Soft Error Mitigation Schemes for High Performance and Aggressive Designs

Naga Durga Prasad Avirneni, Viswanathan Subramanian, and Arun K. Somani
Dependable Computing and Networking Laboratory
Iowa State University, Ames, IA, USA
{avirneni,visu,arun}@iastate.edu, Ph: +1 (515) 294-0442

Abstract—**In this paper, we address the issue of soft errors in random logic and develop solutions that provide fault tolerance capabilities without logic duplication. First, we present a circuit level soft error mitigation technique which allows systems to operate without the performance overhead of soft error detection and correction circuitry. This is achieved by sampling data using our Soft Error Mitigation (SEM) register cell, which uses a distributed and temporal voting scheme for soft error detection and correction. Next, we present a scheme to concurrently detect and correct soft and timing errors using a Soft and Timing Error Mitigation (STEM) register cell, which offers soft error protection in aggressive designs that allow overclocking. Timing annotated gate level simulations, using $45nm$ libraries, of a pipelined adder-multiplier circuit and five-stage DLX processor show that both of our techniques achieve near 100% fault coverage. For DLX processor, even under severe fault injection campaigns, SEM achieves an average performance improvement of 26.58% over a temporal Triple Modular Redundancy (TMR) voter based register cell, while STEM outperforms SEM by 27.42%.**

Index Terms— Soft error, dependable and adaptive systems, overclocking

I. Introduction

The threat of soft error induced system failure is becoming more prominent in modern computing systems implemented in deep sub micron process technologies. In the past, single event upsets (SEUs) were a major concern only in space applications, creating hard threats like loss of control, resulting in catastrophic failures. An SEU is induced when a high energy particle, either from cosmic radiations or decaying radioactive materials, strikes the silicon substrate. If enough charge is deposited by the strike, it causes a bit flip in the memory cell or a transient pulse in the combinational logic. The latter is referred to as a Single Event Transient (SET). Radiation induced SET pulses have widths in the range of $500ps$ to $900ps$ in the $90nm$ process, as compared to $400ps$ to $700ps$ in the $130nm$ process [1]. As a result, terrestrial applications also require fault tolerant techniques to ensure their dependability.

Much of the previous work has concentrated on how to protect memories from soft errors, as they occupy a significant portion of on-chip area. Shivakumar, et. al. [2] show that, problem of soft error in combinational circuits is becoming comparable to that of unprotected memory elements in current and future technologies. Recently, techniques have been proposed to make combinational logic immune to soft errors, as they threaten the usefulness of technological advancements. Providing fault tolerance capabilities for random and complex logic is expensive, both in terms of area and power. Techniques such as, duplication and comparison, and triple modular redundancy (TMR) and majority voting have been proposed to mitigate soft error rate (SER) in logic circuits [3]. These approaches incur performance overhead, even during error free operation. Also at this juncture, when static power is comparable to dynamic power, logic replication is not a viable alternative.

Increasing system wide integration force designers to adopt worst-case design methodologies while designing individual system components. With these design practices, safety margins are added to address parameter variations, which include intra-die and inter-die process variations, and environmental variations, which include temperature and voltage variations [4]. These additional guard bands are becoming non-negligible in nanometer technologies. Designers conservatively add these safety margins to salvage chips from timing failures and shortened lifetime. Most systems are characterized to operate safely within vendor specified operating frequency. When they are operated beyond this rated frequency, timing errors and system failure may happen.

Overclocking as a means to improve performance is gaining popularity among high-performance enthusiasts [5]. Circuits exhibit worst case delay only when their longest delay paths are sensitized by the inputs. However, these worst case delay inducing inputs and operating conditions are rare, leading to room for performance improvement that overclockers exploit [6]. Overclocking without guaranteeing functional correctness leads to unpredictable system behavior and loss of data. Aggressive, but reliable, design methodologies employ relevant timing error detection and recovery schemes to prevent erroneous data from being used [7]. In [8], it has been shown that operating frequency can be increased beyond worst case limit, allowing systems to operate at optimal clock frequency, by adapting to the current set of instructions and environmental conditions. Moreover, many systems operate at a overclocked frequency, which is 15-20% higher than worst case frequency, without increasing the error rate beyond 1% [9].

