A Hardware-Software Collaborated Method for Soft-Error Tolerant MPSoC

Weichen Liu, Jiang Xu, Xuan Wang, Yu Wang†, Wei Zhang‡, Yaoyao Ye, Xiaowen Wu, Mahdi Nikdast, Zhehui Wang
The Hong Kong University of Science and Technology, Hong Kong, China. E-mail: {weichen,eexu}@ust.hk
†Tsinghua University, Beijing, China. ‡Nanyang Technological University, Singapore.

***Abstract*—Multiprocessor systems-on-chip (MPSoCs) are attractive platforms for embedded applications with growing complexity, because integrating a system or a complex subsystem on a single chip provides better performance and energy efficiency and lower cost per function. As feature sizes and power supply voltages continually decrease, MPSoCs are becoming more susceptible to soft errors. However, traditional soft-error tolerant methods introduce large area, power and performance overheads to MPSoCs. This paper presents a low-overhead hardware-software collaborated method, called SENoC, to dynamically mitigate soft errors on MPSoCs using an on-chip sensor network. We developed a low-cost on-chip sensor network to collaboratively monitor and detect soft errors, and implemented software-based mechanisms to guarantee correct task executions. To maximize the performance of soft-error tolerant MPSoCs, a hybrid scheduling scheme is proposed to effectively manage applications and resources under uncertainties. We studied the new method on MPSoCs with different scales and tested it using typical embedded applications under different cosmic ray flux conditions. Experimental results show that comparing to traditional methods SENoC requires substantially lower protection overheads to achieve the same level of soft-error tolerance. For instance, soft-error tolerant MPSoCs using SENoC archive on average 114.1% better performance than a latest traditional method, and SENoC only introduces 0.42% area overhead to a 256-core MPSoCs.**

I. Introduction

As feature sizes continue to shrink with the advancement of nanotechnologies, multiprocessor systems-on-chip (MPSoCs) become promising solutions to satisfy the growing demands of embedded applications, because integrating a system or a complex subsystem on a single chip provides better performance and energy efficiency and lower cost per function [1]. On the other hand, nanotechnologies also make MPSoC more susceptible to various reliability threats, such as soft errors. Soft errors occur when high-energy particles or electromagnetic noises cause state inversions directly in registers or memory elements, or indirectly through glitches propagating across combinational logic [2]. Without proper protections, they can compromise system reliability, and cause unpleasant user experiences or even serious danger.

Soft-error tolerant system designs have been studied extensively in the past [3]. There are two types of methods to achieving system-level soft-error protections for multiprocessor systems. In hardware-based methods, additional circuits are designed in company with common functional units to provide error protections, e.g., Triple Modular Redundancy (TMR) [4], in which three identical subsystems are used to process the same task and a majority voting of the results is used to determine the correct outputs. Hardware-based methods can mitigate soft errors and do not introduce extra workload to software. However, they require significant hardware modifications and involve large overheads on area, power, and performance [5]. In software-based methods, instead of depending on extra hardware, additional copies of a program are executed in order to obtain error-resilient results [6]. Software-based methods minimize hardware changes but introduce large overheads on task execution time and power consumption, which make it hard to meet the stringent requirements of embedded applications.

Traditional soft-error tolerant methods dramatically increase MPSoC cost and reduce performance due to their large protection overheads. This significantly limits the scalability and flexibility of MPSoCs. The root of this problem is traditional methods rely on fixed protections against transient soft errors. Fixed protections are designed for worst cases, but worst cases only account for a small portion of the total MPSoC run time, and this makes traditional methods become burdens for MPSoCs in most of the time.

This paper presents a novel low-overhead dynamic method, called SENoC, for soft-error tolerant MPSoCs. SENoC uses hardware-software collaborated approaches to mitigate transient soft errors and meet applications' real-time constraints with minimal overheads. We developed a low-cost on-chip sensor network to collaboratively monitor and detect soft errors, and implemented software-based mechanisms to mitigate soft errors and guarantee correct task executions. To maximize the performance of soft-error tolerant MPSoCs, we proposed a hybrid scheduling scheme to effectively manage applications and resources under uncertainties. The hybrid scheme first optimizes applications by off-line static scheduling techniques, and then uses lightweight scheduling adjustment techniques at runtime to complement the static analysis and handle transient incidents caused by soft errors and task execution variations.

