An Asymmetric Checkpointing and Rollback Error Recovery Scheme for Embedded Processors

Hamed Tabkhi, Seyed Ghassem Miremadi, and Alireza Ejlali
Dependable Systems Laboratory (DSL)
Department of computer engineering
Sharif University of Technology, Tehran, Iran
tabkhi@ce.sharif.edu, miremadi@sharif.edu, ejlali@sharif.edu

Abstract

This paper presents a checkpointing scheme for rollback error recovery, called Asymmetric Checkpointing and Rollback Recovery (ACRR) which stores the processor states in an asymmetric manner. In this way, error recovery latency and the number of checkpoints are reduced to increase the probability of timely task completion for soft real-time applications. To evaluate the ACRR, this scheme was studied analytically. The analytical results show that the recovery latency is reduced as non-uniformity of the checkpoint increases. As a case study, the ACRR is implemented and simulated on a behavioral VHDL model of LEON2 processor. The simulation results follow the results obtained in the analytical study.

1. Introduction

Embedded processors are widely used in safety-critical real-time systems such as flight control systems, automotive electronics and fabrication equipment systems [9], where the occurrence of failures in such systems can cause catastrophic consequences. Fault-tolerant and error recovery methods are used to prevent these systems to fail [16]. The dominant error recovery technique for the uniprocessor systems is the rollback error recovery [1, 11, 12, 14, 18, 19, 20] that is based on re-execution of specific instructions, when an error occurs during execution of these instructions. Two main measures to evaluate the rollback error recovery are the recovery latency and the number of checkpoints inserted in an application program [8]. The optimal recovery latency and the number of checkpoints have been analytically investigated by several researchers [7, 10, 17, 21, 22]. These investigations are based on one or more of the following assumptions:

- A zero error detection latency is assumed [7, 10, 21, 22],
- The interval between checkpoints in an application program is uniform [7, 21, 22],
- The interval between checkpoints in an application program is non-uniform [10].

The main drawback of the above studies is the assumption of the zero error detection latency. However, many practical systems are involved with error detection latency [3, 13], which in turn may impose delay in the error recovery latency. The assumption to zero error detection latency certifies that the optimal rollback should be to the most recent checkpoint [10, 21]. However, it has been shown that rollback to the most recent checkpoint is not a valid choice, because of the error detection latency which in practical systems, is not zero [6, 15, 19, 20].

1550-5774/08 $25.00 © 2008 IEEE
DOI 10.1109/DFT.2008.27

This paper presents a checkpointing and error recovery scheme that supports the non-zero error detection latency. This scheme called Asymmetric Checkpointing and Rollback Recovery (ACRR). The ACRR scheme is based on a static non-uniform checkpoints placement that stores the processor states in an asymmetric manner. The analytical results show that the ACRR has less average recovery latency than the uniform checkpointing methods. Consequently, the ACRR scheme increases the probability of timely task completion for soft real-time systems. As a case study, the ACRR scheme is implemented and simulated on a behavioral VHDL model of LEON2 processor. In implementation of the ACRR scheme, a checkpointing mechanism that stores a lower amount of information of the processor state with deterministic checkpointing latency is presented.

The organization of this paper is as follow: Section 2 introduces some definitions and restrictions in checkpointing and rollback error-recovery. The analytical study of the ACRR scheme is presented in Section 3. Section 4 presents an experimental implementation of the ACRR scheme on LEON2 processor. The ACRR simulation results are presented in Section 5, and finally Section 6 concludes this paper.

2. Checkpoint and Rollback Policies

The following five definitions are necessary in the discussion:

- Checkpointing interval: the duration of time between two consecutive checkpoints (speculative epoch).
- Checkpointing latency: the duration of required time that the execution of a program is stalled to store the state of a processor.
- State restoration latency: the duration of required time for restoring the last fault free state of a processor
- Recovery latency: the duration of required time to re-executing faulty instructions.
- Total checkpointing and recovery latency: the duration of required time for checkpointing, state restoration, and error-recovery that is added to the real execution time of an application program.

The error detection latency should not be neglected in the checkpointing and error recovery methods, especially in embedded systems where cost and power consumption are important factors. The error detection mechanisms with low detection latency such as module redundancy impose a high overhead which cannot be acceptable for a varied range of the embedded systems like automotive electronics [14].

