Conjoined Pipeline: Enhancing Hardware Reliability and Performance through Organized Pipeline Redundancy*

Viswanathan Subramanian and Arun K. Somani
Dependable Computing and Networking Laboratory
Iowa State University, Ames, IA, USA
{visu,arun}@iastate.edu

Abstract

Reliability has become a serious concern as systems embrace nanometer technologies. In this paper, we propose a novel approach for organizing redundancy that provides high degree of fault tolerance and enhances performance. We replicate both the pipeline registers and the pipeline stage combinational logic. The replicated logic receives its inputs from the primary pipeline registers while writing its output to the replicated pipeline registers. The organization of redundancy in the proposed Conjoined Pipeline system supports overclocking, provides concurrent error detection and recovery capability for soft errors, intermittent faults and timing errors, and flags permanent silicon defects. The fast recovery process requires no checkpointing and takes three cycles. Back annotated post-layout gate level timing simulations, using 45nm technology, of a conjoined two stage arithmetic pipeline and a conjoined five stage DLX pipeline processor, with forwarding logic, show that our approach achieves near 100% fault coverage, under a severe fault injection campaign, while enhancing performance, on an average by about 20%, when dynamically overclocked and 35%, when maximally overclocked.

1. Introduction

Technology scaling and hazardous operating environments make embedded processors and system-on-chips highly susceptible to faults. The impact of soft errors and silicon failures on system reliability have been steadily rising as we progress toward 32nm technologies and beyond. Soft errors, induced by high energy radiation and external noise, may result in incorrect computation and silent data corruption. Intermittent faults that persist for a short duration of time and silicon defects resulting from silicon failure mechanisms such as transistor wearouts, gate breakdown, hot carrier degradation, and manufacturing limitations degrade life-time and reliability of fabricated devices [4]. Transient, intermittent and permanent fault classes constitute the three major reasons for hardware failure. Reliability issues in combinational logic, control logic and random logic continues to be a major issue in the quest for highly dependable systems [7]. A major reason for this is the increasing probability of longer single event transient (SET) pulses in newer technologies [11].

Commercial systems opt for low overhead approaches that provide limited fault coverage and tolerate a subset of hardware fault classes, while incurring a modest performance penalty. On the other hand, servers designed for continued operation, such as IBM zSeries, have robust reliability, availability and serviceability features [10]. With the advent of Chip Multiprocessors (CMP), fault tolerance techniques that also improve performance have been developed [13]. These approaches utilize two cores to run an application with the goal of executing the application faster than on a single core, while leveraging the redundancy to tolerate faults.

Our earlier work SPRIT3E [15] employed temporal redundancy to reliably overclock a superscalar processor. By means of duplicating critical registers and clocking the redundant register by a delayed version of the system clock, SPRIT3E demonstrated that considerable performance improvement can be achieved through reliable overclocking. However, in the presence of faults, the redundant register cannot be relied upon fully, and this necessitates spatial redundancy of combinational logic to ensure that the value stored in the redundant register is “gold”. Our proposed approach is designed to tolerate transient and intermittent faults along with timing errors, and implements a robust error detection and recovery mechanism.

In this paper, we propose a conjoined duplex system approach to provide tolerance for myriad hardware faults that plague modern computing systems. The proposed approach is capable of protecting both the datapath and control logic. With minor additions to the error recovery procedure, the proposed system is capable of recovering from timing errors, thereby allowing a significant degree of overclocking. When coupled with a dynamic clock tuning mechanism based on a set target error rate, the system frequency adapts...
to application characteristics during run time. The concept of increasing the frequency and phase shifting the clocks makes sure that both the primary and redundant pipelines can run faster and the second pipeline is timing safe.

The rest of this paper is organized as follows: The CPipe architecture, pipeline datapath and the error detection and recovery methodology is described in Section 2. In Section 3, the relevant parameters that affect dynamic frequency scaling and the possible range of operating frequencies are derived. In Section 4, CPipe system implementation issues are discussed. Experiments and results are presented in Section 5. Section 6 provides a review of related literature. Section 7 concludes the paper.

2. Conjoined Pipeline Architecture

The proposed “Conjoined Pipeline” (in short, CPipe) employs a special way to organize pipeline redundancy, with the goal of tolerating the three major fault classes that severely undermine the reliability of current and future computing systems. The CPipe microarchitectural technique builds on the better-than-worst-case design methodologies [1] proposed in Razor [6] and SPRIT3E [15] performance enhancement techniques.

