Low Overhead Soft Error Mitigation Techniques for
High-Performance and Aggressive Systems

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

Abstract

The threat of soft error induced system failure in high performance computing systems has become more prominent, as we adopt ultra-deep submicron process technologies. In this paper, we propose two techniques, namely Soft Error Mitigation (SEM) and Soft and Timing Error Mitigation (STEM), for protecting combinational logic blocks from soft errors. Our first technique (SEM), based on distributed and temporal voting of three registers, unloads the soft error detection overhead from the critical path of the systems. Our second technique (STEM) adds timing error detection capability to guarantee reliable execution in aggressively clocked designs that enhance system performance by operating beyond worst-case clock frequency. We also present a specialized low overhead clock generation scheme that ably supports our proposed techniques. Timing annotated gate level simulations, using 45nm libraries, of a pipelined adder-multiplier and DLX processor show that both 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 conventional triple modular redundancy voter based soft error mitigation scheme, while STEM outperforms SEM by 27.42%.

Keywords: Parameter Variations, Soft Error, Dependable and Adaptive Systems, Overclocking

1. Introduction

Nano-sized transistors, coupled with deployment in hazardous environments, have magnified the reliability concerns plaguing modern computing systems. Rapid enhancements in VLSI technology have fueled the increasing apprehension of system hardware being susceptible to myriad of faults. Many fault tolerance techniques are proposed at different levels of design hierarchy, starting from the design of hardened latches to system-level fault tolerant architectures [1, 2, 8, 11]. All these techniques strive to provide high degrees of fault coverage by providing redundancy in either information, spatial or temporal domains.

In the past, single event upsets (SEUs) were a major concern only in space applications, creating hard threats like loss of control, and 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 SETs have widths in the range of 500ps to 900ps in the 90nm process, as compared to 400ps to 700ps in the 130nm process [13]. As a result, terrestrial applications also require fault tolerance techniques to ensure their dependability.

In current and future technologies, the problem of soft errors in combinational circuits is becoming comparable to that of unprotected memory elements [14]. 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 temporal triple modular redundancy (TMR) and majority voting have been proposed to mitigate soft error rate (SER) in logic circuits [10]. 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, which require safety margins to be added to individual system components to address parameter variations that include intra-die and inter-die process variations, and environmental variations, which include temperature and voltage variations [4, 5]. 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 that lead to system failure may happen.
Overclocking, as a means to improve performance, is a popular technique among high-performance enthusiasts [6]. Microprocessor vendors are even introducing overclocking capabilities in their chipsets; examples being AMD’s Overdrive and Advanced Clock Calibration techniques. Circuits exhibit worst-case delay only when their longest delay paths are exercised by the inputs. However, these worst-case delay inducing inputs and operating conditions are rare, leading to room for performance improvement that overclockers exploit [3]. The problem is that timing errors occur at overclocked speeds and may lead 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 [16], it has been shown that the operating frequency can be increased reliably beyond the worst-case limit; allowing systems to operate at an optimal overclocked 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% [17].

Safety critical systems with hard real-time constraints require wide fault coverage with no compromise in performance. An interesting capability in nanometer design space, we believe, is to provide soft error tolerant reliable execution for high performance aggressive designs. In this paper, we propose new ways of designing fault tolerant and reliably overclocked register cells that enable systems to improve both their performance and dependability.

1.1. Our Contribution

In this paper, we address the issue of soft errors in random logic and develop solutions that provide fault tolerance capabilities without requiring 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. SEM allows systems to operate without the overhead of soft error detection circuitry. Our second technique, STEM, concurrently detects and corrects soft and timing errors using Soft and Timing Error Mitigation (STEM) register cells. STEM cells have soft error mitigation capabilities comparable to those of SEM cells, and they also support reliable overclocking. Both of our techniques employ a distributed and temporal voting scheme that enables 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 when an error happens we void the computation and perform re-computation.

Both SEM and STEM cells require three clocks for proper operation. Clock distribution and routing are significant challenges in nanoscale technologies. Clock distribution network (CDN) consumes a significant portion of the power, area and metal resources in an integrated circuit. As a consequence, a specialized clock generation, distribution and routing scheme that minimizes the clock overhead incurred by our fault mitigation techniques is important. Also, to support reliable dynamic overclocking, as discussed in [16], it is important to precisely control the relative phase shifts of clock signals at high frequencies. Therefore, we focus on developing an efficient local clock manager (LCM), which helps in generating the required clock signals, with the desired phase shifts, locally. The clocks, so generated and distributed, satisfy the timing constraints required for proper working of our techniques. We also analyze the area overhead incurred for developing such LCMS.