This is the first time that a dynamic hardware-software collaborated method is proposed to systematically mitigate soft errors for MPSoCs using an on-chip sensor network. The new method can effectively reduce protection overheads and improve MPSoC scalability. We studied the proposed method on MPSoCs with different scales and tested it using typical embedded applications under different cosmic ray

978-0-7695-4447-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ISVLSI.2011.48

flux conditions. Experimental results show that comparing to traditional methods the new method requires significantly lower protection overheads to achieve the same level of soft-error tolerance. For instance, soft-error tolerant MPSoCs using the hardware-software collaborated method archive on average 114.1% better performance than a latest software-based method, and it only introduces 0.42% area overhead on a 256-core MPSoCs.

The rest of the paper is organized as follows. Related work is discussed in Section II. An overview of the proposed method for soft-error tolerant MPSoCs is presented in Section III. Its hardware architecture is detailed in Section IV. The software components are presented in Section V. The hybrid scheduling scheme is detailed in Section VI. Section VII shows evaluation and comparison results. Section VIII concludes our work.

II. Related work

Several hardware-based methods are proposed for soft error protections. A globally optimized robust system "BISER" is presented to correct soft errors in latches, flip-flops and combinational logic by reusing on-chip scan resources [5], [7]. It can reduce the cost for error correction quite significantly compared to classical techniques such as TMR. However, performance and energy penalties are still relatively large. A fault-tolerant MPSoC prototype is presented in [8]. It uses checker processors to achieve runtime fault recovery. A multiple clocking of data technique [9] is proposed to remove the error detection overhead from the circuit critical path.

Software-based fault-tolerant scheduling techniques are discussed in [6], [10] for hard real-time multiprocessor systems. Smolens *et al.* [11] presented a spatial and temporal redundancy and value based detection for soft error protections. When a soft error is detected, error correction is achieved by rolling back an execution to its previous checkpoint state and re-executing instructions. Code redefinition techniques [12], [13], [14] derive a new program with the same functionalities as the original application, but with soft error detection capability through redundant execution. These techniques have no special hardware requirements, while they induce significant performance degradation and memory overhead to the system.

A fault-tolerant optimization method is presented in [15] to selectively use hardware redundancy as well as software re-execution. Given an application scenario, it performs a trade-off between selective duplications on hardware component or software process to provide required level of fault tolerance with minimized system costs. Many techniques are proposed to protect transient failures on on-chip interconnects like routers and communication links [16], [17].

III. System overview

SENoC is a hardware-software collaborated solution for soft-error tolerant MPSoCs. It is a systematic approach not only aiming to protect the system from soft errors, but also to optimize system performance in the presence of task execution uncertainties caused by soft errors. It provides robust soft error protection in two steps. A low-cost hardware-based on-chip sensor network is used to detect and report soft errors, and a software-based scheduling technique is applied for error recovery. A hybrid task management strategy is used for system performance optimization and includes two stages — static task scheduling stage at static time and dynamic schedule adjustment stage at runtime.

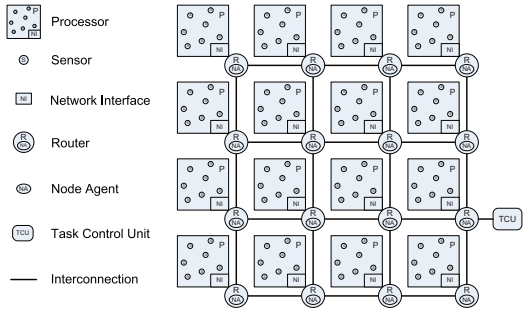


Fig. 1. SENoC architecture overview on a 16-core MPSoC.

Fig. 1 shows an example of SENoC on a 16-core MPSoC. The communication subsystem of SENoC is a scalable Network-on-Chip (NoC), which is shared by regular MPSoC payload packets. On-chip sensors are distributed in each processor tile for soft error detection, and each NoC router is augmented with a node agent for coordination among on-chip sensors. A task control unit is built for centralized management. We will discuss the architecture design of SENoC and its working strategies in Section IV. The software-based soft error recovery technique combines traditional rollback strategy with scheduling techniques to optimize performance. Its realization is detailed in Section V. Besides the soft error protection functions, we further enhance the overall system performance under task execution uncertainties caused by soft errors. We apply static optimization techniques to maximize the applications' performance at static time, and adjust the static scheduling result at runtime according to task execution variations and soft error exceptions. The two-stage static scheduling and dynamic adjustment (SSDA) approach for SENoC is discussed in Section VI.