If an error occurs before a checkpoint and be detected afterward, the stored data in this checkpoint may be corrupted. As a result, two most recent checkpoints should have been stored previously to prevent the processor from rollback to an invalid checkpoint. It means that the processor should be returned to the second recent checkpoint which in this paper is called penultimate checkpoint (Figure 1).

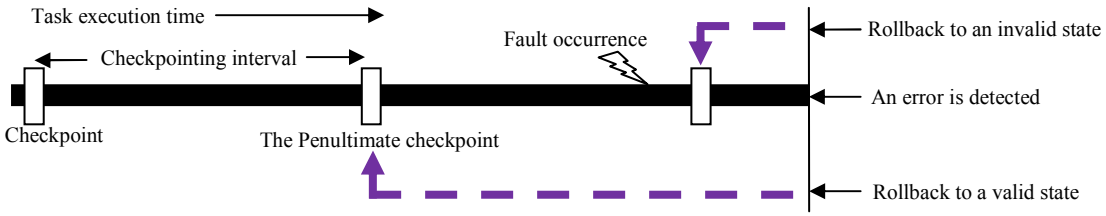

Figure 1. Rollback to the penultimate checkpoint

Rollback to the penultimate checkpoint is a general constraint which is imposed by the error detection latency and is independent of the checkpointing interval. Also, the checkpointing interval must be greater than the worst-case latency of the error detection mechanism. Otherwise, the error may corrupt the stored data of both recent checkpoints. The analytical studies which neglect error detection latency, relinquish the above constraints [7, 10, 21, 22]. In contrast to the analytical studies, most of the practical rollback error recovery methods store two most recent checkpoints and rollback to the penultimate checkpoint in case of error occurrence [6, 15, 19, 20].

3. Optimal Checkpoints Placement

This study is performed for uniprocessor and aperiodic task which assumes K errors would occur during task execution time and the probability of error occurrence is uniform during task execution time. Below notations are used through the remainder of this paper:

- E: Real execution time of the task without checkpointing and error recovery.
- I: Total execution time of the task with checkpointing and error recovery.
- N: Number of checkpoints.
- N_{op}: Optimal number of checkpoints.
- T: Checkpointing interval that equaled to $E/(N+1)$.
- R: Recovery latency.
- S: State restoration latency.
- C: Checkpointing latency.
- K: Number of errors which occur during task execution time.
- P: Probability of timely task completion.

3.1 Uniform Checkpointing

As it mentioned in Section 2, the processor rolls back to the penultimate checkpoint when an error is detected. Therefore, the two recent intervals affect the recovery latency and must be considered in the analysis of the optimal checkpoints placement. When the checkpoints are uniform and the checkpointing intervals are equal (Figure 2(a)), the average recovery latency is given by:

$$R_{average} = T + \frac{\int_{T}^{2T} t\,dt}{T} = T + \frac{\frac{1}{2}T^2}{T} = \frac{3}{2}T = \frac{3}{2}\frac{E}{N+1} \qquad (4\text{-}1)$$

The average execution time of a program is given by:

$$I_{average} = E + (N+K)C + KS + KR_{average} = E + (N+K)C + KS + K\frac{3}{2}\frac{E}{N+1} \qquad (4\text{-}2)$$

In equation (4-2), the number of times that checkpointing latency has been considered is $N+K$. because, when a processor rollbacks to the penultimate checkpoint, one extra checkpoint must be repeated each time. We can derive the value of N which minimizes I by differentiating (4-2) with respect to N. This gives a cubic equation in N, given by:

$$\frac{dI_{average}}{dN} = 0 \quad \Rightarrow \quad C - K\frac{3}{2}\frac{E}{(N+1)^{-2}} \quad \Rightarrow N_{op} = \sqrt{\frac{3}{2}\frac{KE}{C}} - 1 \qquad (4\text{-}3)$$

In equation (4-3), N_{op} should not be less than zero and $E/(N_{op}+1)$ should be greater than the error detection latency. Otherwise as described before, error may corrupt two consecutive checkpoints.