The basic principle behind the CPipe system is to replicate the entire pipeline, and interlink the two pipelines in a way so as to provide capability to tolerate various fault types. The term “conjoined” implies the intertwining of the two pipelines and their constant and continued dependency on each other. Both primary and redundant pipelines are susceptible to faults that are uniformly distributed in time and space. Timing errors occur when the primary pipeline is overclocked to speed up execution. The redundant pipeline is guaranteed to have sufficient time for execution, and is free from timing errors. The ensuing description of the CPipe architecture explains how random occurrence of faults and timing errors are handled concurrently. The following description assumes that the CPipe system is running at an overclocked frequency, when errors are detected.

2.1. CPipe Datapath Description

The organization of redundancy in CPipe is illustrated in Figure 1. The figure shows three pipeline stages: P-STAGE N-1, P-STAGE N and P-STAGE N+1. The CPipe concept in its entirety is portrayed in the figure for P-STAGE N. The primary pipeline is referred to as the L-Pipeline (Leading Pipeline) and the redundant pipeline as the S-Pipeline (Shadow Pipeline). In the figure, the shaded pattern distinguishes the L-Pipeline from the S-Pipeline. The L-Pipeline registers, S-Pipeline registers, E-Detect module, and the MUX before the L-Pipeline registers together form the local fault detection and recovery (LFDR) circuit. The LFDR circuit, highlighted in the figure, replaces the pipeline registers that are present in a normal pipelined system. In the figure, feedback signals indicate signals received from other pipeline stages other than the immediately preceding stage.

To provide tolerance for soft errors that occur in the combinational logic, the pipeline stage combinational logic between the pipeline registers are duplicated. The leading logic, L-LOGIC, receives its inputs from the previous stage L-Pipeline register, and stores its computed results in the current stage L-Pipeline register. However, the shadow logic, S-LOGIC, though receiving its inputs from the previous stage L-Pipeline register, stores its outputs in the current stage S-Pipeline register. To lucidly understand the CPipe concept, observe in Figure 1 that the L-Pipeline register of P-STAGE N-1 feed both the L-LOGIC and S-LOGIC of P-STAGE N, and the L-LOGIC of P-STAGE N writes its results to the L-Pipeline register of P-STAGE N, while the S-LOGIC of P-STAGE N writes its output to the S-Pipeline register of P-STAGE N. The above implementation ensures that both datapath and control signals are protected from hardware faults. P-STAGE N+1 speculatively uses the L-Pipeline register values. If an error is detected, then the computed results are discarded.

The CPipe architecture conceptually requires three clocks for proper operation. The error clock, E_CLK, and the shadow clock, S_CLK, are phase shifted versions of the
leader clock, $L_{CCLK}$. As is seen, two of the clocks are derived from the main clock, and as will be explained in Section 4, if desired only one clock needs to be routed inside the design. These three clocks, along with the error signals from all the pipeline stages, control $L_{CCLK}$ and $S_{CCLK}$. The L-Pipeline registers are clocked by $L_{CCLK}$, while the S-Pipeline registers are clocked by $S_{CCLK}$. $E_{CCLK}$ is required to precisely control when $L_{CCLK}$ and $S_{CCLK}$ needs to be stalled to ensure correct operation. Also on error detection, the control signal to load the S-Pipeline register values into the L-Pipeline register, shown as $Load_{SP}$ in Figure 1, is asserted for a cycle. This entire control mechanism is performed in the clock stall control module, shown as CLK STALL CNTRL in Figure 1.

2.2. Error Detection and Recovery

The most important thing to note is that even though the results computed by the S-LOGIC are free from timing errors, they are susceptible to soft errors. Considering this, in the CPipe architecture error detection is performed before storing the results in the S-Pipeline registers. Only if the results computed by the S-LOGIC matches the values stored in the L-Pipeline registers, then the S-LOGIC outputs are written into the S-Pipeline registers. The E-DETECT module incorporates metastability detection, as in [5], for the L-Pipeline registers, as the L-Pipeline flip-flops may enter a metastable state when overclocked, or when a soft error reaches the registers during the latching window. The Error flag is asserted to indicate an error.

