Timing speculation and adaptive reliable overclocking techniques for aggressive computer systems

Viswanathan Subramanian

Iowa State University

Follow this and additional works at: http://lib.dr.iastate.edu/etd

Part of the Electrical and Computer Engineering Commons

Recommended Citation
http://lib.dr.iastate.edu/etd/10967
Timing speculation and adaptive reliable overclocking techniques for aggressive computer systems

by

Viswanathan Subramanian

A dissertation submitted to the graduate faculty in partial fulfillment of the requirements for the degree of DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Arun K. Somani, Major Professor
   Akhilesh Tyagi
   Randall L. Geiger
   Joseph A. Zambreno
   David Fernández-Baca

Iowa State University
Ames, Iowa
2009

Copyright © Viswanathan Subramanian, 2009. All rights reserved.
To my dear parents

To my enlightening teachers

To my loving wife

To my caring family

To my beloved friends
TABLE OF CONTENTS

LIST OF TABLES ................................................................. vi
LIST OF FIGURES ............................................................. viii
ACKNOWLEDGEMENTS ......................................................... xii
ABSTRACT ................................................................. xiv

CHAPTER 1. INTRODUCTION ................................................. 1
  1.1 High Performance Computing ........................................ 3
    1.1.1 Device Scaling .................................................. 4
    1.1.2 Microprocessor Architectures .................................. 4
    1.1.3 Better-Than-Worst-Case Designs ............................... 5
    1.1.4 Adaptive Systems .............................................. 6
  1.2 Fault Tolerant Computing ........................................... 7
    1.2.1 Transient Faults ............................................... 8
    1.2.2 Redundancy Techniques ....................................... 8
    1.2.3 Fault Mitigation Techniques ................................. 9
    1.2.4 Exploiting Fault Tolerance to Improve Performance ............ 10
  1.3 Power/Thermal Aware Computing ................................... 11
  1.4 Contributions of this Thesis ..................................... 12