In this paper, we address the issue of soft errors in random logic and develop solutions that provide fault tolerance capabilities without logic duplication. We propose two techniques that have low area and performance overhead. Our first technique, SEM, replaces register elements in a circuit with **Soft**

Error Mitigation (SEM) register cells. Our second technique, STEM, concurrently detects and corrects soft and timing errors using **Soft and Timing Error Mitigation** (STEM) register cells. STEM cells support reliable overclocking. Both techniques employ distributed and temporal voting schemes with in-situ error detection and fast recovery. For error detection and correction, our temporal sampling mechanisms sample data at three different time intervals. In both SEM and STEM techniques, we support circuit level speculation. We allow data to move forward speculatively, and if an error happens we void the computation and redo it.

For our initial experimental study, we integrated our data sampling mechanisms into a two stage pipeline consisting of an adder and a multiplier. In these experiments, we measured the amount of fault masking that occurs on the faults injected into pipeline logic and performance improvement. In order to fully understand the performance improvement and fault coverage that our scheme can provide to a microprocessor, we also experimented with three micro-benchmarks on a pipelined five-stage DLX processor. We report performance and fault coverage results for both SEM and STEM techniques over the baseline TMR technique.

The remainder of this paper is organized as follows. In Section II, we describe our soft error mitigation technique and recovery mechanism. Section III describes how both timing error and soft error are concurrently detected and corrected. Section IV, we discuss the issues in designing a pipeline system with our proposed soft/timing error mitigation technique. We present our results in Section V. Section VI concludes the paper.

II. Soft Error Mitigation

Prior soft error mitigation techniques at the circuit level are either based on temporal redundancy, spatial redundancy or a combination of both. All these techniques strive to achieve high degree of fault coverage, whilst degrading or trading performance, silicon area and other resources. For example, in [3], a specific design of a voting mechanism based on temporal triple modular redundancy is discussed, which mitigates all single event upsets. However, the overhead incurred is very high, as the operating frequency of a system built with such fault mitigation scheme must include the delays of combinational logic blocks, phase shifts of the clocks and the delay incurred by the voter. In this section, we present a variant of this scheme, and show that with a combination of local and global recovery, we can remove the additional overhead imposed on the system operating frequency by the fault mitigation scheme.

The intent of our scheme is to make systems operate at frequencies same as that of non fault tolerant designs, by unloading the fault mitigation overhead from the worst case timing delay estimation. To keep the overhead of error detection and recovery off the critical path, we present the following redundancy organization using our Soft Error Mitigation, SEM, cells. Figure 1 (a) shows a gate-level embodiment of a SEM cell.

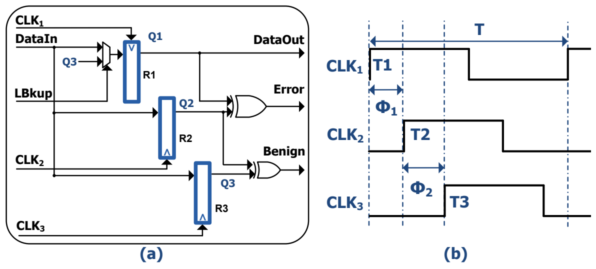

Fig. 1. SEM Cell

TABLE I
Possible Soft Error Scenarios

Case	R_1	R_2	R_3	Error	Benign	Recovery
I	$\surd$	$\surd$	$\surd$	0	0	No Recovery required
II	$\times$	$\surd$	$\surd$	1	0	Load R_2 or R_3 into R_1
III	$\surd$	$\times$	$\surd$	1	1	No Recovery required
IV	$\surd$	$\surd$	$\times$	0	1	No Recovery required