IV. Hardware architecture and soft error detection

As a system for soft error monitoring and management, SENoC is mainly composed of four types of functional modules: sensors, node agents, a communication subsystem and a centralized task control unit (TCU). On-chip sensors are distributed over the chip to collect information from processors; node agents integrate and analyze information obtained by sensors, and send feedback advices to processors; the communication subsystem is used to share information among the modules; the centralized TCU is used to coordinate among processors to collaboratively complete task executions for applications.

On-chip sensors are built to detect soft errors. Similar to the circuit used by the time redundancy method proposed in [18], the sensor design is shown in Fig. 2. Extra shadow flip-flops

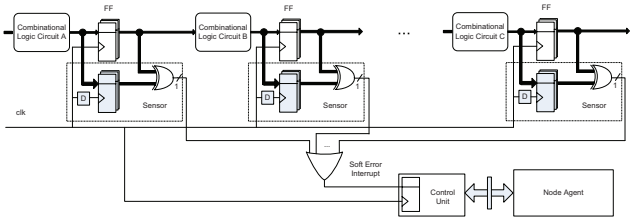


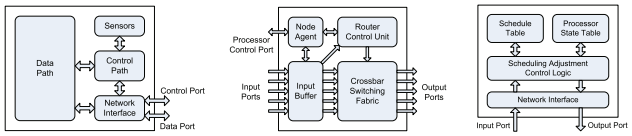
Fig. 2. The design of on-chip sensors.

are added to processors and triggered by a delayed clock signal, such that the two flip-flops on the same processing stage capture the outputs of a combinational circuit at two time instances that differ by a delay. The content of the functional flip-flop is compared with the shadow flip-flop by a series of the XOR gates (working as a set of one bit comparators) to identify soft errors. Once there is a transient pulse on the combinational logic output, it will potentially make the output of the two flip-flops different. The difference is captured by the control unit and reported to a node agent. This sensor design is effective in detecting soft errors occurring indirectly through combinational logic, and it can also detect soft errors occurring in one of the two flip-flops.

SENoC has several advantages over some related techniques who use hardware to provide error-free circuits by integrating detection and recovery circuits together [5]. First, error-free designs result in longer delays in hardware circuits, which are not negligible for modern systems. The tiny sensors only require very low hardware cost compared to error-free designs. Actually, our hardware-software collaborated strategy allows the soft error detected interrupt to be processed in the next clock cycle. Since the clock delay is only applied to the shadow flip-flops, the clock frequency of the functional logic can be largely maintained as normal in most circuit designs. The one cycle delay for setting the soft error interrupt signal is a small overhead for soft error recovery. Second, SENoC makes minimal changes to the existing hardware designs, which makes it easy for system integration. Third, the system becomes flexible for task scheduling since a universal task controller is applied to take care of task executions under different situations.

With the sensors for soft error detection, a communication network is used to collect, share and coordinate the system with sensing information. NoC is composed of routers and interconnects. We add the following functional components to NoC for SENoC: node agents for collecting information from sensors, controlling packet generation and interpretations for intercommunications among processors and the central TCU, and advising processor for task re-execution for error recovery; TCU in charge of centralized control of the system, including maintaining processor states and task executions as well as making scheduling decisions. The communication protocols are proposed for SENoC including packet formats and transmission control. The packet types and their functions are described in Section V.

The structure of a processor core is showed in Fig. 3(a). Sensors with the functionality of soft error detection are spread on each processor. The control logic of a processor is used to maintain processor states and coordinate all the components to make them work correctly. The data path executes tasks for the processor. The network interface is used for communications with a nearby router for packet transmission and delivery. With the design of sensor-enabled processor, we can detect soft error problems during task executions. A processor has four states: clock off, free, busy and exception handling. A special "Exception Handling" state is used for dealing with soft error exceptions. The processor sets its program counter back to the start point of the running task and conducts related rollback operations for soft error recovery. A "Safety Period" is set inside the "Free" state for state transition from "Busy" to "Free". As mentioned, the sensors are possible to report soft error threat after a cycle delay. When a task finishes execution and the processor returns to "Free" state, a delayed soft error report can indicate affected task execution results and soft error recovery is needed in this case, though generally soft error report can be omitted when processor is in "Free" state.