CHAPTER 2. BACKGROUND ..................................................... 16
  2.1 Parameter Variations ............................................... 16
  2.2 Reliable Overclocking .............................................. 18
    2.2.1 Timing Error Detection and Recovery ......................... 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.2 Timing Error Rate Based Feedback Control System</td>
<td>20</td>
</tr>
<tr>
<td>2.2.3 Timing Speculation</td>
<td>20</td>
</tr>
<tr>
<td>2.3 Razor Architecture</td>
<td>22</td>
</tr>
<tr>
<td>2.4 SPRIT$^3$E Framework</td>
<td>24</td>
</tr>
<tr>
<td>CHAPTER 3. MANIPULATING SHORT-PATHS FOR PERFORMANCE</td>
<td>27</td>
</tr>
<tr>
<td>3.1 Impact of Short-paths</td>
<td>28</td>
</tr>
<tr>
<td>3.1.1 Timing Constraints</td>
<td>28</td>
</tr>
<tr>
<td>3.1.2 Variable or Fixed Phase Shift</td>
<td>30</td>
</tr>
<tr>
<td>3.1.3 Manipulating Contamination Delay</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Increasing Contamination Delay of a CLA Adder Circuit - A Case Study</td>
<td>33</td>
</tr>
<tr>
<td>3.2.1 Analysis of Reliable Overclocking Performance</td>
<td>37</td>
</tr>
<tr>
<td>CHAPTER 4. CHARACTERIZING ADAPTIVE RELIABLE OVERCLOCKING</td>
<td>41</td>
</tr>
<tr>
<td>4.1 Evaluating Speculative Reliable Overclocking</td>
<td>42</td>
</tr>
<tr>
<td>4.1.1 Performance Metrics</td>
<td>46</td>
</tr>
<tr>
<td>4.2 Analysis Framework</td>
<td>47</td>
</tr>
<tr>
<td>4.2.1 Modeling a Reliably Overclocked Processor (ROP)</td>
<td>48</td>
</tr>
<tr>
<td>4.2.2 Power and Thermal Modeling</td>
<td>53</td>
</tr>
<tr>
<td>4.3 Adaptive Clocking</td>
<td>55</td>
</tr>
<tr>
<td>4.3.1 Clock Tuning Schemes</td>
<td>57</td>
</tr>
<tr>
<td>4.3.2 Comparing Adaptive Clocking Techniques</td>
<td>59</td>
</tr>
<tr>
<td>4.4 Reliable Overclocking Analysis</td>
<td>63</td>
</tr>
<tr>
<td>CHAPTER 5. THERMAL IMPACT OF RELIABLE OVERCLOCKING</td>
<td>69</td>
</tr>
<tr>
<td>5.1 Thermal and Reliability Management</td>
<td>70</td>
</tr>
<tr>
<td>5.2 Analysis Framework for Estimating On-chip Temperature</td>
<td>71</td>
</tr>
<tr>
<td>5.2.1 Thermal Throttling</td>
<td>72</td>
</tr>
<tr>
<td>5.2.2 Simulation Parameters</td>
<td>72</td>
</tr>
<tr>
<td>5.3 On-chip Temperature Trends in Reliably Overclocked Processors</td>
<td>73</td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>6</td>
<td>RELIABLE OVERCLOCKING AND TECHNOLOGY SCALING</td>
</tr>
<tr>
<td></td>
<td>6.1 Technology Scaling</td>
</tr>
<tr>
<td></td>
<td>6.2 A Reliable Overclocking Approach</td>
</tr>
<tr>
<td></td>
<td>6.3 Analysis Framework</td>
</tr>
<tr>
<td></td>
<td>6.4 Performance at Different Technology Nodes</td>
</tr>
<tr>
<td></td>
<td>6.5 Comparing Technology Scaling with Reliable Overclocking</td>
</tr>
<tr>
<td>7</td>
<td>FAULT TOLERANT AGGRESSIVE SYSTEMS</td>
</tr>
<tr>
<td></td>
<td>7.1 Conjoined Pipeline Architecture</td>
</tr>
<tr>
<td></td>
<td>7.1.1 Conjoined Pipeline Datapath Description</td>
</tr>
<tr>
<td></td>
<td>7.1.2 Error Detection and Recovery</td>
</tr>
<tr>
<td></td>
<td>7.2 Timing Requirements</td>
</tr>
<tr>
<td></td>
<td>7.3 Implementation Considerations</td>
</tr>
<tr>
<td></td>
<td>7.3.1 Two Clock Approach</td>
</tr>
<tr>
<td></td>
<td>7.4 Experiments and Results</td>
</tr>
<tr>
<td>8</td>
<td>CONCLUSIONS AND FUTURE WORK</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<p>| Table 3.1 | Implementation details of CLA adder circuits | 37 |
| Table 4.1 | Processor specifications | 49 |
| Table 4.2 | Synthesis report of major pipeline stages | 49 |
| Table 4.3 | Simulator parameters | 53 |
| Table 4.4 | Comparing various performance metrics between a base non-overclocked processor, a reliably overclocked processor tuned using a single clock generator and a reliably overclocked processor tuned using dual clock generators. All the systems execute SPEC2000 integer benchmarks | 61 |
| Table 4.5 | Comparing various performance metrics between a base non-overclocked processor, a reliably overclocked processor tuned using a single clock generator and a reliably overclocked processor tuned using dual clock generators. All the systems execute SPEC2000 floating point benchmarks | 62 |
| Table 4.6 | Comparing various performance metrics for non-overclocked and reliably overclocked processors executing SPEC2000 integer benchmarks | 67 |
| Table 4.7 | Comparing various performance metrics for non-overclocked and reliably overclocked processors executing SPEC2000 floating point benchmarks | 67 |
| Table 4.8 | Effect of memory overclocking on the performance benefits of a ROP executing SPEC2000 integer benchmarks | 68 |
| Table 4.9 | Effect of memory overclocking on the performance benefits of a ROP executing SPEC2000 floating point benchmarks | 68 |
| Table 5.1 | Mean Time To Failure (MTTF) for critical wear out models | 71 |</p>
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 5.2</td>
<td>Simulator parameters</td>
<td>73</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Technology scaling parameters</td>
<td>82</td>
</tr>
<tr>
<td>Table 6.2</td>
<td>Comparing various performance metrics across different technology nodes for</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td>a non-overclocked processor executing SPEC2000 integer benchmarks</td>
<td></td>
</tr>
<tr>
<td>Table 6.3</td>
<td>Comparing various performance metrics across different technology nodes for</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>a non-overclocked processor executing SPEC2000 floating point benchmarks</td>
<td></td>
</tr>
<tr>
<td>Table 7.1</td>
<td>Possible error scenarios</td>
<td>99</td>
</tr>
<tr>
<td>Table 7.2</td>
<td>Fault injection results</td>
<td>107</td>
</tr>
<tr>
<td>Table 7.3</td>
<td>Timing errors</td>
<td>108</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Cross section of a n-channel MOSFET in the ON state showing channel formation. The channel exhibits pinch-off near drain indicating operation in saturation (active) region.</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>Typical pipeline stage in a ROP. Local timing error detection and recovery scheme for critical registers is shown in detail.</td>
<td>19</td>
</tr>
<tr>
<td>2.3</td>
<td>Timing diagram showing overclocking advantage per cycle, as compared to the worst-case clock.</td>
<td>20</td>
</tr>
<tr>
<td>2.4</td>
<td>Timing diagram showing pipeline stage level timing speculation.</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Reduced overhead Razor flip-flop and metastability detection circuits (Figure reproduced from [27]).</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>SPRIT$^3$E framework.</td>
<td>25</td>
</tr>
<tr>
<td>3.1</td>
<td>Clock timing waveforms showing governing requirements, for $MAIN_{CLK}$ and $PS_{CLK}$, over the full range of overclocked aggressive frequencies ($F_{MIN}$ ⇐ $F_{MAX}$).</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>Examples of Main and PS clocks with variable and fixed phase shifts.</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Timing waveforms after increasing contamination delay to half the propagation delay for the full range of overclocked aggressive frequencies ($F_{MIN}$ ⇐ $F_{MAX}$).</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>8-bit CLA adder.</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Delay distribution for an 8-bit CLA adder.</td>
<td>34</td>
</tr>
<tr>
<td>3.6</td>
<td>8-bit CLA adder with additional delay blocks to increase contamination delay.</td>
<td>35</td>
</tr>
<tr>
<td>3.7</td>
<td>Delay distribution for an 8-bit CLA adder after increasing contamination delay.</td>
<td>36</td>
</tr>
<tr>
<td>3.8</td>
<td>Experimental setup to estimate performance improvement of CLA adder circuits.</td>
<td>37</td>
</tr>
</tbody>
</table>
Figure 3.9  Percent of error cycles versus clock period for an 8-bit delay added CLA adder circuit .............................................................. 38
Figure 3.10 Percent of error cycles versus clock period for a 32-bit delay added CLA adder circuit (Contamination delay 1.21 ns) .............................................. 39
Figure 3.11 Percent of error cycles versus clock period for a 32-bit delay added CLA adder circuit (Contamination delay 1.38 ns) .............................................. 39
Figure 3.12 Percent of error cycles versus clock period for a 64-bit delay added CLA adder circuit ........................................................................................................ 40