A. Timing Constraints :

Figure 1 (b) shows the timing relationship between the clock signals and the data sampling intervals. Clock signals, CLK_1, CLK_2 and CLK_3, have the same frequency, but they are out-of-phase by an amount governed by the timing constraints, explained below. Contamination delay (T_{CD}) is the minimum amount of time beginning from when the input to a logic becomes stable and valid to the time that the output of that logic begins to change. Propagation delay (T_{PD}) refers to the maximum delay of the circuit, under worst case conditions. T_{PW} is the soft error/noise pulse width.

$$\Phi_1 = T_2 - T_1 \geq T_{PW} \tag{1}$$

$$\Phi_2 = T_3 - T_2 \geq T_{PW} \tag{2}$$

Equations (1) and (2) ensure that registers R_1, R_2 and R_3 are not corrupted by the same soft error. Since the system is running with CLK_1, data is forwarded to later stages after latching in register R_1, and later stages can start the computation speculatively after this time period. Short paths present in the combinational circuit can corrupt the data before it gets latched in registers R_2 and R_3. Consequently, it is required to constrain short paths so that relevant data registered in R_1 is also latched in registers R_2 and R_3 during no error scenarios. Equation (3) ensures that this condition is met by increasing the contamination delay above the desired combined phase shift values, given by Φ_1 and Φ_2. Equation (4) makes sure that temporal sampling happens only after the computation by the combinational logic is completed. This technique is capable of detecting all SEUs happening on registers, and all SETs having pulse duration less than T_{PW}.

$$T_{CD} \geq \Phi_1 + \Phi_2 \tag{3}$$

$$T \geq T_{PD} \tag{4}$$

B. Soft Error Detection and Recovery :

Table I presents the possible soft error scenarios that a SEM technique is capable of detecting and recovering from. The table also lists the corresponding recovery mechanism used. Once the data is latched in registers R_1, R_2 and R_3, they are compared with each other as shown in Figure 1 to produce $Error$ and $Benign$ signals. This comparison operation will complete the voting process required to mitigate soft errors. Below, we explain the recovery mechanism used in case of an error situation.

- Case I : No soft error occurs. Data latched in all three registers are correct. Both $Error$ and $Benign$ signals stay low, and no recovery mechanism is triggered. System operation continues without any interruption.
- Case II : A soft error corrupts the data latched in register R_1. $Error$ signal goes high after the data is latched in R_2. Since the next stage speculatively uses the data forwarded from R_1, re-computation is required next cycle to ensure functional correctness. The data stored in registers R_2 and R_3 are unaffected by the soft error. During the next cycle, value stored in R_2 or R_3 is loaded back into register R_1 with the help of the control signal $Load\_Backup$, completing the local recovery process. Figure 1 shows R_3 being loaded into R_1. Global recovery in the form of stall signal sent to all other SEM cells, unaffected by soft error, is initiated and completed in one cycle.
- Case III : A soft error corrupts the data latched in register R_2. Both the signals, $Error$ and $Benign$, go high once temporal data sampling is completed. This is a false positive scenario. No recovery is required as data forwarded to next stage is correct. System operation is not interrupted.
- Case IV : This represents a case where register R_3 is corrupted with a soft error. In this case $Error$ signal stays low, while $Benign$ signal is asserted high. No recovery and interruption is required in this case too, as $Benign$ signal is high.

As we can see, SEM cell detects and recovers from all possible soft error scenarios. This scheme is well suited for fast transient pulses. Since fast transients typically correspond to soft errors with high strike rate probabilities, SEM cells have near 100% transient fault mitigation capability. On error detection, a single cycle system stall is all that is required for complete recovery. As can be seen, our scheme does not trigger error recovery for false positive scenarios. Also, since the data latched in R_1 is speculatively used, by the succeeding stages, as soon as it is available, the error detection overhead is removed from normal system operation. This is also a low overhead solution, as it shuns the need for system to checkpoint at regular time intervals. Thus, we enable systems to mitigate soft errors, using SEM cells, without any loss of performance, compared to a non fault tolerant design.