(a) Processor (b) Router with node agent (c) TCU

Fig. 3. The structures of processor, router and TCU.

Routers relay messages (by packets) from one processor to another over interconnected links. The router structure diagram is given in Fig. 3(b). Normally, a router has five bidirectional ports: four of which connect nearby routers in a mesh topology, and the left one is connected to the local processor. Each input port has an input buffer for temporary packet storage. The control unit is used to control packet switching. Most importantly, there is a node agent built inside each router. Node agents mainly have three functions: receiving soft error report from sensors, advising local processor for task re-execution to correct soft errors, and working as a medium between processor and communication network. It can analyze input packets for the processor and generate commands to the processor control logic. On the other hand, it can synthesize processor commands and states into packets to transmit to other network components.

TCU is a global controller used for task management. Its structure diagram is shown in Fig. 3(c). TCU is used to assign new tasks to processors, and maintain the status of all processors. It has a scheduling adjustment control logic, which is used for online task scheduling implementation and dynamic adjustment of pre-defined schedules for performance control. The detailed working strategy of TCU is discussed in Section VI. Note that TCU can also be protected with similar techniques proposed in this paper.

V. Software Design and Soft Error Recovery

Our recovery scheme is a software solution based on task-level scheduling adjustment. As a promising software-based

technique, [11] does not need redundant hardware support to solve soft error problems. It duplicates instruction execution and compare the results to address soft errors, and once a soft error is detected, the software rollbacks to the last checkpoint where the input data for the broken instructions were stored and restarts the execution. Checkpoints are set regularly among software instructions. However, the frequent data backup operations due to the instruction-level checkpoint buildup increase the memory usage, and duplicated instruction execution introduces high performance overhead.

In SENoC, we do not set checkpoints at the instruction level, but use abstraction techniques to represent applications by task graph models, and set checkpoints at the task level (a task in a task graph normally represents a piece of code or a batch of instructions). A buffer memory is usually allocated to the edge between tasks (denoting communication channel) for data sharing. We exploit this feature to set checkpoints on the edges between tasks, and use the buffer for data backup. The input data for a task are kept in the buffer until the task finishes without soft error alert. Once a soft error occurs, the processor will roll the next instruction back to the start point of the affected task and run the task again with the kept input data. Unlike other software-based methods, our checkpoint setting naturally takes advantages of task graph semantics and does not induce additional time or memory overhead to the system.

Based on our scheme, duplicated task execution or backup memory is not necessary, and task-level error recovery is triggered only when the soft error really defects task running. Software overhead is substantially reduced, and recovery overhead is minimized, which are the main benefits of the hardware-software combined solution. A centralized TCU is built to handle the task management and scheduling over the entire chip as well as coordinate on-chip resources. The detailed working strategy is discussed in Section VI.

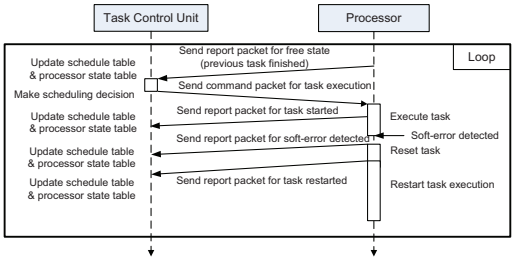


Fig. 4. The timeline sequence diagram for communications between processors and TCU.

The communication protocol between the centralized TCU and processors is described in Fig. 4. Three types of network packets are defined: command packet sent from TCU to processors for delivering task scheduling decisions; report packet sent from processors to TCU for processor state report; payload packet sent between processors for data transmission between inter-processor tasks. Normally, once a processor is known to be free by the TCU and a scheduling decision is made to invoke a task on that processor, a command packet will be sent to the processor. During the task execution, four types of report packets are possible to be sent to the TCU for the events of task started, task finished, soft error detected and task restarted for soft error recovery, respectively. TCU will update the local schedule table and processor state table according to the received reports. As a future extension, the protocol in Fig. 4 can be further enhanced by sending the command packet for task execution to the processor earlier as soon as the next task gets ready. In this way, the communication overhead can be covered and reduced by the simultaneously executed task.