Figure 4.1 Alpha 21264 integer and floating point pipeline showing timing error detection and recovery circuit for critical registers .............................................. 41
Figure 4.2 Simulation framework ................................................................................................................................. 48
Figure 4.3 Cumulative error profile for all pipeline stages at overclocked operating frequencies for SPEC2000 integer benchmarks. Also shown separately are error profiles for issue stage and execute stage ................................................................. 51
Figure 4.4 Error profile for three SPEC2000 integer benchmarks executing five different instruction and data sets ........................................................................... 52
Figure 4.5 Feedback control system to dynamically tune clock frequency: Single clock generator with variable phase shift ........................................................................... 55
Figure 4.6 Feedback control system to dynamically tune clock frequency: Dual clock generators with fixed phase shift ........................................................................... 56
Figure 4.7 Run time, energy and energy-delay product trends for SPEC2000 integer benchmarks as target error rate varies from 0% to 20%. All values are normalized to 0% target error rate (no overclocking mode) ..................................................... 65
Figure 4.8 Run time, energy and energy-delay product trends for SPEC2000 floating point benchmarks as target error rate varies from 0% to 20%. All values are normalized to 0% target error rate (no overclocking mode) ..................................................... 66
Figure 5.1  Simulation framework depicting thermal throttling, alongside timing error based feedback control, for a reliably overclocked system .............................. 72