III. Soft Error Mitigation in Aggressive Designs

Aggressive designs are based on the philosophy that it is possible to go beyond worst case limits to achieve best performance by not avoiding, but detecting and correcting a modest number of timing errors. In this section, we further investigate our SEM cell design and explain how it can be modified for soft error mitigation in aggressive designs, which uses reliable overclocking technique for improving system performance. The proposed Soft and Timing Error Mitigation, STEM, cell is similar to the SEM cell in area complexity. However, the error detection and recovery mechanism is significantly different to address the requirements of concurrent soft and timing error mitigation.

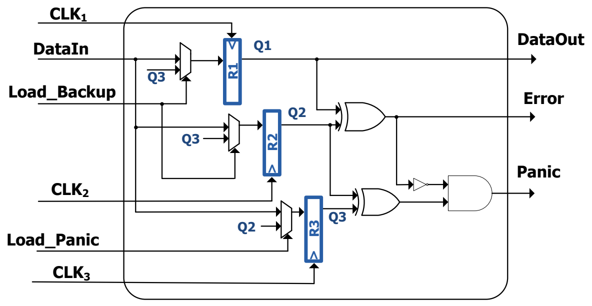

Fig. 2. STEM Cell

A. Error Detection :

Figure 2 shows a gate-level embodiment of a STEM cell, which acts as a on-line-fault monitor for soft and timing error mitigation. The working of a STEM cell is as follows:

Once the data is latched in registers R_1 and R_2, they are compared with each other. This comparison operation completes the timing error detection process, since R_2 is timing safe [8], [10]. But in the presence of soft errors, this comparison operation presents an ambiguous situation, as it is not possible to distinguish which one of these two registers is corrupted by an erroneous value. Also, value in R_2 is not to be trusted during the error recovery process.

If the comparison between R_1 and R_2 flags a mismatch, register R_3 is shielded from the incoming data value, and its content is used to recover the system state. This is done because any soft error that happens after comparing R_1 and R_2 has the potential to corrupt R_3 and push the systems into an unrecoverable state. Only when there is no mismatch between registers R_1 and R_2, register R_3 is allowed to latch the data safely. However, we have not yet ascertained whether R_3 is free from soft error. Therefore, we perform another comparison operation to complete the error detection process. After register R_3 is updated, we compare it with register R_2, to detect any error happening in register R_3. If there is no mismatch, register R_3 is trusted for error recovery purposes. If they mismatch that represents a case where register R_3 is corrupted by a soft error. At this point, it is possible to say that data latched in registers R_1 and R_2 are uncorrupted. The

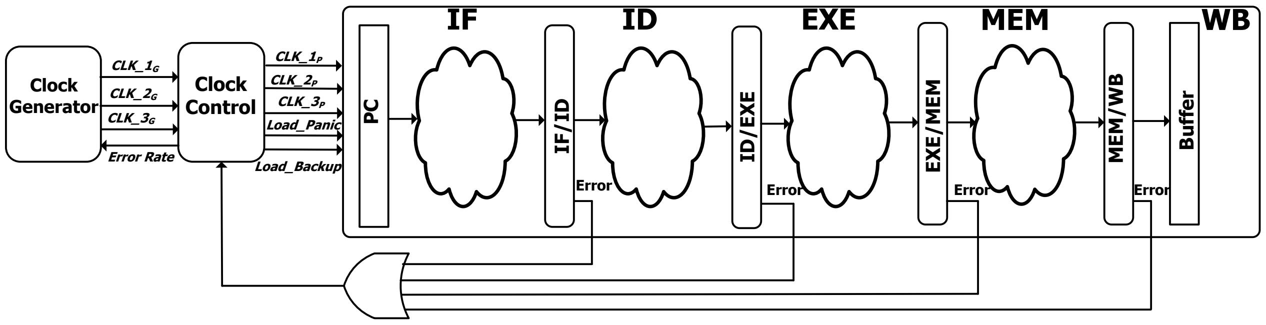

Fig. 3. Pipeline Design with STEM Cells

system is stalled for one cycle for flushing out the erroneous value from R_3, and loading either R_1 or R_2 value into R_3.