VI. Two-stage scheduling strategy for performance optimization

State-of-the-art techniques for soft error protection only consider using rollback scheduling to recover from soft errors. We do more than that: besides task-level rollback for soft error recovery, we further propose a scheduling strategy to optimize the performance of the entire system.

We conduct centralized scheduling strategy for global management of the entire chip resources and coordination among runtime variations and exceptions. The TCU monitors the entire chip including processor and task states, and makes scheduling and controlling decisions. We propose a static-scheduling-and-dynamic-adjustment (SSDA) strategy, as illustrated in Fig. 5, which is the composition of an offline static task mapping and scheduling algorithm and an online dynamic adjustment strategy.

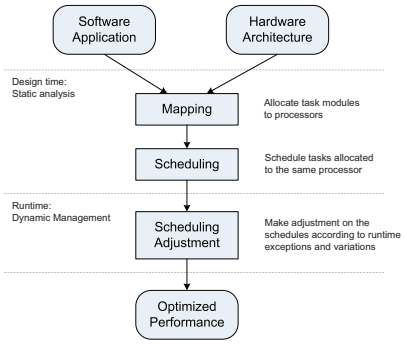


Fig. 5. Two-stage static scheduling and dynamic adjustment (SSDA) strategy.

At design time, without any knowledge of soft error occurrences or task running variations, we try to optimize the application's performance using static real-time scheduling techniques. Load balancing algorithms considering processor workload and network traffic load are applied to distribute processing and network transmission workload evenly to make high resource utilization. The cost function for evaluating a task mapping decision is defined as a linear combination of the task's occupation of the processing time and the total network delay for sending its output data to all the dependent tasks on other processors. The processor with the minimum cost is selected for the task. The tasks assigned to the same processor execute following a static order strictly. This scheduling strategy is proved to be more effective for multiprocessor systems [19].

At runtime, we generally follow the schedule obtained offline, but make slight adjustments according to the runtime

uncertainties, like soft error occurrences and task execution time variations. The light-weight online adjustment algorithm is given in Alg. 1. We use the work-conserving principle that keeps the processors working efficiently on the scheduled tasks, and adjusts the task invocation orders if necessary.

Algorithm 1 Online dynamic scheduling adjustment algorithm

Require: MPSoC P, pre-determined schedule table S, report packets from processors in P

```
for each processor p do
  if p reports free then
    Select the next task v in the schedule table S(p) that is already ready
    Schedule v on p
    Update the schedule table S
  end if
  if p reports soft error occurrence on task v then
    Predict task v's new finish time
    Make adjustment on the schedule table S
  else if p reports execution time variation on v then
    Make adjustment on the schedule table S
  else if p reports processor state change then
    Update the schedule table S
  end if
end for
```

The pre-determined schedule may be advanced or postponed by the runtime events. If a task finishes in a shorter time than its estimation, the TCU will possibly invoke its successors and the next task on that processor earlier, and the performance is expected to be enhanced than the prediction. If a task is affected by soft error, the TCU will re-execute the task and delay its successors and the following tasks on the same processor. Fortunately, if its successors are on other processors, the TCU will adjust the schedule on that processor to allow unaffected tasks to be invoked earlier than the plan, so that the impact of soft error on total performance is reduced to minimum. Alg. 1 has linear complexity to the problem input and runs fast in practice.

VII. Performance Evaluation

In order to verify the effectiveness of the proposed method, we develop a cycle-accurate simulator using SystemC,and conduct extensive performance evaluations using several well-known real-world applications. We compare our solution with state-of-the-art soft-error tolerant techniques on application performance. We construct three MPSoCs using mesh-based NoCs with 16, 64 and 256 processors, respectively. XY routing and wormhole switching techniques are applied in the communication subsystem. We run three groups of simulations. First, we assume the system is running without soft error attack, and obtain the "Error-free" performance of the applications. Second, the proposed "SENoC" is applied to an environment with soft error attacks. Soft errors are simulated to appear on each processor randomly following uniform distribution. A series of terrestrial locations with different cosmic ray flux conditions are selected as the X-axis [20], where the soft error rates can differ in more than three orders of magnitude for different altitudes. Last, for comparison purpose, we assume the same cosmic ray flux conditions as in the second group, and handle soft errors using a latest software-based technique [11]. In the simulations, task execution time variations are described by assigning random execution times following the Gaussian distribution. In order to obtain steady and useful results, we run each application continuously for multiple iterations with pipelining. The end-to-end latency of each simulation run is recorded for analysis.