The delay between the clocking of the L-Pipeline and the S-Pipeline registers of a pipeline stage introduce the necessary spatial and temporal redundancy required to detect timing errors. The contamination delay, which 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, of the S-LOGIC needs to be increased to a value more than the delay between the $L_{CCLK}$ and the $S_{CCLK}$. This is important to ensure that the S-LOGIC outputs are not changed by the values newly registered in the L-Pipeline register.

The error detection and recovery process does not differentiate between errors occurring in the S-Pipeline and the L-Pipeline. The L-Pipeline is susceptible to both soft errors and timing errors, while the S-Pipeline is susceptible to soft errors. The transient fault tolerance mechanism is overloaded to detect and recover from any timing errors that might occur because of overclocking.

Figure 2 illustrates the entire error detection and recovery mechanism when a soft error or a timing error occurs in the L-Pipeline of P-STAGE N (see Figure 1). The figure shows the instructions that are being executed in the L-LOGIC and the S-LOGIC of the three pipeline stages. If the instructions in the L-Pipeline execute without errors, the conjoined pipeline proceeds without any interruption.

An error occurrence is highlighted in cycle 3. The error occurs during the execution of INST 1 in the L-LOGIC of P-STAGE N. This error is not yet detected, leading to the output of L-LOGIC being stored in the registers of P-STAGE N. The L-Pipeline of P-STAGE N starts executing INST 2 in cycle 4. However, the L-Pipeline of P-STAGE N+1 executes INST 1 in cycle 4 with incorrect result provided by P-STAGE N. This needs to be corrected. After the S-LOGIC of P-STAGE N finishes execution, the E-DETECT module detects the mismatch between the L-Pipeline register values and the outputs of S-LOGIC. The error flag is asserted, triggering the recovery process.

The Error signals from all the pipeline stages are combined together using “or” gates, and latched by the rising edge of $E_{CCLK}$ in the CLK STALL CNTRL module. The latched signal is referred to as $G_{Error}$ in Figure 2. The global error signal ensures correct execution of the pipeline, and helps in global recovery. The $G_{Error}$ signal is asserted or deasserted only when the recovery counter ($RC_{Counter}$) is $B"00"$ or $B"11"$. This guarantees two cycles for the L-Pipeline to re-execute the erroneous instruction. This is necessary since the error might have been caused because of overclocking. It can be observed in Figure 2 that on error detection the entire pipeline goes back by one instruction. It is also possible to insert bubbles to avoid re-execution of instructions in the forward pipeline stages.

The entire error detection and recovery mechanism happens in three cycles. The cycle counts are with respect to the leader clock. The following sequence takes place in cycles
4, 5 and 6 as a result of the error in cycle 3:

- **FIRST** (See cycle 4 in Figure 2): Error flag is asserted by the E-DETECT module, and $G_{\text{Error}}$ goes high at the rising edge of $E_{\text{Clk}}$. Immediately after $G_{\text{Error}}$ goes high, $S_{\text{ClkStall}}$ goes low before the $S_{\text{Clk}}$ edge, and $Load_{SP}$ goes high before the $L_{\text{Clk}}$ edge. As a result, $S$-PIPELINE registers are not updated, as $S_{\text{GC}}$ is low, and value from $S$-PIPELINE registers are loaded into the corresponding $L$-PIPELINE registers. $R_{\text{Counter}}$ is incremented at the end of the cycle.

- **SECOND**: $L_{\text{ClkStall}}$ goes low at the negative edge of $L_{\text{Clk}}$. This stalls $L_{\text{GC}}$ in the next cycle and avoids any glitches. $S_{\text{ClkStall}}$ remains low. Both $L_{\text{ClkStall}}$ and $S_{\text{ClkStall}}$ are active low signals. In this cycle, both $L$-PIPELINE and $S$-PIPELINE registers are not updated. $R_{\text{Counter}}$ is incremented at the end of the cycle.

- **THIRD**: $L_{\text{ClkStall}}$ signal goes high at the negative edge of $L_{\text{Clk}}$. $S_{\text{ClkStall}}$ remains low and $R_{\text{Counter}}$ is incremented.

At the end of cycle 6, the erroneous instruction completes its re-execution successfully. In cycle 7, the execution of CPipe returns to normal. In Figure 2, it can be seen that during the recovery process the $S$-LOGIC computes intermediate results, but the outputs are not written to the $S$-PIPELINE registers.