B. Timing Constraints :

As is the case with SEM cells, STEM cells also require strict timing constraints, to detect and correct soft and timing errors. STEM cells must satisfy Equations (1), (2) and (3). Equation (4) is modified as shown in Equation (5) for STEM cells. Equations (1) and (2) ensure that registers present in a STEM cell are not corrupted by the same SET. Equations (3) and (5) ensure that data latched in registers R_2 and R_3 are timing correct, i.e. free from timing errors. The timing relationships shown in Figure 1 (b) still holds, with the caveat that Φ_1 also includes the extent of overclocking that is possible every cycle.

$$T + \phi_1 \geq \ T_{PD} \tag{5}$$

C. Error Recovery :

Table II lists all possible error scenarios with corresponding recovery mechanisms. In the table, NE represents No Error; SE represents Soft Error and TE represents Timing Error. In the following discussion, we explain the various possible events that take place in the STEM cell, and the associated recovery mechanism that is used in case of an error. It employs either a single cycle or three cycle fast local recovery based on the values of $Error$ and $Panic$ signals, shown in Figure 2.

TABLE II
Possible Error Scenarios

Case	R_1	R_2	R_3	Error	Panic	Recovery
I	NE	NE	NE	0	0	No Recovery
II	SE	NE	NE	1	0	Load R_3 into R_1, R_2
III	NE	SE	NE	1	0	Load R_3 into R_1, R_2
IV	NE	NE	SE	0	1	Load R_2 into R_3
V	TE	NE	NE	1	0	Load R_3 into R_1, R_2
VI	TE	SE	NE	1	0	Load R_3 into R_1, R_2

- Case I: No error case. Both signals, $Error$ and $Panic$, stay low. System moves forward without any interruption
- Case II, III, V, VI : This represents a case where one of the registers R_1 or R_2 is corrupted. In this case, $Error = 1$ and $Panic = 0$. In this scenario R_3 is not updated, and the system recovers by loading R_3 in to R_1 and R_2 triggering re-computation. A three cycle global recovery process is initiated, which includes: one cycle stall for loading data back into the registers R_1 and R_2, and two cycles for re-computation. This two cycle re-computation is required, as the error might have occurred because of overclocking, and this error will repeat in R_1, if sufficient time is not given for re-computation. This prevents recurrent system failures.
- Case IV : Only R_3 is corrupted. In this case, $Error = 0$ and $Panic = 1$. No re-computation is required. However, it is necessary to flush the erroneous data from R_3, to facilitate error recovery in subsequent cycles. As data in only R_3 is corrupted, "golden" data present in R_2 is loaded in to R_3. This requires a single cycle system stall, during which erroneous STEM cells perform a local correction.

As is seen, STEM cell detects and recovers from all possible soft and timing error scenarios, wherein the soft error is only of type SET. Our technique leads to silent data corruption, if an SEU happens in R_3. However, since register R_3 is only used as a checkpointing register, a corrupted R_3 value may lead to failure, only if an error occurs in R_1 or R_2 in the next cycle. The possibility of a system failure using our fault tolerance scheme is heavily mitigated. For Case VI, we expect that a TE or SE affects several STEM cells, and the possibility of all cells having a TE in R_1 and SE in R_2 is insiginificant. Hence, we hope one of the STEM cells will have the error signal triggered, preventing R_3 of all STEM cells from being loaded. If $Error = 1$, then we do not look at $Panic$ signal.