For our initial experimental study, we integrated our data sampling mechanisms into a two stage pipeline consisting of an adder and a multiplier. Our results show that, with STEM cells, performance of this system can be increased by 55.93% over conventional TMR schemes, while providing near 100% fault coverage. In order to fully understand the performance improvement and fault coverage that our schemes can provide to a microprocessor, we experimented with three micro-benchmark applications on a DLX processor. For the processor, our results show that SEM technique achieves an average performance improvement of 26.58% over the TMR scheme and STEM outperforms SEM by 27.42%, while providing near 100% fault coverage.

The remainder of this paper is organized as follows. In Section 2, we describe our soft error mitigation technique and recovery mechanism. Section 3 describes how both timing error and soft error are concurrently detected and corrected. In Section 4, we discuss the issues in designing a pipeline system with our proposed soft/timing error mitigation techniques. Section 5 discusses the implementation and area overheads of implementing a local clock manager. We present our results in Section 6. Section 7 presents the related work and Section 8 concludes the paper.

2. 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. These techniques achieve high degree of fault coverage, whilst degrading or trading performance, silicon area and other resources. For example, in [10], 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, by the fault mitigation scheme, on the system operating frequency.

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 circuit 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. It consists of three registers $R_1$, $R_2$ and $R_3$, clocked by clock signals $CLK_1$, $CLK_2$ and $CLK_3$, respectively. Data is sampled at three different time intervals $T_1$, $T_2$ and $T_3$, and are stored in registers $R_1$, $R_2$ and $R_3$, respectively.

**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. Data is stored in registers at the rising edge of the clock signals, and strict timing constraints are required for efficient mitigation of soft errors. 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.

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 at $CLK_1$ frequency, data is forwarded speculatively to subsequent stages after latching in register $R_1$, and subsequent stages start their computation immediately.

$$
\Phi_1 = T_2 - T_1 \geq T_{PW} \quad (1)
$$

$$
\Phi_2 = T_3 - T_2 \geq T_{PW} \quad (2)
$$

Short paths present in the combinational circuit may corrupt the data before it gets latched in registers $R_2$ and $R_3$. Consequently, it is required to constrain short paths so that the same data registered in $R_1$ is also latched in registers $R_2$ and $R_3$, during error-free operation. Equation (3) ensures that this condition is met, by increasing the contamination delay above the desired combined phase shift values, given by $\Phi_1$ and $\Phi_2$. Equation (4) makes sure that temporal sampling happens only after the computation by the combinational logic is done. Our technique is capable of detecting all SEUs happening on registers, and all SETs having pulse duration less than $T_{PW}$.

$$
T_{CD} \geq \Phi_1 + \Phi_2 \quad (3)
$$

$$
T \geq T_{PD} \quad (4)
$$

**SOFT ERROR DETECTION AND RECOVERY:** Table 1 presents the possible soft error scenarios that a SEM technique is capable of detecting and recovering from. The table also lists the corresponding recovery mechanisms 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(a) to produce ERROR and BENIGN signals. This comparison operation completes the voting process required to detect soft errors. On error detection, a single cycle system stall is all that is required for complete recovery. Below, we explain the different possible scenarios and the recovery mechanism used when an error happens.

- **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 LBKUP, completing the local recovery process. Figure 1 (a) shows $R_3$ being loaded into $R_1$. Global recovery, in the form of a stall signal sent to all other SEM cells that are unaffected by the 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 the 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 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 not incurred during normal system operation. This is also a low overhead solution, as it shuns the need for check pointing 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.

**Fault Tolerance Analysis:** The SEM technique detects and recovers from all possible soft error scenarios involving both SEUs and SETs. This scheme is well suited for fast transient pulses. Since fast transients typically correspond to soft errors with high spike rate probabilities, SEM cells have near 100% transient fault mitigation capability. Our scheme offers protection for pulses of widths less than the phase shifts provided between the clock signals. Any noise signal, whose pulse width exceeds this limit, cannot be detected by our scheme.