Figure 5.2  On-chip temperature trends and MTTF results for bzip2 benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling. ................................................................. 73

Figure 5.3  On-chip temperature trends and MTTF results for crafty benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling. ................................................................. 74

Figure 5.4  On-chip temperature trends and MTTF results for gzip benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling. ................................................................. 75

Figure 5.5  On-chip temperature trends and MTTF results for mcf benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling. ................................................................. 75

Figure 5.6  Relative performance for SPEC2000 integer benchmarks ............................. 77

Figure 6.1  Technology scaling vs. speculative reliable overclocking: Power consumption trends for SPEC2000 integer benchmarks. ................................................................. 86

Figure 6.2  Technology scaling vs. speculative reliable overclocking: Power consumption trends for SPEC2000 floating point benchmarks. ................................................................. 86

Figure 6.3  Technology scaling vs. speculative reliable overclocking: Temperature trends for SPEC2000 integer benchmarks. ................................................................. 87

Figure 6.4  Technology scaling vs. speculative reliable overclocking: Temperature trends for SPEC2000 floating point benchmarks. ................................................................. 87
Figure 6.5 Technology scaling vs. speculative reliable overclocking: Run time for SPEC2000 integer benchmarks. All values are normalized to 90nm run time. 88

Figure 6.6 Technology scaling vs. speculative reliable overclocking: Run time for SPEC2000 floating point benchmarks. All values are normalized to 90nm run time. 88

Figure 6.7 Technology scaling vs. speculative reliable overclocking: Energy consumption for SPEC2000 integer benchmarks. All values are normalized to 90nm energy values. 89

Figure 6.8 Technology scaling vs. speculative reliable overclocking: Energy consumption for SPEC2000 floating point benchmarks. All values are normalized to 90nm energy values. 89

Figure 6.9 Technology scaling vs. speculative reliable overclocking: EDP for SPEC2000 integer benchmarks. All values are normalized to 90nm EDP. 90

Figure 6.10 Technology scaling vs. speculative reliable overclocking: EDP for SPEC2000 floating point benchmarks. All values are normalized to 90nm EDP. 90

Figure 7.1 Conjoined Pipeline Architecture: Shaded region represents the L-Pipeline. Dotted line encompasses the Local Fault Detection and Recovery (LFDR) circuit 94

Figure 7.2 Waveforms highlighting error detection and recovery in a Conjoined Pipeline system 97

Figure 7.3 Dynamic frequency scaling 100

Figure 7.4 Modular implementation 104

Figure 7.5 Clock generation circuitry 105

Figure 7.6 Execution time for three different applications running on Conjoined Processor in various modes 109
ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis.

First and foremost, Dr. Arun K. Somani for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I am very thankful to him for giving me the liberty to pursue a research direction that I liked, and the guidance required to be successful at it. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Akhilesh Tyagi, Dr. Randall L. Geiger, Dr. Joseph Zambreno and Dr. David Fernandez-Baca. I am also thankful to Dr. Chris Chu and Dr. Shashi K. Gadia for being part of my program committee at various times. I am thankful to the many professors at Iowa State who taught me and provided me with sufficient knowledge to conduct this research. I am thankful to the ECPE department and Iowa State University for providing me with such a wonderful atmosphere for carrying out quality research.