**Possible Error Scenarios:** The possible error scenarios include soft error or timing error in $L$-PIPELINE, combined with a soft error in the $S$-PIPELINE. Intermittent faults are also possible in either of the pipelines. The error detection and recovery mechanism described is robust and can handle any number of errors in a single cycle.

**Intermittent Faults:** If in cycle 7, the global error signal does not go low, the entire recovery process is repeated. The recovery process is triggered repeatedly until the error disappears. This allows recovery from transients that persist for a short duration of time. Intermittent faults that occur in bursts are handled similarly by the CPipe architecture.

**Permanent Faults:** If after significant number of retries the error persists and the pipeline is stuck in a loop, then the fault is declared as permanent. The permanent fault flag is asserted indicating a system failure. In this case, it could be possible to reconfigure the CPipe system to run either using only the $L$-PIPELINE or the $S$-PIPELINE with no fault tolerance and overclocking. In this paper, this possibility is not pursued further. Also, it can be noted here that with additional logic, we can choose a combination of $L$-PIPELINE and $S$-PIPELINE stages, if the need arises during reconfiguration.

**Timing Errors:** Timing errors occur when the system is overclocked. However, when the CPipe is used primarily for fault tolerance, and not for improving performance, the signal Overclock is deasserted, indicating that there will be no timing errors in the $L$-PIPELINE. In this case, error recovery takes two cycles, as $L_{\text{Clk}}$ is not stalled to accommodate recovery from timing errors.

**Fault Tolerance Analysis:** The possibility of the CPipe architecture not detecting a fault is extremely low. One possibility is a timing error happening in the $L$-PIPELINE and a soft error happening in the $S$-PIPELINE, and the error flag not being asserted because of identical corruption. This possibility is extremely low since even if a single mismatch happens in the entire system, the error flag is asserted. Timing errors and soft errors affect multiple bits. Another case is when both the $S$-LOGIC and $L$-LOGIC are affected by soft errors. The same soft error cannot affect both the logic, if so, then this will be detected by the previous stage E-DETECT module. This is because the $L$-PIPELINE register outputs feeding the $S$-LOGIC also goes to the E-DETECT module. Another failure possibility is when a transient pulse occurs right after the error signal is latched and before the $S$-LOGIC outputs are stored in the $S$-PIPELINE registers. This duration is extremely small (one NOT and one AND gate, plus global routing delay), and given the distribution of soft errors in time and space, this error possibility is insignificant. The error register is metastability hardened, and any small variation will make the $G_{\text{Error}}$ signal go high. In essence, the CPipe architecture is capable of providing very high degrees of fault coverage.

### 3. Timing Requirements

![Figure 3. Dynamic Frequency Scaling](image)

For proper operation of systems implementing the CPipe architecture, it is of paramount importance to respect the timing relationship between the three clocks, namely, $L_{\text{Clk}}$, $E_{\text{Clk}}$ and $S_{\text{Clk}}$. To support reliable dynamic overclocking, certain governing conditions needs to be met at all times. Figure 3 shows the parameters that control the full range of frequencies, $F_{\text{Min}} = F_{\text{Max}}$, that are possible when a
The system is dynamically overclocked beyond the worst-case operating frequency, $F_{\text{Min}}$. For our CPipe architecture, we extend the clock generation methodology described in SPRIT$^\text{E}$ [15]. In a CPipe system, there are three clocks and two of those, $E_{\text{Clik}}$ and $S_{\text{Clik}}$, are phase shifted versions of the $L_{\text{Clik}}$. Additionally, CPipe requires minimum phase shift guarantees for correct operation.

The following parameters that can be estimated for $F_{\text{Min}}$ settings of any digital system are defined below to calculate the dynamic frequency operation range:

- Let $T_{\text{Max}}$ represent the worst-case time period required by the digital system under consideration.
- Let $T_{\text{Err}}$ represent the time required for error detection and assertion of the global error signal. This includes the E-Detect module delay and generation of the global error signal from the pipeline stage error signals.
- Let $T_{\text{SS Stall}}$ represent the time required to stall $S_{\text{Clik}}$ to prevent incorrect value from being loaded into the S PIPELINE registers. This includes the clock gating delay and the clock propagation delay.
- Let $T_{\text{LSP}}$ represent the time required to assert $L_{\text{SP}}$ signal on detection of an error, the routing delay, and the multiplexer delay required to load the S PIPELINE register values into the L PIPELINE registers.
- Let $T_{\text{FmaxCD}}$ represent the minimum contamination delay of the S-LOGIC of all the pipeline stages.