**3. 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 the solution presented in previous section for soft error mitigation, and explain how it can be modified for soft error mitigation in aggressive designs, which uses reliable overclocking technique for improving system performance. With a conventional voter design, to detect and correct $n$ errors simultaneously, we need to have up to $2n + 1$ data samples. In our case, we have $n = 2$, since we need to detect and correct both soft and timing errors. For this analysis, we consider soft errors to be of only type SET. A traditional fault tolerance technique requires five different data values for guaranteeing both soft error and timing error detection and correction. The overhead incurred by this approach is very high as it increases the number of registers by four times, and requires five different clocks to sample data at five different times. Our goal is to develop a soft and timing error mitigation scheme that incurs minimal overhead. 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.

**Error Detection:** Figure 2 shows a gate-level embodiment of a STEM cell, which acts as an 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 [7, 16]. 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 system 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, then 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 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 $R_3$.

**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 $\Phi_1$ also includes the extent of overclocking that is possible every cycle.

\[
T + \phi_1 \geq T_{PD} \tag{5}
\]

**Error Recovery:** Table 2 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 \( \text{ERROR} \) and \( \text{PANIC} \) signals, shown in Figure 2.

### Table 2: Possible Error Scenarios

<table>
<thead>
<tr>
<th>Case</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
<th>( \text{ERROR} )</th>
<th>( \text{PANIC} )</th>
<th>Recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>NE</td>
<td>NE</td>
<td>NE</td>
<td>0</td>
<td>0</td>
<td>No Recovery</td>
</tr>
<tr>
<td>II</td>
<td>SE</td>
<td>NE</td>
<td>NE</td>
<td>1</td>
<td>0</td>
<td>Load ( R_3 ) into ( R_1 ), ( R_2 )</td>
</tr>
<tr>
<td>III</td>
<td>NE</td>
<td>SE</td>
<td>NE</td>
<td>1</td>
<td>0</td>
<td>Load ( R_3 ) into ( R_1 ), ( R_2 )</td>
</tr>
<tr>
<td>IV</td>
<td>NE</td>
<td>SE</td>
<td>SE</td>
<td>0</td>
<td>1</td>
<td>Load ( R_2 ) into ( R_1 )</td>
</tr>
<tr>
<td>V</td>
<td>TE</td>
<td>NE</td>
<td>NE</td>
<td>1</td>
<td>0</td>
<td>Load ( R_3 ) into ( R_1 ), ( R_2 )</td>
</tr>
<tr>
<td>VI</td>
<td>TE</td>
<td>SE</td>
<td>SE</td>
<td>1</td>
<td>0</td>
<td>Load ( R_3 ) into ( R_1 ), ( R_2 )</td>
</tr>
</tbody>
</table>

- **Case I**: No error case. Both signals, \( \text{ERROR} \) and \( \text{PANIC} \), stay low. System operation is not interrupted.

- **Case II, III, V, VI**: This represents a case where one of the registers \( R_1 \) or \( R_2 \) is corrupted. In this case, \( \text{ERROR} = 1 \) and \( \text{PANIC} = 0 \). 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 \( R_2 \), using \( \text{LBKUP} \) signal, 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, \( \text{ERROR} = 0 \) and \( \text{PANIC} = 1 \). 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 all STEM cells perform a local correction, using \( \text{LPANIC} \) signal.

**Fault Tolerance Analysis**: As is seen, the STEM technique detects and recovers from all possible soft and timing error scenarios, wherein the soft error is only of type SET. Also, the case where \( \text{ERROR} = 1 \) and \( \text{PANIC} = 1 \) never happens by design.

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. Consequently, the possibility of a system failure because of a SEU in \( R_3 \) 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 insignificant. 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 \( \text{ERROR} = 1 \), then we do not look at \( \text{PANIC} \) signal. The fault coverage is similar to that of the SEM technique, except that in case of false positives, we still need to take appropriate corrective action. In case of the SEM scheme, this value will be overwritten, as \( R_3 \) is used only for error detection. However, the STEM technique allows reliable overclocking, achieving higher performance than those systems incorporated with SEM cells.

### 4. Pipeline Design

The basic step in using SEM or STEM cells in a pipeline is to replace all pipeline registers with either 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, as well as, timing error when 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 STEM cells follow straightforward.