I would like to acknowledge the contributions of my colleagues Premkumar Ramesh, Prasad Avirneni, Mikel Bezdek, Roy Lycke, and Adam Jackson in developing various aspects of this thesis. I am very much thankful to Prem and Prasad for making the research atmosphere lively, and without them, I would have found it extremely difficult to wade through the highs and lows of conducting research. I would like to thank Ganesh for his guidance throughout the initial stages of my graduate career. I am very much indebted to Kavitha Balasubramanian for being the painstaking proofreader of each one of my papers. I am thankful to my wife, Kamala, for helping me plot graphs and arrange the humongous data I collected from my simulation runs. I am thankful to my other research group colleagues Srivatsan, Mike, Kamna, Nathan, Koray, Jinxu, Ramon, Kritanjali, David, Nishanth, Pavan, and Zachary for the many wonderful discussions we had during the weekly seminars.
If not for my friends who made my stay at ISU so totally memorable and enjoyable, I would have found living so far away from everything I cared about extremely unbearable. I am forever grateful to these amazing folks - Vasanth, Puvi, Atul, Hari, Kavitha, Kamna, Prem, Prasad, KK, Muthu, Vatsan, Satya, Abhijit, Nishanth, Vichu, Shibhi, Nikhil, Niranjan, Rakesh, Rokkam, Satyadev, Richard, Srikanth, Sankalp volunteers, coffee room chitchatters and Friday evening volleyball gang. I am very grateful to Vasanth Balaramudu, who was my roommate during three years of my life at Iowa State, for his exceptional friendship, and I can say without any doubt, if not for his amazing culinary skills, I would have lost twenty pounds for sure. I am not really sure whether that is a boon or a bane, considering the rigor I am going through to lose that twenty pounds.

Finally, I would like to thank my parents, my wife, and family for their loving guidance and motivation during the writing of this work. I am forever indebted to my parents without whose support I would never have come this far.
ABSTRACT

Computers have changed our lives beyond our own imagination in the past several decades. The continued and progressive advancements in VLSI technology and numerous micro-architectural innovations have played a key role in the design of spectacular low-cost high performance computing systems that have become omnipresent in today’s technology driven world. Performance and dependability have become key concerns as these ubiquitous computing machines continue to drive our everyday life. Every application has unique demands, as they run in diverse operating environments. Dependable, aggressive and adaptive systems improve efficiency in terms of speed, reliability and energy consumption.

Traditional computing systems run at a fixed clock frequency, which is determined by taking into account the worst-case timing paths, operating conditions, and process variations. Timing speculation based reliable overclocking advocates going beyond worst-case limits to achieve best performance while not avoiding, but detecting and correcting a modest number of timing errors. The success of this design methodology relies on the fact that timing critical paths are rarely exercised in a design, and typical execution happens much faster than the timing requirements dictated by worst-case design methodology. Better-than-worst-case design methodology is advocated by several recent research pursuits, which exploit dependability techniques to enhance computer system performance.

In this dissertation, we address different aspects of timing speculation based adaptive reliable overclocking schemes, and evaluate their role in the design of low-cost, high performance, energy efficient and dependable systems. We visualize various control knobs in the design that can be favorably controlled to ensure different design targets.

As part of this research, we extend the SPRIT$^3$E, or Superscalar PeRformance Improvement Through Tolerating Timing Errors, framework, and characterize the extent of application dependent performance
acceleration achievable in superscalar processors by scrutinizing the various parameters that impact the operation beyond worst-case limits. We study the limitations imposed by short-path constraints on our technique, and present ways to exploit them to maximize performance gains. We analyze the sensitivity of our technique’s adaptiveness by exploring the necessary hardware requirements for dynamic overclocking schemes. Experimental analysis based on SPEC2000 benchmarks running on a SimpleScalar Alpha processor simulator, augmented with error rate data obtained from hardware simulations of a superscalar processor, are presented.

Even though reliable overclocking guarantees functional correctness, it leads to higher power consumption. As a consequence, reliable overclocking without considering on-chip temperatures will bring down the lifetime reliability of the chip. In this thesis, we analyze how reliable overclocking impacts the on-chip temperature of a microprocessor and evaluate the effects of overheating, due to such reliable dynamic frequency tuning mechanisms, on the lifetime reliability of these systems. We then evaluate the effect of performing thermal throttling, a technique that clamps the on-chip temperature below a predefined value, on system performance and reliability. Our study shows that a reliably overclocked system with dynamic thermal management achieves 25% performance improvement, while lasting for 14 years when being operated within 353K.