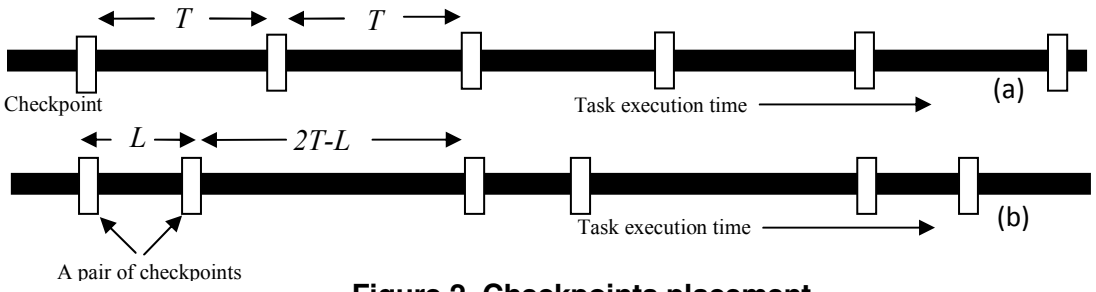

Figure 2. Checkpoints placement

3.2 The ACRR: Asymmetric checkpointing and rollback recovery

Instead of placing *N* checkpoints in the task execution time uniformly, the ACRR scheme inserts *N/2* pairs of checkpoints with interval *L* inside the pairs, and *2T-L* between two consecutive pairs. In such approach, sum of two consecutive intervals always equals to *2T* which is similar to the uniform checkpointing (Figure 2(b)). In fact, the uniform checkpointing is a special case of the ACRR scheme where *L* is equal to *T*. The value of *T* depends on the number of checkpoints. It should be noticed that both *L* and *2T-L* are greater than the error detection latency. In this case, with assumption that the probability of error occurrence is uniform during the task execution time, the average recovery latency is given by:

$$R_{average} = \frac{L}{2T}(2T - L + \frac{\int_0^L tdt}{L}) + \frac{2T-L}{2T}(L + \frac{\int_0^{2T-L} tdt}{2T-L}) = T + L - \frac{L^2}{2T} \qquad (4\text{-}4)$$

In comparison with the uniform checkpointing, it can be shown that for all values of *L*, *Raverag* from the equation (4-4) are equal or less than *Raverag* from the equation (4-1). Therefore, the *Raverag* in uniform checkpointing is the worst-case of the *Raverag* in the ACRR scheme with the same value of *N*. The average recovery time (*Raverag*) for different values of parameter *L* is shown in Figure 3. The *E* values are obtained from real execution of four benchmarks on the LEON2 processor. Figure 3 illustrates that in the ACRR scheme, the *Raverag* decreases as the parameter *L* decreases.

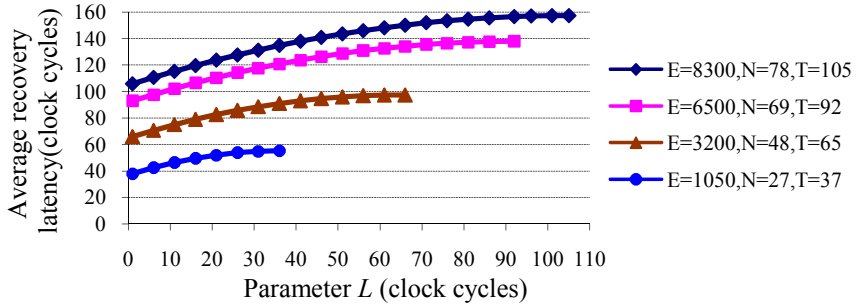

Figure 3. The average recovery latency for the same *N* when *K*=1

By substituting the values of *Raverag* derived from (4-4) in the equation (4-2), we obtain:

$$I_{average} = E + (N+K)C + KS + K(T + L - \frac{L^2}{2T}) = E + (N+K)C + KS + K(\frac{E}{N+1} + L - \frac{L^2(N+1)}{2E}) \qquad (4\text{-}5)$$

We can derive a new optimal value of *N* which minimizes *Iaverage* by differentiating (4-5) with respect to *N*. This gives a cubic equation in *N*, given by:

$$\frac{dI_{average}}{dN}=0 \quad \Rightarrow \quad C+K(-\frac{E}{(N+1)^{-2}}-\frac{L^2}{2E})=0 \quad \Rightarrow \quad N_{op}=\sqrt{\frac{2KE^2}{2EC-L^2K}}-1 \qquad (4\text{-}6)$$

Based on the experimental implementation in Section 5, the value of C is chosen 2 clock cycle. The N_{op} for the different ranges of parameter L is shown in Figure 4 (when C=2, and K=1). This figure illustrates that reduction of L value will decrease the amount of N_{op}.