First, we compare the performance of the techniques in terms of the time used to complete the applications. Figure 6 shows the results for respective applications, where the Error-free performance on 256-core MPSoC is normalized to 1, and the rest results are shown by relative values to it. Results for Error-free cases are very close to those for SENoC and omitted in the figures for clarity purpose. Figure 7 shows the average performance comparison on the three MPSoCs, where the performance values are normalized to [11] on 16-core MPSoC. The average performance overhead by our technique is 0.8% compared to the Error-free case, and the average performance improvement compared to [11] is 114.1%. Experimental results show that our technique has consistently low performance degradation for soft error protection, though soft error rates are largely changed with different terrestrial locations, and it outperforms [11] impressively.

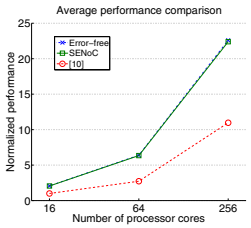
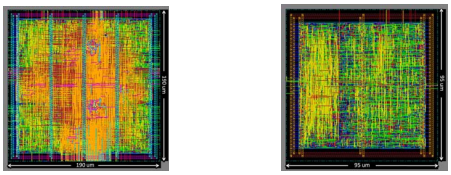


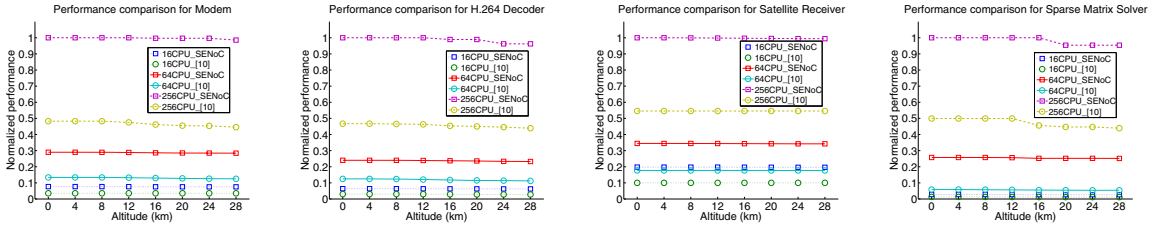
Fig. 7. The comparison of average performance on 16-core, 64-core and 256-core MPSoCs.



(a) Router (b) TCU

Fig. 8. The layouts of the router and task control unit for a 16-core MPSoC.

Next, we implement the hardware and analyze the chip area overhead induced by our technique. Figure 8 shows the layouts of the router and TCU obtained by Cadence Encounterin a 45nm process [21]. Table I summarizes the areas of the hardware components. The area overhead contains three parts: sensors, node agents and the TCU. We assume the area of a processor core to be 660*660 μm^2 and has five pipeline stages.Routers are designed with five bi-directional ports and input buffers with the depth of two packets. With the synthesized results, we show that the area overhead of our technique is 0.37%, 0.39% and 0.42% for 16-core, 64-core and 256-core MPSoCs, respectively, and the average area



(a) Modem (b) H.264 decoder (c) Satellite receiver (d) Sparse matrix solver

Fig. 6. Performance comparisons between SENoC and [11] on 16-core, 64-core and 256-core MPSoCs.

overhead from the additional hardware components used by our technique is 0.39%, which is negligible considering the huge performance gain.

TABLE I
AREAS OF SENOC COMPONENTS.

Components	Area (μm^2)
Processor core	435600
Router	22803
Sensors (on a core)	724
Node agent	687
TCU on 16-core MPSoC	8599
TCU on 64-core MPSoC	50608
TCU on 256-core MPSoC	127315

VIII. CONCLUSION

This is the first time that a dynamic hardware-software collaborated method is proposed to systematically mitigate soft errors for MPSoCs using an on-chip sensor network. We developed a low-cost on-chip sensor network to collaboratively monitor and detect soft errors, and implemented software-based mechanisms to mitigate soft errors and guarantee correct task executions. To maximize the performance of soft-error tolerant MPSoCs, we proposed a hybrid scheduling scheme to effectively manage applications and resources under uncertainties caused by soft errors. The new method can substantially reduce protection overheads and improve MPSoC scalability. We studied the proposed method on MPSoCs with different scales and tested it using typical embedded applications under different cosmic ray flux conditions. Experimental results show that comparing to latest traditional methods the new technique requires significantly lower protection overheads to achieve the same level of soft-error tolerance.