Over the past five decades, technology scaling, as predicted by Moore’s law, has been the bedrock of semiconductor technology evolution. The continued downscaling of CMOS technology to deep sub-micron gate lengths has been the primary reason for its dominance in today’s omnipresent silicon microchips. Even as the transition to the next technology node is indispensable, the initial cost and time associated in doing so presents a non-level playing field for the competitors in the semiconductor business. As part of this thesis, we evaluate the capability of speculative reliable overclocking mechanisms to maximize performance at a given technology level. We evaluate its competitiveness when compared to technology scaling, in terms of performance, power consumption, energy and energy delay product. We present a comprehensive comparison for integer and floating point SPEC2000 benchmarks running on a simulated Alpha processor at three different technology nodes in normal and enhanced modes. Our results suggest that adopting reliable overclocking strategies will help skip a technology node altogether, or be competitive in the market, while porting to the next technology node.
Reliability has become a serious concern as systems embrace nanometer technologies. In this dissertation, we propose a novel fault tolerant aggressive system that combines soft error protection and timing error tolerance. 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 over-clocking, 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, even under a severe fault injection campaign, achieves near 100% fault coverage and an average performance improvement of about 20%, when dynamically overclocked.
CHAPTER 1. INTRODUCTION

Microprocessors and application specific integrated circuits (ASICs) have evolved spectacularly since the early 1970s, paving the way for the digital revolution we continue to witness in our everyday life. The wide range of applicability of digital systems have subjected them to diverse demands, in terms of performance, power consumption and dependability, as they perform a plurality of tasks and run in a multitude of operating environments. These demands are interrelated and need to be addressed cohesively, as improving one metric alone is counterproductive for another. For instance, choosing operating voltage and frequency for performance, results in increased power consumption.

Technology scaling, in line with ”Moore’s Law”, has sustained for long the unparalleled growth of the semiconductor industry. Technology scaling leads to smaller transistors, higher packing densities, decreased supply voltages and increased clock frequencies; thereby contributing to the goals of higher performance and lower power consumption. However, with ultra deep sub-micron technologies, the integrated circuit reliability is impacted, as thinner circuits and wires are exercised more aggressively leading to premature device failures. In addition, some of the operating environments are hazardous to the working of the integrated electronics that forms the fabric of these computer systems.

Reliability issues in combinational logic have become more pronounced and their manifestations result in frequent error occurrence, as we rapidly adopt technological advancements [85]. A major reason for this is the increasing probability of longer single event transient (SET) pulses in newer technologies. Radiation induced SET pulses have widths in the range of 500\,ps to 900\,ps in the 90\,nm process, as compared to 400\,ps to 700\,ps in the 130\,nm process [66]. Control logic and random logic protection continues to be a major issue in the quest for highly dependable systems [77].

Traditional computing systems are designed to run reliably at discrete voltage and clock frequency settings, which are determined by taking into account the implementation technology, power budget,
worst-case timing paths, operating conditions and process variations. Dynamic voltage and frequency scaling (DVFS) techniques that choose one of the discrete voltage-frequency settings during run-time are commonly employed in today’s microprocessors, to reduce power consumption by enabling high performance only when demanded by the currently executing application. Adaptive techniques such as DVFS are beneficial since every application has unique demands.

For a given supply voltage, the clock frequency of a processor is selected such that there is sufficient time for the longest delay path to stabilize under adverse operating conditions. However, this worst-case propagation delay estimate is too conservative, as process, voltage and temperature variations that are introduced during circuit fabrication and operation have a wide range of possible values. Processor manufacturers assume the worst, while determining the critical operating voltage and frequency values. These operating levels are defined as critical because going beyond them may result in system crashes, due to erroneous computation and device malfunction, due to overheating. However, the rarity of occurrence of worst-case scenarios, combined with input data dependent circuit delays, led to techniques such as overclocking that exploit the latent best-case performance in computing systems [3, 54, 103].

Overclocking is a procedure in which the operating frequency is increased beyond manufacturer specified frequency limits for reliable operation, without changing the system supply voltage [19]. The interest in designs that operate beyond worst-case design settings started when PC tweakers in the early 1990s modified their existing computers to run at higher speeds, enabled by exotic cooling solutions. Over the last decade, gaming enthusiasts have embraced overclocking in their pursuit forever faster execution times and fantastic gaming experiences. During recent years, overclocking has become mainstream, with chipset manufacturers introducing technologies that support overclocking: AMD’s Overdrive and Advance Clock Calibration technologies are cases in point [1].