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, \( \text{ERROR} \), is generated from all pipeline stage error signals, by combining them using another “OR” function. Similarly, global \( \text{LPANIC} \) signal is generated from individual \( \text{PANIC} \) signal from all STEM cells. Timing errors may occur once the operating frequency exceeds the worst-case frequency estimate. As explained in the previous section, our data latching scheme of STEM cell guarantees sufficient time before latching values in registers \( R_2 \) and \( R_3 \). However, data latched in all three registers are susceptible to soft errors that are uniformly distributed in time and space.

Here, we explain the pipeline operation for \( \text{ERROR} = 1 \) and \( \text{PANIC} = 0 \) (Case II, III, V, VI), as this is the most complicated case. Once an error is detected in any one of the pipeline stages, the global error signal 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, the load backup signal, \( \text{LBKUP} \), is asserted, and in each STEM cell, the content of register \( R_3 \) is loaded into corresponding \( R_1 \) and \( R_2 \) registers. After this, the clock to the pipeline is stalled for two cycles, completing the error recovery process.

**Error Recovery**: In the following discussion, we present the error recovery scheme in detail for a pipeline using STEM cells. Various events involved in the recovery process are illustrated with the help of a timing diagram. Figure 4 shows how our global error recovery scheme rescues a system when an error happens. The figure also de-
picts the timing relationship between various control signals. As mentioned, the global recovery takes three clock cycles and the following description explains the events that take place during the whole process.

![Timing Diagrams](image)

**Figure 3: Pipeline Design with STEM Cells**

**Figure 4: Timing Diagrams**

Figure 4 shows a set of clock signals, CLK$_{1G}$, CLK$_{2G}$ and CLK$_{3G}$, that are generated from the main clock signal, CLK, using a LCM. Next, it shows a set of clock signals, CLK$_{1P}$, CLK$_{2P}$ and CLK$_{3P}$, that are routed to the pipeline. These clock signals, which are gated versions of CLK$_{1G}$, CLK$_{2G}$ and CLK$_{3G}$ respectively, are stalled in a manner that enables the pipeline to recover from different error scenarios. Signal ERROR$_{N}$ indicates an error happening in the pipeline stage $N$. Error signals from all the pipeline stages are OR-ed together to generate the global error signal, GERROR, which is latched in the clock control unit. Once an error is detected, the very next clock edge of clock signal CLK$_{3G}$ is gated and in the next cycle, LBKUP signal is asserted high for one clock cycle. In the same clock cycle, using CLK$_{1P}$ and CLK$_{2P}$, recovery data from register $R_3$ is loaded back into registers $R_1$ and $R_2$. During the next clock cycle, all clock signals, CLK$_{1G}$, CLK$_{2G}$ and CLK$_{3G}$, are clock gated to give the pipeline sufficient time for recomputation. Clock gating is achieved through control signals CLKSTALL$_{12}$ and CLKSTALL$_{3}$, which are generated by the clock control unit.

To illustrate our error recovery mechanism, an error occurrence is highlighted in Cycle 3. The error occurs during the execution of INST 1 of pipeline stage $N$. This event triggers the error recovery mechanism that spans from Cycle 4 to Cycle 6. During Cycle 4, data is loaded into register $R_1$ and $R_2$ from the corresponding stage golden register $R_3$. Pipeline is allowed to perform the computation during Cycles 5 and 6. Results are again checked at the end of Cycle 6. Since no error is detected in this cycle, normal pipeline operation resumes. From the waveforms, we can see that on an error detection, the entire pipeline goes back by one instruction. Similar recovery actions are performed for a panic situation and it involves stalling the clock signals CLK$_{1G}$ and CLK$_{2G}$ for just one cycle. In this case, the pipeline does not roll back and just the corresponding stage $R_3$ register is updated.

**Dynamic Frequency Scaling:** In the following discussion, we derive the limits of frequency scaling within which a system integrated with STEM cells operates reliably. Pipeline starts execution with a minimal phase shift required between the clocks, and the clock frequency is gradually increased, while satisfying the error rate constraint. To support reliable dynamic overclocking, certain governing conditions need to be met at all times, during pipeline operation. Let us assume that the pipeline operates reliably between the clock frequencies, $F_{MIN}$ and $F_{MAX}$, governed by time periods, $T_{MAX}$ and $T_{MIN}$ respectively. $T_{MAX}$ is estimated by the worst-case design settings, and is equal to worst-case clock period, $T_{WC}$. The following clocking constraints decide $T_{MIN}$. Under overclocking conditions, the following constraints must be satisfied for proper error detection and recovery.