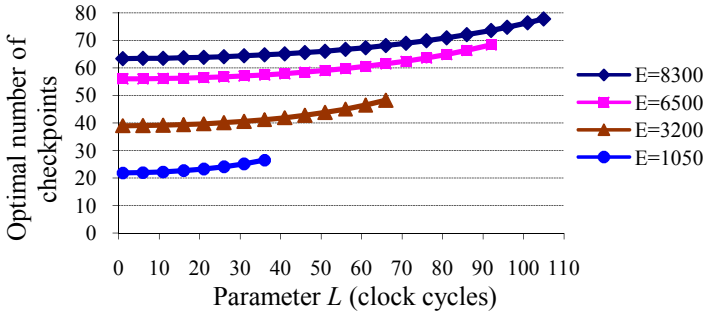

Figure 4. The optimal number of checkpoints for different ranges of parameter *L*

The total latency is equal to $(N+K)\times C+K(R+S)$. Figure 5 illustrates the Total latency of the ACRR scheme for the different values of parameter L (when E=6500, C=2, K=1). According to the equation (4-6), decreasing parameter L leads to reduction in N_{op} *value*. This means that the amount of N_{op} in the ACRR scheme is lower than the amount of N_{op} in the uniform checkpointing with the same task execution time.

The fact that the N_{op} decreasing can lead to an increase in R_{averag} for several values of L is not important, because the total latency, which is the sum of total checkpointing latency and recovery latency, always decreases as N decreases. The relations between N_{op}, L, R_{averag}, and the total latency in ACRR scheme, are illustrated in Figure 5.

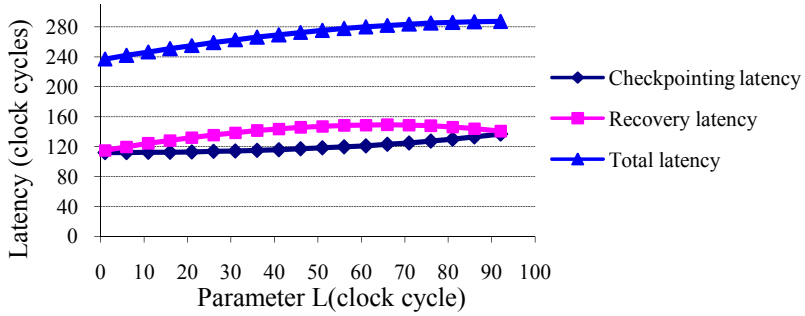

Figure 5. Total checkpoint and recovery latency for Matrix multiply when *K*=1

Both of (4-5) and (4-6) equations show that the optimal number of checkpoints decreases as the parameter L decreases. In the ACRR scheme the lower bound of L is error detection latency which seems to be the best value for L. Although we explain in Section 5 that determining a precise value for error detection latency is difficult in some cases. The analytical results show that the total latency is reduced as the non-uniformity of the checkpointing intervals increase.

The reduction of total checkpointing and recovery latency can increase the probability of timely task completion for soft real-time applications. As an example, for a soft real-time task with parameters E=8300, C=2, S=20, and soft deadline equals to 8750 clock cycle, both uniform checkpointing and ACRR schemes can tolerate a single error (K=1) without missing

the soft deadline. In case of K=2, the figures for the uniform checkpointing scheme are: N_{op}=110, T=74, P=16%. For the ACRR scheme, the figures for K=2, and L=40 are: N_{op}=95, T=40 and 132, P=40%, and the figures for K=2, and L=15 are: N_{op}=90, T=15 and 165, P= 61%.

