal, and the detector checks the outputs of the three modules. If the detector outputs the error signal, then the faulty module is reconfigured. Voters and detectors are also implemented between MBs and the OPB, and at a UART. The voter between the OPB and the MB stops SEUs at MBs from affecting peripherals, such as timers and IntCs. In order not to output an error, a voter is implemented at the UART. In the present study, the dependability of the softcore processor is discussed. Therefore, the Partial Reconfiguration Region (PRR) is defined for MBs only. We achieve partial reconfiguration using the Early Access Partial Reconfiguration (EAPR) flow.

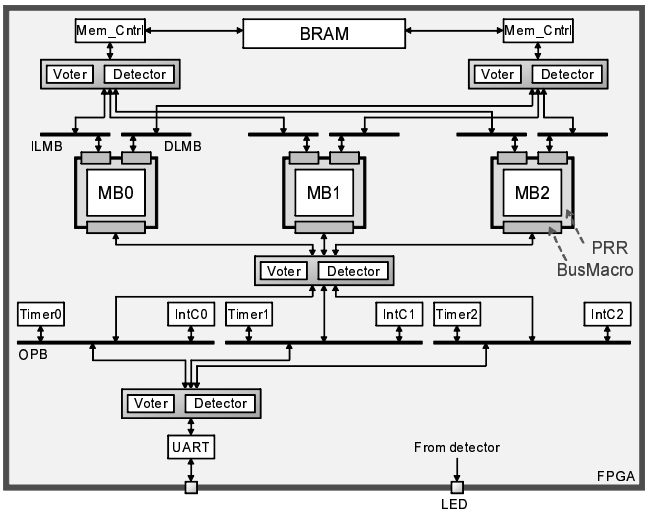

Fig. 2. Proposed TMR Softcore processor.

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 Subsection 2.2. In the present paper, the BRAM is not triplicated because the BRAM can be handled using a reliable memory technique, such as an error correcting code (ECC).

2.2. Recovery mechanism

Reconfiguration only is insufficient, because the state of the MB is initialized. We propose a recovery technique using partial reconfiguration because TMR cannot deal with two or more errors. Therefore, the MBs require a synchronization process in order to apply the TMR scheme.

The proposed recovery mechanism incorporates partial reconfiguration and synchronization. Fig. 3 shows the proposed recovery mechanism. The following describes the recovery procedure:

(1) The three MBs are operating normally

(2) An SEU occurs at an MB

(3) 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

First, if the detector outputs an error signal, the faulty MB is reconfigured immediately. Then, all MBs receive an interrupt signal, and the registers of correct MBs are copied to the registers of the reconfigured MB.

The synchronization is described in detail in the following. For instance, we assume that an SEU occurs on MB0. When an SEU is detected, MB0 is reconfigured. The MB

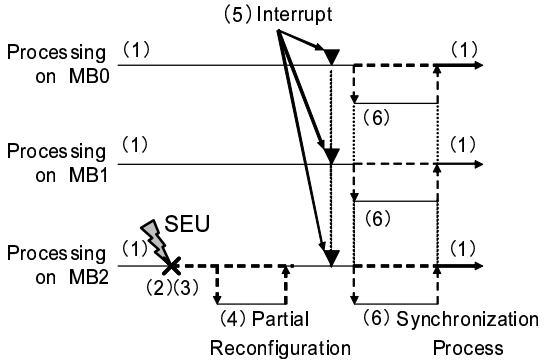

Fig. 3. Recovery mechanism.

has 32 general-purpose registers and 18 special-purpose registers. The MBs copy these registers except the Processor Version Register (PVR), which users cannot overwrite as the synchronization process. The operation of the register store process is loaded into all of the MBs. However, the reconfigured MB0 cannot accept this operation because MB0 tries to execute a start process. To access a memory, the operation must proceed through the voter. Therefore, the voter rejects the start process of MB0, and MB0 waits for the Acknowledge signal. On the other hand, the voter accepts the other MBs, which operate normally. As a result, the registers that MB operates normally are written in a shared memory. Then, the stored value is written back to each of the registers of the MBs, and all MBs are synchronized. This is the why the MBs share the BRAM.

An interrupt process of the RTOS triggers the synchronization process. While the faulty MB is recovered, the other MBs continue to run. Therefore, system recovery 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 because Xilinx suggested that the mitigation rate be at least ten times the expected upset rate, in order to minimize upset accumulation [6]. Since SEUs occur less frequently, the cycle of periodic interrupt will be set longer.

3. IMPLEMENTATION AND VERIFICATION

3.1. Implementation

Table 2 shows the implementation environment and the design tools used in the present study. We evaluate the area usage and the operating frequency. The evaluation results were referred to the report of Place&Routing using ISE9.1i_PR2. The PRR and the fixed region are implemented separately in the EAPR flow. In addition, the area usage of the entire

Table 2. Implementation environment.

development board	Xilnx ML410 embedded development board FPGA : Virtex-4 XC4VFX60FF1152
design tools	Embedded Development Kit 9.1i
	Integrated Software Environment 9.1.02i_PR2
	PlanAhead 9.1.4

Table 3. Area usage of the proposed system.

		Number of slices	Number of BRAMs
Base processor		1,738	32
Base processor (Partial)	static	788	32
	MB (PRR)	1,877	0
Proposed system †	static	4,330	32
	MB (PRR)	1,877	0

† Estimated values from our analysis

proposed system was not reported. Therefore, the PRR and the fixed region are evaluated separately. The area usage of the PRR for each MB is 3,143 slices, and the area usage of the fixed region is 3,064 slices. The total number of slices in the proposed system is 12,493 slices, which gives an area usage that is approximately seven times larger than that for the base processor. Next, we consider three factors that affect the results for the area usage because the area usage is very larger from the probable result.