IV. Pipeline Design

The basic step in using SEM or STEM cells in a pipeline is to replace all pipeline registers with one of them. Input clocks are to be constrained in a way, so as to provide fault tolerance capabilities to the pipeline from soft error and timing error, if STEM is the cell of choice. In this section, our discussion is based on the use of STEM cells in place of pipeline registers. Using SEM cells follow straight forward.

Figure 3 illustrates how STEM cells are integrated into a processor pipeline. The figure depicts the data and control flow for a five-stage pipeline processor. To the last stage of the pipeline, which is writeback (WB), an extra write buffer, is added. This is to ensure that data written to the register file or memory is always free from timing errors. Every pipeline stage register is replaced with STEM cells, except for the write buffer registers. All error signals from a pipeline stage are logically OR-ed to generate the stage error for that pipeline stage. Global error signal is generated from all pipeline stage error signals, by combining them using another "OR" function. Here, we explain the pipeline operation for $Error = 1$ and $Panic = 0$ (Case II, III, V, VI), as this is the most complicated case. Once an error is detected in any one of the pipeline stages, global error is asserted, and in every stage of the pipeline, registers R_3 of the STEM cells are not updated with the incoming data. In the next clock cycle, load backup is asserted, and in each STEM cell, content of register R_3 is loaded into corresponding R_1 and R_2 registers. After this, clock to the pipeline is stalled for two cycles, completing the error recovery process.

A key factor that limits frequency scaling is error rate. As frequency is scaled up, the number of input combinations that result in delays greater than the new clock period also increases. The impact of error rate on frequency scaling is analyzed as below:

Let t_{wc} denote the worst case clock period. Let t_{ov} denote the clock period after overclocking the circuit. Let n be the number of cycles needed to recover from an error. Let us assume that a particular application takes N clock cycles to execute, under normal conditions. Let t_{diff} be the time difference between the original clock period and the new clock period. Then the total execution time is reduced by $t_{diff} \times N$, if there is no error. Let us assume that the application runs at an overclocked frequency of period t_{ov} with an error rate $k\%$. To achieve any performance improvement at this frequency, Equation (6) must be satisfied. It states that even after accounting for error penalty, execution time required is still less than the worst case frequency operation.

$$N \times t_{ov} + n \times N \times k \times t_{ov} < N \times t_{wc} \tag{6}$$

$$k \; < \; \frac{(t_{wc} - t_{ov})}{n \times t_{ov}} \tag{7}$$

For STEM cells, an error can happen in five different scenarios, as mentioned in Table II, and also error penalty paid is not the same for all the cases. If we assume all these error scenarios are equally likely, then the average error penalty paid in cycles is: $n = \frac{4\times3+1\times1}{5} = 2.6$. According to Equation (7), for a frequency increase of 15%, the error rate must not be higher than 5.76%, for STEM cells to yield no performance improvement. For error rates less than *1%*, a frequency increase of *2.6%* is enough for STEM cells to have a performance improvement over non fault tolerant designs.

V. Experimental Results

In this section, we present our results based on the experiments conducted on a two stage arithmetic pipeline and a five stage DLX in-order pipeline processor, where pipeline registers are augmented with our fault detection and correction circuitry. Arithmetic pipeline performs a 64-bit addition in the first stage and a 32-bit multiplication in the second stage. Adder output is fed to the multiplier as multiplicand and multiplier. RTL level models are developed and are synthesized using $45nm$ OSU standard cell library [11]. Timing-annotated gate level simulations are then carried out by extracting timing information in standard delay format(SDF), and back annotating them on the design. From the static timing analysis reports, we estimated the value of worst-case frequency, T_{Max} = $9ns$. Pulses of varying width ranging from $500ps$ to $900ps$ are injected in the pipeline. Each cycle results are checked for correctness after the computation is done to ensure that the recovery mechanism works. Whenever recovery is activated, occurrence of an error is logged. All our experiments are performed for a set error rate target of 1% over 10000 cycles.