4. The ACRR implementation

Fault occurrences for any reason such as cosmic ray radiations may cause to control or data errors. These errors finally lead to incomplete execution, data violation or processor crash. So, the aim of error recovery is returning processor to a valid state that the wrong manipulations on data and control become ineffective. The state of processor includes the memory elements values, which hold data or control signals like registers file, control registers, and cache memory. There is no need to restore all memory elements if we need to rollback to a valid state. Checkpoints can accomplish in a manner that only manipulated data during checkpointing interval have been stored (incremental checkpointing). In this section, the ACRR scheme in micro architecture-level is implemented on the VHDL model of LEON2 processor. LEON2 is a 32 bit processor that conforms to the SPARC V8 architecture [4].

4.1 Recovery unit controller (RUC)

In order to implement ACRR, LEON2 processor is extended by adding an extra unit, called RUC (Recovery Unit Controller). RUC has been connected to execution unit, registers file and data cache memory. RUC schedules and controls all related checkpointing and recovery activities. As mentioned before, two most recent checkpoints should have been stored to prevent from returning to an invalid checkpoint. RUC manages this activity and determines the checkpointing intervals with respect to error detection latency. All error signals from different parts of the processor are sent to the RUC. Regarding to the detection latency of this error signals, RUC determines the valid checkpoint for error recovery.

4.2 Checkpointing mechanism

The applied checkpointing mechanisms in each part of a processor may be different, but the checkpointing consistency between these mechanisms can leads to a successful checkpointing and error recovery. For implementing the ACRR scheme, two different policies have been applied: 1) for global, status, and control registers and register file, 2) for data memory and cache.

Two backup registers are allotted to each control and status registers. The backup registers hold the content of the control and status registers in two most recent checkpoints. In each checkpoint, RUC stalls pipeline and the content of control and global registers is copied to the backup registers. In case of error occurrence, RUC stalls pipeline and the content of the backup registers are restored to the original registers. Both store and restoration can be done in just one clock cycle. For register file the same scenario is applied. Two extra register files are allotted for holding backup values of the register file.

For data memory, the same scenario cannot be used. Memory, both in size and access frequency is not comparable with register file. Most of the pervious works have implemented memory checkpointing in cache-level [2, 15, 19]. These works try to avoid writing the manipulated data in the cache memory to the main memory until be assured of the data correctness. The weakness point of these methods is undeterministic checkpointing latency. In these methods, the required time for checkpointing is unpredictable and depends on the number of write instructions in each checkpointing interval. Also in these methods the

performance of the data cache degrades as the checkpointing interval increases. As a result, the total performance of the systems that using the cache-level error recovery is unpredictable.

In the ACRR scheme, instead of holding manipulated data in cache memory until the next checkpoint, the cache writes are updated in the main memory and the previous values of the written addresses is stored, simultaneously. Two backup buffers are allotted to the data cache. The backup buffers hold previous data in two most recent checkpointing intervals. If any write instruction executes during the checkpointing interval, the previous data and its address are stored in the backup buffers. Therefore, all of the memory checkpointing activity can be done parallel with real execution of the application program. In case of error occurrence, the old data in the backup buffers restore to their original places either in the data cache or main memory. In this case, the state restoration latency contain the duration of required time to update the data cache and main memory with the valid data from the backup buffers. In order to have shorter and predictable state restoration latency, copy to the main memory can be done in parallel with re-execution. According to the equation (3-5), reducing the checkpointing latency, causes an increase in the number of checkpoints and decrease in the checkpointing interval. Note that the probability of write instruction in each interval is reduced as the checkpointing interval decreases. This leads to low and more predictable state restoration latency.

In the proposed scheme, the checkpointing latency equals to 2 clock cycle which is a constant value and is independent of checkpointing interval. Since the main goal of this experimental work is implementing the ACRR scheme on the LEON2 processor as a case study, we neglect interrupt or I/O action during the runtime of benchmarks. However it is possible to remove this limitation without any negative effects on the proposed scheme. Also we assume the checkpoint memory elements are robust and fault-tolerant techniques like error correction codes are applied to them.

5. Experimental Results

To carry out the experiments to evaluate the ACRR scheme, four benchmarks have been run on the LEON2 (Bitcount and basicmath of automotive benchmarks from MiBench suit [5], bubble sort and matrix multiply). A controllable random signal generator is used for sending error detection signals to the Recovery Unit Controller (RUC). To determine the optimal number of checkpoints, the amount of C (checkpointing latency) and L (error detection latency) in each benchmark is required. Following to the practical implementation in Section5, the amount of C equals to 2 clock cycle that is constant value in different benchmarks and different checkpointing intervals. The total latency and overhead of uniform checkpointing are shown in Table 1 for each benchmark. The numbers of checkpoints are obtained analytically by formulas which have been presented in Section 2.