Let $D_1$ represent the phase shift that needs to be provided for CLK$_2$, with respect to CLK$_1$, for soft and timing error mitigation, when the system is clocked with clock period $T_{MIN}$. Let $D_2$ represent the phase shift that needs to be provided for CLK$_3$, with respect to CLK$_1$, for proper error recovery, when the system is clocked with clock period $T_{MIN}$. Value of $T_{MIN}$, satisfying Equation (6), corresponds to the maximum frequency at which a system can possibly recover, after a timing error occurs.
A key factor that limits frequency scaling is error rate. As frequency is scaled higher, 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 follows:

Let $t_{\text{wc}}$ denote the worst-case clock period. Let $t_{\text{ov}}$ 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 $t_{\text{diff}}$ be the time difference between the original clock period and the new clock period. Then the total execution time is reduced by $t_{\text{diff}} \times N$, if there is no error. Let us assume that the application runs at the overclocked frequency of period $t_{\text{ov}}$, with an error rate of $k\%$. To achieve any performance improvement at this frequency, Equation (9) must be satisfied. It states that even after accounting for error recovery penalty, execution time required is still less than that required for worst-case frequency operation.

$$N \times t_{\text{ov}} + n \times N \times k \times t_{\text{ov}} < N \times t_{\text{wc}}$$ (9)

$$k < \frac{(t_{\text{wc}} - t_{\text{ov}})}{n \times t_{\text{ov}}}$$ (10)

For the STEM technique, an error can happen in five different scenarios, as mentioned in Table 2, and also the error recovery penalty paid is not the same for all the cases. If we assume that all these error scenarios are equally likely, then the average error penalty in cycles is: $n = \frac{4 \times 3.4 \times 1 \times 1}{5} = 2.6$.

According to Equation (10), for a frequency increase of 15%, the error rate must not be higher than 5.76%, for the STEM technique to yield no performance improvement. For error rates less than 1%, a frequency increase of 2.6% is enough for the STEM scheme to have a performance improvement over non fault tolerant designs.

**OVERHEADS :** One of the main overheads incurred by our schemes is fixing the circuit contamination delay to a required value. Increasing this delay involves rapid increase in silicon area, as buffers need to be inserted in the short circuit delay paths. This problem has to be addressed from different design perspectives that include developing new synthesis algorithms and delay buffer design with minimal area consumption. Both SEM and STEM cells require metastability mitigation circuits, as flip-flops may enter a metastable state when overclocked, or when a soft error reaches the registers during the latching window. We envisage the incorporation of a metastability detection circuit, similar to the one developed in [7].

### 5 Local Clock Generation

Reliable dynamic overclocking technique has been proposed earlier, in [16], to improve system performance by tuning the clock frequency beyond the conservative worst-case clock period. It requires a dynamic phase shift (DPS) between the clock signals to support aggressive dynamic clock tuning. At higher frequencies, controlling the phase shift precisely is a challenge and this often restricts the possible operating frequency configurations. To avoid dynamic phase shift between the clock signals, we incorporate a constant phase shift (CPS) between the clocks that are configured to run between frequencies corresponding to the time periods, $T_{\text{MAX}}$ and $T_{\text{MIN}}$.

Let us consider a case where $T_{\text{MAX}} = 10ns$, $T_{\text{MIN}} = 6ns$ and $T_{\text{CD}} = 4ns$. Considering a dynamic phase shift between the clock signals, when we scale the system clock period down to $6ns$, then we need to provide a phase shift of $2ns$. Similarly a phase shift of $3ns$ is required for a $7ns$ clock period. Since the circuit contamination delay is increased to $4ns$ to aggressively clock the system, computed data will remain stable for $(T + 4)ns$, where $T$ is the current operating frequency of the system. Instead of requiring a dynamic phase shift along with frequency scaling, we provide a constant phase shift of at most $T_{\text{CD}}$ at all times.
Processor pipelines occupy only a specific portion of chip area. Local clock managers (LCMs) as shown in Figure 6 are placed only in the segments of the chip where the processor pipeline is present. Employing CPS, delay values $D_1$ and $D_2$ are set to constant values to satisfy the timing constraint explained earlier in Section 4. This kind of approach saves the amount of clocking resources required for SEM and STEM schemes, and also increase the number of possible operating frequencies available for a given system. With CPS scheme, the clock signals required are derived from a single clock distribution network. Figure 6 shows how CLK1, CLK2 and CLK3 are generated with a H-tree clock distribution network.