The gains from overclocking are possible because of the worst-case assumptions used by traditional design methodologies. However, systems running at overclocked speeds cannot be relied upon, as the possibility of a system failure always exists. As a result, to account for the timing errors that occur at overclocked speeds, it is important to overclock the system reliably, to make the common case faster [27, 92]. When reliably overclocked, performance benefits can be seen only if computed data is used at overclocked speeds. The concept of using data speculatively assuming no timing errors, is called
Timing speculation. Timing speculation based reliable overclocking mechanisms employ proven fault
tolerance techniques to detect and recover from timing errors [5, 8, 27, 33, 32, 92].

Future systems need to have crucial reliability enhancements to have high trust and dependability
on their computational correctness. Fault tolerance approaches strive to achieve high degree of fault
coverage, while being as conservative as possible in terms of area and power overhead. They attempt
to minimize performance degradation compared to a non fault-tolerant system. 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
[56]. With the advent of Chip Multiprocessors (CMP), fault tolerance techniques that also improve
performance have been developed [88, 95]. 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.

In this dissertation, we address various aspects of timing speculation based adaptive reliable over-
clocking schemes, and evaluate its role in the design of aggressive computer systems. The goal of this
thesis is to develop low-cost, high performance, energy efficient and dependable systems. We char-
acterize the various factors that influence the performance gains achievable through adaptive reliable
overclocking. We evaluate the effectiveness of reliable overclocking, as compared to technology scal-
ing. We also develop fault tolerant aggressive systems that have the twin goals of guaranteeing high
performance and fault tolerance.

1.1 High Performance Computing

In conformance with Moore’s law [63], the semiconductor industry has witnessed its effectiveness
doubling roughly every two years for the past several decades. This seemingly everlasting improve-
ment, in the performance of microprocessors and other digital systems, is being sustained by a host of
innovations at the manufacturing level, circuit level and micro-architectural level.
1.1.1 Device Scaling

Device scaling has enabled the doubling of transistor density and clock frequency with each technology generation [9, 24]. However, recent scaling trends indicate the need for new materials and manufacturing methodologies to circumvent the predicted scaling limits at ultra deep sub-micron technology nodes [96]. International Technology Roadmap for Semiconductors (ITRS) uses the term “technology node” to indicate overall industry progress in IC feature scaling [29]. Technology nodes used to be defined based on the smallest half-pitch of contacted metal lines on DRAM. However, for microprocessors and ASICs, technology feature size is indicated by the gate-length isolated feature size.

Current Intel microprocessors are manufactured using the 45nm process, while 32nm technology is targeted for Q4 2009 [45]. ITRS predicts that technology scaling will continue through the next decade, and nano-CMOS technology will remain the dominant implementation technology for silicon VLSI chips. Technology scaling enables the following: For every 30% downscaling of technology node, transistor density doubles, gate delay reduces by 30%, operating frequency improves by 43%, active power consumption halves, and energy savings of 65% is observed. However, power and process variations impose a limit on frequency scaling that is achievable by device scaling alone [23, 30]. Device scaling also complicates the on-chip communication as interconnects do not scale as well as logic gates [9]. Wire delays have started dominating the overall delay and in modern microprocessors, pipeline stages are dedicated only for moving signals across the chip.

1.1.2 Microprocessor Architectures

Micro-architectural innovations complement improvements in process technology, and contribute to the immense advancements in computer technology. To continue the pace of this growth, advances in microprocessor architectures are critical. An important factor that has contributed significantly to the improvement in microprocessor performance is instruction level parallelism (ILP). Superscalar processors issue more than one instruction in a given cycle thereby increasing the number of instructions being issued and executed in a cycle. These processors have hardware support to perform dynamic scheduling. Very Long Instruction Word (VLIW) machines are compiler assisted and execute more than one instruction in a cycle.
Along with device scaling that enable faster logic circuits, pipelining has played a major role in increasing clock frequency. Processor pipeline depths have increased from five stages to thirty stages. Deep pipelining or superpipelining [61] allows a faster clock frequency by dividing the clock limiting stages into multiple sub-stages. However, the benefits of superpipelining are limited by the penalties imposed by branch and other hazards. Intel Pentium 4 has a twenty stage mis-prediction pipeline [41]. Recent research suggests that clock frequency improvements from increasing pipeline depths are reaching a point of diminishing returns. It has been shown in [37] that while the optimal number of pipeline stages is application dependent, on an average, performance is maximized with around twenty stages.