Figure 3 shows the time available for the above operations under $F_{\text{Min}}$ and $F_{\text{Max}}$ settings. $PS_{\text{Min}}$, defined by Equation 1, represents the minimum required phase shift to ensure correct operation, and it should satisfy Equation 2.

$$PS_{\text{Min}} = T_{\text{Err}} + T_{\text{SS Stall}} \quad (1)$$

$$PS_{\text{Min}} \leq T_{\text{FmaxCD}} \quad (2)$$

**Clock Timing Requirements:** $L_{\text{Clik}}$ active edge occurs first, followed by $E_{\text{Clik}}$ edge, and then $S_{\text{Clik}}$ edge. $E_{\text{Clik}}$ time lag should at least be equal to $T_{\text{Err}}$, and the $S_{\text{Clik}}$ phase shift amount should at least be $PS_{\text{Min}}$. Fixing the phase shift between $E_{\text{Clik}}$ and $S_{\text{Clik}}$ as $T_{\text{SS Stall}}$ makes dynamic frequency operation easier, since only the phase shift between $L_{\text{Clik}}$ and $E_{\text{Clik}}$ needs to be controlled. Also, maintaining the phase shift value between $E_{\text{Clik}}$ and $S_{\text{Clik}}$ to the bare minimum reduces the possibility of common mode failure. The rest of the discussion in this paper is based on this approach. The effects that lead to variable circuit delays, such as temperature, voltage, and process variations, also cause variations in the clock period, referred to as clock skew. In order to account for this possibility, the worst case clock skew is assumed when determining the maximum frequency scaling achievable, and is added to the estimation of $T_{\text{Err}}, T_{\text{LSP}}$ and $T_{\text{SS Stall}}$.

**Dynamic Overclocking:** When dynamic overclocking is done to improve performance, the following additional parameters needs to be derived for $F_{\text{Max}}$ settings:

- Let $T_{\text{Min}}$ represent the minimum clock period at which the system is guaranteed to recover from timing errors that might happen as a result of overclocking.
- Let $PS_{\text{Max}}$ represent the maximum phase shift required to ensure correct operation.
- Let $T_{\text{FmaxCD}}$ represent the minimum contamination delay of the S-LOGIC of all the pipeline stages.

As seen in Figure 3, the only parameter that becomes critical because of frequency scaling is $T_{\text{LSP}}$. $T_{\text{Err}}$ and $T_{\text{SS Stall}}$ are taken care by the clock timing requirements, and their criticalness continue to remain the same as in $F_{\text{Min}}$ settings. Depending on the extent of overclocking required, the value $T_{\text{FmaxCD}}$ is fixed at any value within the range, $PS_{\text{Min}} \leq T_{\text{FmaxCD}} \leq T_{\text{Min}}$. If a higher value is chosen, then the contamination delay of the S-LOGIC of the pipeline stages needs to be increased above this value.

The error detection and if necessary, the recovery, should be initiated before the L PIPELINE registers receive the next set of values. The minimum clock period, $T_{\text{Min}}$, and the corresponding phase shift, $PS_{\text{Max}}$, should satisfy the following.

$$T_{\text{Min}} \leq \frac{T_{\text{Max}} + T_{\text{Err}} + T_{\text{LSP}}}{2} \quad (3)$$

$$PS_{\text{Max}} = T_{\text{Max}} - T_{\text{Min}} + PS_{\text{Min}} \quad (4)$$

$$PS_{\text{Max}} \leq T_{\text{FmaxCD}} \quad (5)$$

Let $T_{\text{PS}}$ represent the adjustable phase shift value. Equation 6 and 7 define the range of phase shift values and the range for $T_{\text{PS}}$.

$$PS_{\text{Min}} \leq T_{\text{PS}} + T_{\text{Err}} + T_{\text{SS Stall}} \leq PS_{\text{Max}} \quad (6)$$

$$0 \leq T_{\text{PS}} \leq T_{\text{Max}} - T_{\text{Min}} \quad (7)$$

For a system under consideration, the values of $T_{\text{Min}}$ and $PS_{\text{Max}}$ are derived using the above method. Then, for any frequency $F$, such that $F_{\text{Min}} \leq F \leq F_{\text{Max}}$, the associated time period $T$ equals $T_{\text{Max}} - T_{\text{PS}}$ and the phase shift $PS$ equals $PS_{\text{Min}} + T_{\text{PS}}$.