**Figure 6: Local clock generation with single clock routing**

**CASE STUDY : Local Clock Generation using buffers.** For generating clock signals required by SEM and STEM schemes locally, we present a possible implementation using buffers. We perform this study using 45nm spice models distributed by Nangate Technologies [12]. Post layout spice models containing parasitic information are used. Area overhead, incurred for generating constant phase shift clocks, is analyzed by applying a load of 128 STEM cells. From this study, we observe that, even for a 2.5 ns phase shift, only 14 clock buffers are needed. This overhead is much lower than a having a second and third clock tree networks. Study results are summarized in Table 3.

<table>
<thead>
<tr>
<th>DELAY(ns)</th>
<th>BUFFERS</th>
<th>DELAY(ns)</th>
<th>BUFFERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>6</td>
<td>1.5</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>2.5</td>
<td>14</td>
</tr>
</tbody>
</table>

**6. Experiments & 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, wherein pipeline registers are augmented with our fault detection and correction circuitry.

**EXPERIMENTAL METHODOLOGY :** To estimate the performance gains and fault tolerant capabilities offered by SEM and STEM techniques, simulations are carried out on a two stage arithmetic pipeline. This circuit 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 for both the circuits, and are synthesized using the 45nm OSU standard cell library [15]. 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.

**Figure 7: Fault Injector Framework**

Figure 7 illustrates our fault injection methodology. The working of our fault injector is as follows: A total of $2^N$ (N being 7 in our experiments) fault injection test nodes that are spread uniformly across the area of the logic circuit are selected. To make sure that our injected fault has indeed produced a SET, we modified the circuit netlist to insert XOR gates at all selected nodes, as shown in Figure 7. If a location $i$ is chosen for fault injection, Inject$_i$ is made high to invert the signal $A$ driven by the fault injection node $i$. Out of $2^N$ locations, one location is chosen randomly for fault injection at a time, by using the output of a $N$-bit random number generator. For our experiments, we used a linear feedback shift register (LFSR) for generating the $N$-bit random number. Final fault location is then selected with the help of a $N:2^N$ decoder.

**RESULTS FOR ARITHMETIC PIPELINE :** For the arithmetic pipeline, from static timing analysis reports, we estimated the value of $T_{MAX}$ to be 9ns. For aggressively clocking the design, we increased the contamination delay to 3ns. Area of the circuit is increased by 38%, for fixing the contamination delay to 3ns. Pulses of varying widths ranging from 500 ps to 900 ps are injected in the unit under test (UUT). Each cycle, results are checked for correctness after the computation is over to ensure that the recovery mechanism works. Whenever recovery is triggered, we logged the occurrence of an error.

For evaluating STEM technique, we performed our experiments for a set error rate target of 1% over 10000 cycles. 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 the set target error rate. We considered a linear control scheme for switching clock frequency between the worst-case clock frequency, $F_{MIN}$ and the overclocked frequency, $F_{MAX}$. 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. Our fault injection results for the arithmetic pipeline are presented in Table 4. We initialized the LFSR with different seeds, and the fault injection results are presented for three different runs.

<table>
<thead>
<tr>
<th>RUN1</th>
<th>RUN2</th>
<th>RUN3</th>
</tr>
</thead>
<tbody>
<tr>
<td>TE</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>TE</td>
<td>14</td>
<td>12</td>
</tr>
<tr>
<td>TE</td>
<td>432</td>
<td>450</td>
</tr>
<tr>
<td>TE</td>
<td>2033</td>
<td>2033</td>
</tr>
<tr>
<td>TE</td>
<td>421</td>
<td>414</td>
</tr>
<tr>
<td>TE</td>
<td>2025</td>
<td>2025</td>
</tr>
<tr>
<td>TE</td>
<td>315</td>
<td>315</td>
</tr>
<tr>
<td>TE</td>
<td>2026</td>
<td>2026</td>
</tr>
<tr>
<td>TE</td>
<td>325</td>
<td>325</td>
</tr>
<tr>
<td>TE</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>TE</td>
<td>266</td>
<td>268</td>
</tr>
</tbody>
</table>