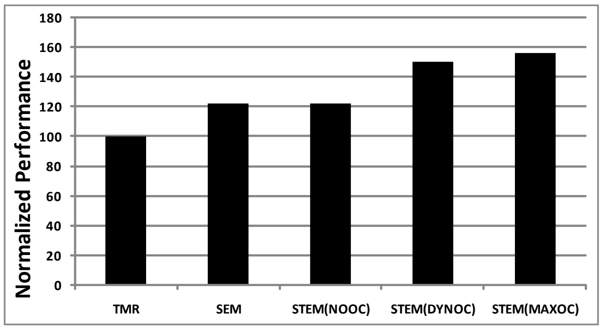

Fig. 4. Normalized Arithmetic Pipeline Execution time

During run time, the number of errors that happened during a sampling interval is communicated to the clock controlling unit at the end of each interval. The clock controlling unit makes a decision based on the error rate, during the previous sampling interval, and set target error rate. We considered a linear control scheme for clock frequency switching between worst-case frequency, T_{Max} and highest frequency, T_{Min}. For our design, T_{Min} is set at $7ns$. This range is divided into 32 steps, and if the error rate is less than 1%, clock frequency is increased by one step size, otherwise it is decreased by one step size. We initialized the pipeline with different seeds, and the fault injection results are presented in Table III for three different runs. We ran our experiments for $3ms$ duration.

We configured the arithmetic pipeline designed with STEM cells to operate in three different modes. They are no overclocking (NOOC), wherein $T_{Max} = 9ns$ and $T_{Min} = 9ns$, maximum overclocking (MAXOC), wherein $T_{Max} = 7ns$ and $T_{Min} = 7ns$, and dynamic overclocking (DYNOC), wherein $T_{Max} = 9ns$ and $T_{Min} = 7ns$. For TMR system, worst-case frequency, T_{Max}, is set at $11ns$. Performance improvements offered by both SEM scheme and different modes of STEM is shown in Figure 4. From this, we can see that dynamic overclocking(DYNOC) mode offers 49% over TMR while maximally overclocked (MAXOC) mode offers 55%. Performance of no overclocking (NOOC) mode is comparable to that of SEM and SEM offers 23% performance

TABLE III
FAULT INJECTION RESULTS FOR ARITHMETIC PIPELINE

	STEM(MAXOC)			STEM(DYNOC)			STEM(NOOC)		SEM		TMR	
	TE	Transient Faults		TE	Transient Faults		Transient Faults		Transient Faults		Transient Faults	
		Injected	Detected		Injected	Detected	Injected	Detected	Injected	Detected	Injected	Detected
RUN1	14	2031	432	14	2033	421	2030	325	2031	334	2026	256
RUN2	14	2031	450	12	2033	414	2025	315	2028	323	2026	268
RUN3	14	2031	449	15	2032	424	2025	307	2030	311	2034	273

improvement over TMR. From Table III, we can see that fault masking rate is high in TMR design when compared with SEM and STEM designs. This is because, its operating frequency includes the phase shifts of the clocks and voter delay. Hence, TMR operates with longer clock period compared to SEM and STEM, resulting in more SET pulses attenuating before the latching window.

We also simulated three different micro benchmarks to evaluate the performance improvement and fault coverage of both SEM and STEM on a five stage in-order pipelined processor. This processor, implemented in $45nm$ technology, is based on the DLX instruction set architecture. First application, RandGen, calculates a simple random number generation to give a number between 0 and 255. The MatrixMult application multiplies two 50x50 integer matrices and the BubbleSort program implements bubble sort algorithm on 5,000 half-word variables. Here, we followed the same fault injection strategy and clock control used for two stage arithmetic pipeline. For each benchmark, processor state has been checked to verify the correctness of the computed results after simulation.