**Fixed Frequency Operation:** For operating without any run-time optimizations, the frequency of the three clocks is fixed at the desirable operating frequency satisfying the above conditions, and the required phase shifts between the clocks are enforced. Under these conditions, the CPipe architecture offers protection from soft errors and permanent fault detection, while achieving performance improvement if the error rate is low. Also, it is important to ensure that the contamination delay of the S-LOGIC of all the pipeline stages is more than the phase shift required for this frequency. If the frequency is fixed at the worst-case operating frequency, the CPipe system is guaranteed to have the same performance as an unprotected system, while offering high reliability. If operating at or below worst-case operating frequency, the Overclock signal is deasserted enabling two cycle recovery from transient errors.
4. Implementation Considerations

![Figure 4. Modular Implementation](image)

The CPipe architecture is easy to integrate in any system during the RTL/structural level design phase. After the modules representing pipeline combinational logic are designed, they can be assembled together by using the local fault detection and recovery (LFDR) circuits instead of the registers. As explained earlier, the LFDR module includes error detection logic and both leader and shadow pipeline registers. The LFDR circuit is designed as a separate module, with its database width configurable. Modular design makes it easy to replicate the logic. The connectivity is done, as explained in the previous Sections, and the CPipe system implementation is complete. Figure 4(a) illustrates the modular implementation of CPipe architecture, where L and S stand for leader logic and shadow logic, respectively. This can be extended to any number of pipeline stages.

One of the major issues that needs to be taken care of is the clocking of the LFDR circuit, and ensuring that the timing requirements derived in Section 3 are met. For pipeline stages with less latencies, the error detection delay, $T_{Err}$ will be significant. Also because of the global routing delays the performance gain will be modest, as frequency cannot be scaled much. However, in most pipelined designs, the longest pipeline stage limits the frequency. If the critical path in the slowest pipeline stage is not exercised often, then it can be overclocked, and all other pipeline stages will benefit. The CPipe approach guarantees high degree of fault coverage for all designs, while offering performance gains whenever possible. One significant benefit that is derived from implementing CPipe architecture is the reduction in design optimization time to achieve particular performance. The adaptive clocking mechanism allows performance to match or exceed expected levels during run-time.

Two Clock Approach: Clock distribution and routing inside a design is one of the major design issues. At high frequencies, clock skew will limit the implementation of CPipe architecture, as it requires strict timing requirements between the three clocks at all times to guarantee correct execution. With a modest increase in implementation overhead, the CPipe architecture is allowed to operate with two clocks. The leader clock is inverted inside the LFDR circuit to locally generate the shadow clock. Also the clock stall logic is moved inside the LFDR circuit. With this setup, only one clock needs to be routed inside the design, since the error clock, $E_{Cllk}$, is used only to clock the error register.

The duty cycle of $L_{Cllk}$ is adjusted to maximize performance gain. Since 50% duty cycle is not necessary for the system to operate correctly, it can be adjusted accordingly, instead of varying the phase shift. For this approach to work, few conditions need to be taken care of. Let $T_{High}$ represent the high time, and let $T_{Low}$ represent the low time of $L_{Cllk}$. The assertion and global routing delay of the two stall signals, $L_{Cllk}Stall$ and $S_{Cllk}Stall$, and the load S-Pipeline signal, $Load_{SP}$ determine $T_{Low}$. Since these delays do not change with frequency scaling, sufficient time should be guaranteed at $F_{Min}$ settings. Also, the phase shift of $E_{Cllk}$ is kept below $T_{High} - T_{SSStall}$.

5. Experiments and Results

To prove the viability of the CPipe system architecture, we performed the following experimental runs on a two stage arithmetic pipeline. Our designed CPipe system performs 64-bit addition in the first stage, and a 32-bit multiplication in the second stage. The 64-bit carry look ahead adder output is divided into two, and fed to the multiplier as multiplicand and multiplier.