Table 4: Fault Injection Results for Arithmetic Pipeline

Figure 8: Normalized Arithmetic Pipeline Execution time

We configured the arithmetic pipeline designed with STEM cells to operate in three different modes. They are no overclocking (NOOC), wherein $T_{MAX} = T_{MIN} = 9\text{ns}$, maximum overclocking (MAXOC), wherein $T_{MAX} = T_{MIN} = 7\text{ns}$, and dynamic overclocking (DYNOC), wherein $T_{MAX} = 9\text{ns}$ and $T_{MIN} = 7\text{ns}$. For DYNOC mode, we started with a low frequency setting. For TMR system, worst-case frequency, $T_{MAX}$, is set at $11\text{ns}$. We evaluate SEM scheme at a constant clock period of $9\text{ns}$. Performance improvements offered by both SEM scheme and different modes of STEM are shown in Figure 8. From this, we can see that DYNOC mode offers 49% improvement over TMR, while MAXOC mode offers 55% improvement. Performance of NOOC mode is comparable to that of SEM and SEM offers 23% performance improvement over TMR. From Table 4, 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 a longer clock period compared to SEM and STEM, resulting in more SET pulses attenuating before reaching the latching window.

Results for DLX Processor: We also simulated three different micro benchmarks to evaluate the performance improvement and fault coverage of both SEM and STEM (DYNOC mode) schemes 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 is checked to verify the correctness of the computed results after simulation. From timing reports, the worst-case clock period, $T_{MAX}$, is estimated as $6\text{ns}$. Contamination delay is increased by $2\text{ns}$ and the system operates at an optimal clock period of $4\text{ns}$. Area overhead incurred is less than 15% 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 9, 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.

Figure 9: DLX Execution time for various benchmarks

7. Related Work

In the past, many hardware fault tolerance architectures have been developed by the research community. These schemes incur performance overhead even during error free operation and do not support aggressive clocking. LEON-FT processor [8] uses TMR approach and triplicates every flip flop in the processor and incurs a 100% area overhead. Redundant multi-threading based schemes exploit instruction level parallelism to provide fault tolerance [19]. These approaches trade performance and power for achieving fault tolerance capabilities. Systems designed with SEM cells improve reliability and does not incur any performance loss during normal operation.

Brute-force overclocking does not guarantee reliable execution. TEATIME [18] adjusts the system frequency dy-
namically, based on process and environmental variations, by employing timing error avoidance techniques. System performance can be enhanced further, by allowing a system to operate at a frequency that allows timing errors to happen. At such overclocked frequencies, relevant timing error detection and correction schemes can be used to guarantee functional correctness and to avoid any abnormal execution in the system. Prior work, Razor [7] and SPRIT\textsuperscript{E} [16], employ timing error tolerance techniques to operate beyond worst-case limits. While Razor focuses on achieving lower energy consumption by reducing supply voltage in each pipeline stage, SPRIT\textsuperscript{E} improves performance of a superscalar processor by reliably overclocking the pipeline. Other closely related works are Paceline [9] and CPipe [17]. Paceline employs leader-checker configuration in a chip multiprocessor system and tolerates both timing and soft errors. CPipe architecture enables reliable overclocking and enhances system reliability through core replication and conjoining them. Systems designed with STEM cells improve the reliability and performance of the system without logic duplication. Table 5 summarizes how our schemes differs from previously proposed techniques.

Table 5: Comparing with other schemes in terms of Logic Duplication (LD), Soft Error Protection (SEP), Aggressive Clocking (AC) and Energy Savings (ES)

<table>
<thead>
<tr>
<th>DESIGN</th>
<th>LD</th>
<th>SEP</th>
<th>AC</th>
<th>ES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Razor</td>
<td>√</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>SPRIT\textsuperscript{E}</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>Paceline</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>CPipe</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>SEM</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>STEM</td>
<td>×</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
</tbody>
</table>

8. Conclusions

In this work, we developed two efficient soft error mitigation schemes that remove the error detection overhead from the circuit critical path. One of our schemes, allow overclocking and is capable of tolerating timing errors as well. These specialized register cells provide near 100% fault tolerance against transient faults. Our schemes tolerate fast transient noise pulses, which is the principal characteristic of SETs. Both our schemes have no significant performance overhead during error-free operation. SEM cells are capable of ignoring false positives. One of the salient features of our approach lies in the capability to trigger recovery immediately on error detection, without requiring any checkpointing. Another key feature is that our scheme generates clocks locally with constant phase shift values, increasing the possible frequency settings for aggressively clocked designs. Also, our local clock generation and distribution minimizes the clock routing overhead incurred. 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.

References