Table 1. The uniform checkpointing overhead

Benchmark	Total execution Time [cycles]	number of checkpoints	Checkpointing interval [cycles]	Total overhead [%]
Bubble sort	1171	27	38	11.5
Basicmath	3409	48	65	6.5
Matrix multiply	6793	69	92	4.5
Bitcount	8633	78	105	4.0

In practical applications, error detection latency differs depending on the fault nature and applied error detection mechanism. For example, in [3] and [13] control flow checking methods for embedded and COTS processors are presented which the maximum amount of

error detection latency is about 50 instructions and 60 clock cycles in these methods, respectively. However, values of L are assumed to be greater than error detection latency and the benchmarks are executed for two cases, L=15 and L=30 on the LEON2 processor. Totally, more than 1500 times, the error signal has been activated. The total latency improvement ratio is the proportion of latency decrease in the ACRR scheme to uniform checkpointing. The obtained results are presented in Table 2.

Table 2. The ACRR scheme overhead (when *K*=1, *L*=15 *and L*=35*)*

Benchmark	Total execution time [cycles]	Number of checkpoints	Ch. P. Interval [cycles]		Overhead [%]	Late. Imp.* ratio [%]
			Interval1	Interval2		
Bubble sort, L=15	1158	22	73	15	10	12
Bubble sort, L=35	1165	26	40	35	10.9	5
Basicmath, L=15	3379	39	143	15	5.5	16
Basicmath, L=35	3391	41	115	35	5,9	9
Matrix multiply, L=15	6748	56	211	15	3.8	18
Matrix multiply, L=35	6763	57	187	35	4.0	11
Bitcount, L=15	8578	63	241	15	3.3	19
Bitcount. L=35	8594	64	217	35	3.5	13

* Total latency improvement ratio relative to uniform checkpointing

Table 3. The ACRR scheme for Bitcount benchmark (when k=2)

Benchmark	Total execution time [cycles]	Number of checkpoints	Ch. P. Interval [cycles]		Overhead [%]	Prob. * D=8750	Prob. D=8800
			Interval1	Interval2			
Uniform	8767	110	74	74	5.6	16	76
ACRR, L=15	8719	90	165	15	5.0	59	82
ACRR, L=35	8738	93	140	35	5.2	45	77

* Probability of timely task completion with soft deadline D

Table 2 shows that the latency improvement ratio in the ACRR scheme increases as the execution time of the benchmarks grows. Note that average execution time increases as the parameter L increases. Also, the optimal number of checkpoints in the ACRR scheme decreases relative to uniform checkpointing.

The experiments are repeated for the Bitcount benchmark and the results are averaged out over these runs. We are interested here in the probability of timely task completion P which the task completes before the soft deadline D. To illustrate more advantages of the ACRR scheme relative to the uniform checkpointing, we note that if we set K=2 (using the value of L as before), The proposed ACRR scheme provides higher value of P relative to the uniform checkpointing as the slack time decreases. In some cases, up to 40% increase is obtained in the probability of timely task completion; the results are shown in Table 3.

6. Conclusions

A checkpointing and rollback recovery scheme, called Asymmetric Checkpointing and Rollback Recovery (ACRR) was presented which store the states of a processor in an asymmetric manner. It was shown that the ACRR scheme reduced the error recovery latency and the optimal number of checkpoints to increase the probability of timely task completion. The ACRR scheme was evaluated analytically and practically. The evaluation results showed that the ACRR scheme reduced the average task execution time and was more likely to meet

soft deadlines. The amount of these improvements depended on the error detection latency. The proposed approach could be extended to a set of multiple periodic tasks with considering other important parameters in embedded systems such as power consumption.