We synthesized our design in Synopsys design compiler. We used the 45nm OSU standard cell library [14]. From static timing analysis reports, we estimated the values of $T_{Max}$ as 9.1ns, $T_{Err}$ as 1.7ns, $T_{SSStall}$ as 0.67ns, and $T_{SP}$ as 1.85ns. Then, using the equations derived in Section 3 we calculated the values of $T_{Min}$ as 6.33ns and $P_{Max}$ as 5.14ns. Based on these values, the synthesis was performed again with minimum delay constraints to increase the contamination delay of the S-Logic blocks. Since increasing contamination delay increases area and power, we chose not to overclock all the way to $T_{Min}$, and our implementation supported overclocking up to 7ns. We used SOC encounter tool to layout the design and to extract standard delay format (SDF) timing information. We did timing simulations on the SDF annotated post layout design to evaluate fault coverage and performance improvement. We designed in Vhdl the dynamic clock generation and switching circuit and used a 1000 cycle delay for locking the new frequency.

In our experiment, for 1ns run, we injected faults randomly in time and space and evaluated the fault tolerance capability of the design for the various fault types. Our random fault injectors introduced approximately 100 transient faults and 3 intermittent faults in 100,000 cycles. Some of the intermittent faults persisted longer simulating a permanent fault. The pipeline output is verified for correctness by comparing with a non fault injection run. Timing errors occur as a result of overclocking, and the timing error recovery process was also verified similarly.

We repeated the experiment with three different random seeds, and performed the experiments in three different modes. The three modes were, no overclocking (NOOC), $T_{Min} = 9.1ns$, $T_{Max} = 9.1ns$, maximum overclocking
\( T_{\text{Min}} = 7\text{ns}, T_{\text{Max}} = 7\text{ns} \), and dynamic overclocking (DYNOC), \( T_{\text{Min}} = 7\text{ns}, T_{\text{Max}} = 9.1\text{ns} \). Table 1 reports results for the three types of faults injected, and it also presents the number of correct operations performed in \( 1\text{ns} \) time duration. In [15], for a multiplier circuit, 44\% performance improvement was achieved for an error rate target of 1\%. However, because of the limitations imposed on the clock timing requirements, the maximum frequency that is achievable in \( CP\text{ipe} \) is limited. Even while running at maximum possible frequency, for randomly generated inputs we observed extremely less timing errors. From the results, we can see that when running at worst-case frequency less transient errors get detected, as most of them are masked because of the longer clock period. In dynamic overclocking mode, we perform binary search on the allowed range of frequencies and also account for the clock scaling penalty. Always running at the maximum frequency yields the best results for the two-stage arithmetic pipeline. Even when exposed to a severe fault campaign, we obtain approximately 28\% performance increase over NOOC while operating in MAXOC, DYNOC offers about 21\% performance increase over NOOC.

![Relative performance gains for different applications](image)

**Figure 5. Execution time for three different applications running in three different modes on the Conjoined Processor**

We also designed and simulated a five stage conjoined in-order pipeline processor. The conjoined processor implemented in \( 45nm \) technology supports operand forwarding and is based on the DLX instruction set architecture. The purpose of this experiment was to prove that \( CP\text{ipe} \) architecture works perfectly well in the presence of feedback signals. We ran three different microbenchmarks to evaluate the conjoined processor architecture. The microbenchmarks were written in assembly. The RandGen application performs a simple random number generation to give a number between 0 and 255. One million random numbers are generated, and the distribution of the random variable is kept in memory. The MatrixMult application multiplies two \( 50x50 \) integer matrices and stores the result into memory. The BubbleSort program performs a bubblesort on 5,000 half-word variables. The performance of the three modes is shown in Figure 5. The fault injection campaign is similar to the arithmetic pipeline case. From static timing analysis reports, we estimated the values of \( T_{\text{Max}} \) as 6\( \text{ns} \), \( T_{\text{Err}} \) as 1.4\( \text{ns} \), \( T_{\text{Stall}} \) as 0.67\( \text{ns} \), and \( T_{\text{LSP}} \) as 1.7\( \text{ns} \). We estimated \( T_{\text{Max}} \) to be 4.55\( \text{ns} \). From the figure, we see that all three application show significant performance gains while operating in DYNOC and MAXOC modes, even when subjected to a severe fault campaign.