ACKNOWLEDGEMENT

This work is supported by RGC, Hong Kong SAR.

REFERENCES

[1] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "H. 264 hdtv decoder using application-specific networks-on-chip," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, july 2005, pp. 1508 –1511.

[2] M. Nicolaidis, "Design for soft error mitigation," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 405 – 418, sept. 2005.

[3] D. K. Pradhan, Ed., *Fault-tolerant computer system design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[4] S. Mukherjee, *Architecture Design for Soft Errors*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[5] S. Mitra, M. Zhang, S. Waqas, N. Seifert, B. Gill, and K. S. Kim, "Combinational logic soft error correction," in *Test Conference, 2006. ITC '06. IEEE International*, oct. 2006, pp. 1 –9.

[6] G. Manimaran and C. S. R. Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distrib. Syst.*, vol. 9, no. 11, pp. 1137–1152, 1998.

[7] S. Mitra, "Globally optimized robust systems to overcome scaled cmos reliability challenges," in *DATE '08: Proceedings of the conference on Design, automation and test in Europe*. New York, NY, USA: ACM, 2008, pp. 941–946.

[8] X. Zhu and W. Qin, "Prototyping a fault-tolerant multiprocessor soc with run-time fault recovery," in *DAC '06: Proceedings of the 43rd annual Design Automation Conference*. New York, NY, USA: ACM, 2006, pp. 53–56.

[9] N. Avirneni and A. Somani, "Low overhead soft error mitigation techniques for high-performance and aggressive designs," *Computers, IEEE Transactions on*, 2011.

[10] S. Ghosh, R. Melhem, and D. Mossé, "Fault-tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 3, pp. 272–284, 1997.

[11] J. C. Smolens, B. T. Gold, B. Falsafi, and J. C. Hoe, "Reunion: Complexity-effective multicore redundancy," in *MICRO 39: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 223–234.

[12] M. Rebaudengo, M. S. Reorda, M. Torchiano, and M. Violante, "Soft-error detection through software fault-tolerance techniques," in *DFT '99: Proceedings of the 14th International Symposium on Defect and Fault-Tolerance in VLSI Systems*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 210–218.

[13] B. Nicolescu, R. Velazco, M. Sonza-Reorda, M. Rebaudengo, and M. Violante, "A software fault tolerance method for safety-critical systems: Effectiveness and drawbacks," in *SBCCI '02: Proceedings of the 15th symposium on Integrated circuits and systems design*. Washington, DC, USA: IEEE Computer Society, 2002, p. 101.

[14] G. A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. I. August, "Swift: Software implemented fault tolerance," in *CGO '05: Proceedings of the international symposium on Code generation and optimization*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 243–254.

[15] V. Izosimov, I. Polian, P. Pop, P. Eles, and Z. Peng, "Analysis and optimization of fault-tolerant embedded systems with hardened processors," in *DATE*. IEEE, 2009, pp. 682–687.

[16] A. Dutta and N. A. Touba, "Reliable network-on-chip using a low cost unequal error protection code," *Defect and Fault-Tolerance in VLSI Systems, IEEE International Symposium on*, vol. 0, pp. 3–11, 2007.

[17] A. Patooghy, M. Fazeli, and S. G. Miremadi, "A low-power and seu-tolerant switch architecture for network on chips," in *PRDC '07: Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 264–267.

[18] M. Nicolaidis, "Time redundancy based soft-error tolerance to rescue nanometer technologies," in *VTS '99: Proceedings of the 1999 17TH IEEE VLSI Test Symposium*. Washington, DC, USA: IEEE Computer Society, 1999, p. 86.

[19] W. Liu, Z. Gu, J. Xu, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[20] JEDEC Standard, JESD89, 2001, http://www.jedec.org.

[21] "Nangate 45nm open cell library," http://www.nangate.com.