7. References

[1] M. Bashiri, S. G. Miremadi, and M. Fazeli, "A Checkpointing Technique for Rollback Error Recovery in Embedded Systems," *Proceeding of the 18th IEEE International Conference on Microelectronics (ICM 06),* 16-19 Dec. 2006, pp. 174-177,

[2] N. S. Bowen, and D.K. Pradhan, "Processor and Memory-Based Checkpoint and Rollback Recovery," *Proceeding of Computer,* Vol. 26, Feb. 1993, pp. 22-31.

[3] M. Fazeli, R. Farivar, and S. G. Miremadi, "A software-based concurrent error detection technique for power PC processor-based embedded systems," *Proceeding of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Oct. 2005, pp. 266- 274.

[4] J. Gaisler, *Leon2 Processor*, www.gaisler.com

[5] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R.Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *IEEE International Workshop on Workload Characterization* 2 dec. 2001, pp. 3-14.

[6] M. J. Iacoponi, "Hardware Assisted Real-Time Rollback in the Advanced Fault-Tolerant Data Processor," *Proceeding of the 10th IEEE Digital Avionics Systems Conference*, 1991, pp 169-274.

[7] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of Fault-Tolerant Embedded Systems with Soft and Hard Timing Constraints," *Design, Automation, and Test in Europe (DATE 2008*), Munich, Germany, 10-14 Mar. 2008, pp. 915-920.

[8] I. Koren, and C. M. Krishna, *Fault Tolerant Systems,* Morgan Kaufmann, USA, 2007.

[9] P. Marwedel, *Embedded System Design*, Springer, Netherlands, 2006.

[10] R. Melhem, and E. Elnozahy, "The Interplay of Power-Management and Fault-Recovery in Real-Time Systems," *IEEE Transaction on Computers 2004,* Vol. 53, Issue 2, pp. 217-231, Feb. 2004.

[11] N. Nakka, K. Pattabiraman, and R. Iyer, "Processor-Level selective Replication," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,* june 2007, pp. 544-553.

[12] M. Pflanz, and H. T. Vierhaus, "On-line Check and Recovery Techniques for Dependable Embedded Processors," *IEEE MICRO,* Vol. 21, Issue 5, Sep/Oct. 2001, pp. 24-40.

[13] F. Rota, S. Dutt, and S. Krishna, "Off-Chip Control Flow Checking of On-Chip Processor-Cache Instruction Stream", *Proceeding of the 21th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems,* Oct. 2006, pp. 507-515.

[14] T. Sakata, T. Hirotsu, H. Yamada, and T. Kataoka, "A Cost-Effective Dependable Microcontroller Architecture with Instruction-Level Rollback Recovery," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks,* June 2007, pp. 256-265.

[15] S. Shyam, K. Constantinides, S. Phadke, V. Bertacco, and T. Austin, "Ultra Low-Cost Defect Protection for Microprocessor Pipelines*," Proceedings of the 2006 ASPLOS Conference*, 2006, pp. 73-82.

[16] L. Spainhower, and T. A. Gregg, "G4: A Fault tolerant CMOS mainframe," *proceeding of 28th fault tolerant computing,* June 1998, pp. 432-440.

[17] N. H. Vaidya, "A Case for Two-Level Recovery Schemes," *IEEE Transaction on Computer*, Vol. 47, Issue 6, June 1998, pp. 656-666.

[18] Y. Tamir, and M. Tremblay, "High Performance VLSI Systems Using Micro rollback," *IEEE Transaction on Computers,* Vol. 39, Issue 4, Apr. 1990, pp. 548-554.

[19] R. Teodorescu, J. Nakano, and J. Torrellas, "SWITCH: A Prototype for Efficient Cache-Level Checkpointing and Rollback," *IEEE MICRO,* Vol. 26, Issue 1, Jan/Feb. 2006, pp. 28-39.

[20] H. Wang, S. Rodriguez, C. Dirik, A. Gole, V. Chan, and B. Jacob, "TERPS: The Embedded Reliable Processing System," *Proceedings of the ASP-DAC, IEEE Design Automation Conference,* Vol. 2, Jan. 2005, pp. d/1-d/2.

[21] Y. Zhang, and K. Chakrabarty, "Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition,* 2003, pp. 918-923.

[22] Y. Zhang, and K. Chakrabarty, "Fault Recovery Based on Checkpointing for Hard Real-Time embedded systems," *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI systems,* 3-5 Nov. 2003 pp. 320-327.