For our approach, there are no timing overheads on the leading pipeline except for the MUXing-delay in our approach. The error detection is done in parallel with useful computation. Superficially, area overhead is the cost of a second core along with overclocking and error detection overhead. For the two stage conjoined arithmetic pipeline, the post-layout area is estimated to be \( 1.72E5 \text{um}^2 \), which works out to 285\% the size of the a non fault tolerant arithmetic pipeline. The DLX processor area is about 310\% the size of the original processor. A significant component of the area overhead results from the contamination delay compensation of the S-LOGIC. By designing buffers specifically for this purpose and a robust algorithm to increase short path delays, this overhead can be alleviated.

### 6. Related work

A whole gamut of hardware fault tolerance approaches that provide partial to full protection against the entire or a subset of fault classes have been developed in the past by the research community. To achieve high degree of fault coverage while minimizing the redundancy overhead, SSD [9] consists of an integrity checking architecture for superscalar processors that can achieve fault tolerance capability of a duplex system at much less cost than the traditional duplication approach. The REESE architecture [12] takes advantage of spare elements in a superscalar processor to perform redundant execution. Another popular strategy is to use multiple cores to improve performance and/or improve fault tolerance [13, 17].

While brute-force overclocking improves performance [3], it does not guarantee computational correctness. Hence, researchers have developed solutions that reliably and dynamically adjust frequency to the optimal value. The worst-case delay paths are sensitized only for specific input combinations and sequences [1]. Timing error avoidance techniques that overclock, but within safe limits also have been proposed. TEATIME [16], scales the frequency of a pipeline using dynamic timing error avoidance techniques. Adaptive overclocking automatically adjusts frequency for variations in process and environmental conditions during run-time.

Many dependable computer architectures apply fault-tolerant techniques that guarantee reliable execution with the goal of improving performance past worst case limits. Our work is closely related to Razor [5, 6] and SPRITE [15] techniques. In comparison to other works such as DIVA [2] and Puceline [8], in our work a significant advantage comes from conjoining the processors that allows...
for a very fast recovery and does not require checkpointing. Moreover, by enabling reliable overclocking, CPIpe can run faster and is not limited by execution rate of the slower of the two processors.

7. Conclusions

Most often than not, system designers need to make a difficult tradeoff choice between reliability and high performance. In this paper, we propose a solution that guarantees fault tolerant execution without compromising on the performance of the system. The solution proposed integrates overclocking with redundant execution thereby providing tolerance to soft errors, timing errors, intermittent faults and permanent faults. The CPIpe architecture relies on the organization of redundancy and adaptive clocking capabilities to improve fault coverage and performance. One of the salient features of our approach lies in the capability to trigger recovery immediately on error detection, without requiring any checkpointing, thereby saving the time and space required to store the current execution status. The CPIpe architecture protects both the datapath and control signals. In essence, the CPIpe architecture presents a viable high performance high reliability solution. In the future, we will implement the CPIpe architecture in complex pipelined systems and evaluate the fault coverage and performance for more representative benchmarks.

References


Table 1. Fault Injection Results

<table>
<thead>
<tr>
<th>Mode</th>
<th>Run</th>
<th>Operations</th>
<th>Transient Faults</th>
<th>Intermittent Faults</th>
<th>Permanent Faults</th>
<th>Timing Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Injected</td>
<td>Detected</td>
<td>Injected</td>
<td>Detected</td>
</tr>
<tr>
<td>NOOC</td>
<td>1</td>
<td>109818</td>
<td>981</td>
<td>131</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>NOOC</td>
<td>2</td>
<td>109892</td>
<td>937</td>
<td>105</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>NOOC</td>
<td>3</td>
<td>109777</td>
<td>941</td>
<td>124</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>MAXOC</td>
<td>1</td>
<td>141054</td>
<td>913</td>
<td>216</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>MAXOC</td>
<td>2</td>
<td>140976</td>
<td>953</td>
<td>193</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>MAXOC</td>
<td>3</td>
<td>140879</td>
<td>919</td>
<td>214</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>DYNOC</td>
<td>1</td>
<td>132975</td>
<td>925</td>
<td>207</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>DYNOC</td>
<td>2</td>
<td>133053</td>
<td>914</td>
<td>177</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>DYNOC</td>
<td>3</td>
<td>133031</td>
<td>933</td>
<td>190</td>
<td>26</td>
<td>26</td>
</tr>
</tbody>
</table>