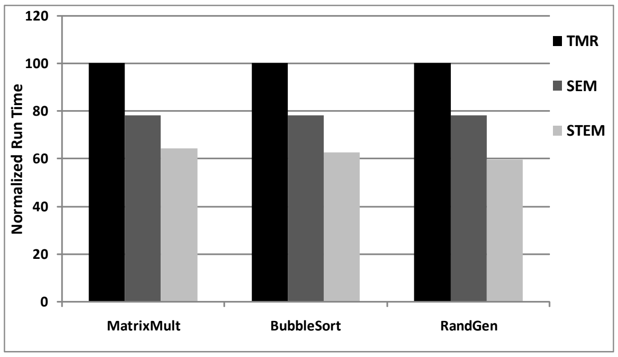

Fig. 5. DLX Execution time for various benchmarks

From timing reports, the worst case frequency, T_{Max}, is estimated as $6ns$. Contamination delay is increased by $2ns$ and the system operates at $4ns$ (optimal frequency). Area overhead incurred is less than 5% for the processor because significant area consumption of the system comes from the memory system. The results for the three different benchmarks are presented in Figure 5, showing relative execution times for conventional TMR, SEM and STEM schemes. From this, we found that SEM offers 26.58% performance improvement over TMR and STEM offers 27.42% over SEM.

VI. CONCLUSIONS

In this work, we developed two efficient soft error mitigation schemes, which remove the error detection and correction overhead from the circuit critical path. One of our schemes is capable of tolerating timing errors as well. Both the schemes tolerate fast transient noise pulses, which is the principal characteristic of SETs. Our schemes have no significant performance overhead during error free operation. One of the salient features of our approach lies in the capability to trigger recovery immediately on error detection, without requiring any check pointing. In the future, we will implement our fault mitigation schemes in complex pipelined systems, and evaluate the fault coverage and performance for more representative benchmarks.

ACKNOWLEDGMENT

The research reported in this paper is partially supported by NSF grant number 0311061, Information Infrastructure Institute (iCUBE) and the Jerry R. Junkins Endowment at Iowa State University.

REFERENCES

[1] B. Narasimham *et al.*, "Characterization of digital single event transient pulse-widths in 130-nm and 90-nm cmos technologies," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 6, pp. 2506–2511, Dec. 2007.
[2] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *DSN*, June 2002, pp. 389–398.
[3] D. Mavis and P. Eaton, "Soft error rate mitigation techniques for modern microcircuits," *Reliability Physics Symposium Proceedings, 2002. 40th Annual*, pp. 216–225, 2002.
[4] S. R. Nassif, "Modeling and forecasting of manufacturing variations," in *ASP-DAC*, January 2001, pp. 145–149.
[5] B. Colwell, "The zen of overclocking," *IEEE Compututer*, vol. 37, no. 3, pp. 9–12, March 2004.
[6] T. Austin, V. Bertacco, D. Blaauw, and T. Mudge, "Opportunities and challenges for better than worst-case design," in *ASP-DAC*, vol. 1, January 2005, pp. 2–7.
[7] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: A low-power pipeline based on circuit-level timing speculation," in *IEEE Micro*, 2003, pp. 7–18.
[8] V. Subramanian, M. Bezdek, N. D. Avirneni, and A. Somani, "Superscalar processor performance enhancement through reliable dynamic clock frequency tuning," in *DSN*, June 2007, pp. 196–205.
[9] V. Subramanian and A. Somani, "Conjoined pipeline: Enhancing hardware reliability and performance through organized pipeline redundancy," in *PRDC*, Dec 2008.
[10] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "A self-tuning dvs processor using delay-error detection and correction," *IEEE Journal of Solid-State Circuits*, vol. 41, no. 4, pp. 792–804, April 2006.
[11] J. S. et al, "FreePDK: An Open-Source Variation-Aware Design Kit," in *Proc. of the 2007 IEEE Intl Conference on Microelectronic Systems Education*, 2007, pp. 173–174.