The first factor is the bus macro. To define the PRR for each MB, we use approximately 50 bus macros. Since each bus macro consists of eight slices, a total of 400 slices were used for each MB. Then, the total number of slices for the bus macros in the proposed system is 1,200 slices.

The second factor is the voter and the detector. For example, the number of signals between the MB and the OPB is over 200. These signals are connected to the voter and the detector. This is why the area usage is too large.

The last factor is that the area usage of the PRR may be incorrect. From our analysis, we found that the area usage of the voter and the detector is considered to be double counting. Therefore, the PRR of the MB in the base processor is defined. Then, in the proposed system, we assume that the area usage of the base processor corresponds to the area usage of the MB. The revised area usages are shown in Table 3. In Table 3,"static" refers to the area usage of the fixed region. Then, considering the area usage of the PRR of the MB for the base processor (1,877 slices), we assume that the maximum combined area usage of the voter and the detector is 1,266 slices ($3,143 - 1,877$). This area usage and the area usage of the fixed region (3,064 slices) indicate that the area usage of "static" will be 4,330 slices (3,064 + 1,266). This result indicates that the area usage of the entire system is 9,961 slices, which is approximately six times larger than the area usage of the base processor. This occurs because

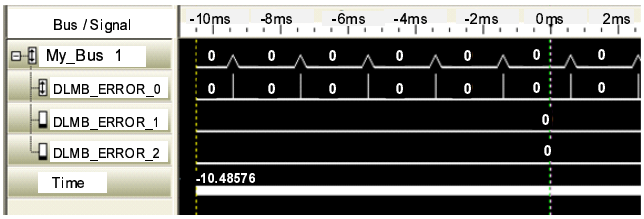

Fig. 4. Error detection after reconfiguration on the DLMB.

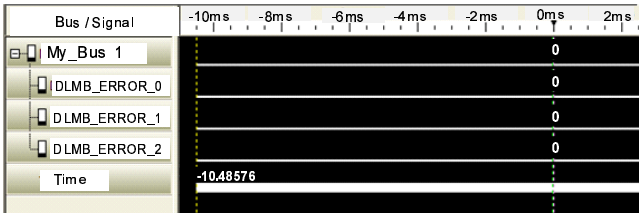

Fig. 5. No error detection after synchronization on the DLMB.

the area optimization was not performed well because the PRR was defined for MBs. The proposed system can then be implemented on a mid-sized FPGA, such as XC4VFX60.

The operating frequency of the proposed system is constrained to 50 MHz. We analyze the operating frequency of the entire proposed system using Timing Analyzer. As a result, maximum operating frequency is 52.695 MHz.

3.2. Verification

We verified the recovery processing using a 16802A 68-channel portable logic analyzer (Agilent Technologies) and observed the error signal output at the detector. The following describes the 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) Synchronization is performed

(5) Check whether MBs are synchronized

In Step 1, we inject the fault by editing a bit corresponding MB in an original bitstream. In Steps 2 and 5, the FPGA outputs the error signals of the detector to the external pins. The signal is then checked using a logic analyzer.

We verified the error detection on the LMB. On the ILMB, observation revealed that the fault was corrected after partial reconfiguration. Fig. 4 shows the error signal of the detector on the DLMB after partial reconfiguration, and Fig. 5 shows the signal after synchronization. As shown in Fig. 4, the error signal was output after reconfiguration on the DLMB because the register values of the reconfigured MB differ from those of the other MBs. Therefore, the load/store operation results are not in agreement. Then, Fig. 5 shows that the error is corrected by the synchronization process. As a result, the proposed technique is found to have a beneficial effect on recovery from an SEU.

Finally, we evaluate the performance of the synchronization process. Therefore, we compare the time required for the proposed recovery technique with that for recovery using a full reconfiguration. The FPGA used in this evaluation is a Virtex-4 XC4VFX60-FF1152 having 2,625,439 Bytes of configuration data, and the Platform Cable USB transfers 750 Byte/ms. Consequently, the time required for a full reconfiguration is 3,500.585 [ms]. On the other hand, 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. Then, we evaluate the synchronization process time using the timer function of the RTOS. The obtained synchronization process time was 8 [μs], which indicates that the proposed technique can be adopted if a system allows 8 [μs] for recovery.

4. CONCLUSION

In the present paper, we have presented a technique for ensuring reliable softcore processor implementation on Field Programmable Gate Arrays (FPGAs). This technique includes TMR schemes coupled with dynamic partial reconfiguration, which handles the effects of SEUs on the configuration memory of the FPGA. In addition, we consider synchronization after partial reconfiguration using an interrupt process. As a result, one faulty softcore processor is recovered on the fly. Furthermore, the recovered MB becomes synchronized with the other softcore processors after a recovery time of 8 μs. In conclusion, a softcore processor can recover from an SEU by dynamic partial reconfiguration and synchronization.

5. REFERENCES

[1] 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.

[2] L. Sterpone, M. Violante, "Analysis of the Robustness of the TMR Architecture in SRAM-Based FPGAs," IEEE Trans. on Nuclear Science, Vol. 52, No. 5, pp. 1545–1549, Oct. 2005.

[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] Y. Ichinomiya, S. Tanoue, T. Ishida, M. Amagasaki, M. Kuga, and T. Sueyoshi, "Memory Sharing Approach for TMR Softcore Processor," In Proc. on ARC'09, pp. 268–274, Mar. 2009.

[5] TOPPERS Project (2008), http://www.toppers.jp/jsp-kernel.html

[6] B. Bridgford, C. Carmichael, and C. Wei Tseng, "Single-Event Upset Mitigation Selection Guide," Xilinx Application Note : XAPP987 (v1.0), 2008.