Asynchronous design techniques present an alternative to synchronous design approaches [26]. For long these methodologies suffered because of lack of computer aided design tool support. In recent years, there have been efforts to automate the asynchronous design process. Counterflow pipeline processors present an interesting approach towards asynchronous microprocessor implementations [89]. However, asynchronous designs are not easy to comprehend and present several implementation bottlenecks [38]. In addition, aggressive time-to-market schedules and significantly low product lifetime make asynchronous design methodologies unattractive.

1.1.3 Better-Than-Worst-Case Designs

Reliable overclocking allows embedded systems and processors to run at higher frequencies than the manufacturer specified worst-case limits. The vendor specified frequency includes a safety margin to provide tolerance for process variations, voltage fluctuations and extreme temperatures. For systems operating in typical operating environments, significant benefits can be achieved through overclocking, if reliable execution can be guaranteed. Also, frequency binning is used by manufacturers to sort fabricated devices based on their speed. The discrete speed grades and inter-die process variations can be exploited by overclocking to improve performance [32].

The most significant aspect that is exploited by reliable overclocking is the input data dependency of the worst-case delays. The worst-case delay paths are sensitized only for specific input combinations and data sequences [3]. Typically, the propagation delay of the digital system is much less, than
the worst-case delay and this can be exploited by overclocking. The benefits of overclocking can be furthered by allowing a tolerable number of errors to occur, and have an efficient mechanism to detect and recover from those errors. This technique along with dynamic voltage scaling has been used to improve energy efficiency [27]. Along with adaptive clocking mechanisms, reliable overclocking improves performance drastically [92]. In [58], the trade-off between reliability and performance is studied, and overclocking is used to improve the performance of register files.

In [54], the issue, register renaming, and ALU logic of a superscalar processor are replaced with approximate versions that execute in half the time, but not necessarily correct. Two versions of the original ALU and register renaming logic are required to detect errors in the approximate versions. Timing error avoidance techniques that overclock, but within safe limits also have been proposed [99].

1.1.4 Adaptive Systems

Increasingly miniaturized systems and higher frequencies of operation have resulted in increased overall power dissipation for the same chip size. Power and energy aware computing approaches scale supply voltage to reduce dynamic power dissipation. This approach also lowers the frequency so that errors are avoided. Dynamic frequency scaling along with dynamic voltage scaling allows energy-performance trade-offs during run time.

Over the last decade, overclocking as a means to improve processor performance is gaining popularity [19]. Overclocking does not guarantee computational correctness. Hence, it is necessary to develop solutions that reliably and dynamically adjust frequency to the optimal value. Having a clock signal whose frequency adapts well to the environment, as well as the application characteristics empowers digital systems to capitalize on significant performance benefits in terms of execution time. Such dynamic clock frequency schemes have been proposed earlier for ASICs [34] and Field Programmable Gate Arrays (FPGAs) [12].

Adaptive overclocking adjusts frequency for variations in process and environmental conditions during run-time. Also, worst-case conditions occur rarely, leaving room for significant performance improvements that can be achieved through dynamically adjusting clock frequency at run time beyond worst-case limits. In [59], a theoretical control technique for a variable speed processor is presented.
Triple-latch monitor based designs that perform hardware self tuning based on circuit performance monitoring have the capability to adapt to process and temperature variations [48].

### 1.2 Fault Tolerant Computing

Along with performance, reliability is becoming one of the preeminent concerns for computer architects. As scaling in VLSI technology continues into the nanometer regime, both memory elements and combinational gates become susceptible to soft errors due to the transient pulses induced by radiation that have durations often higher than the gate propagation delays [7]. This electric pulse can propagate without masking, get latched in a memory element and may result in an error (soft error) at the application level, hence resulting in unfavorable system behavior. Further, higher clock speeds decrease the cycle time, increasing the probability that a soft error is latched. These trends imply that future digital systems need to be protected against both single event transients (SETs) and single event upsets (SEUs) [47].

Laprie [52] defines fault as an erroneous state of hardware or software, while an error as the manifestation of a fault. A failure occurs when the actual operation deviates from the desired operation. Not all faults become errors, and not all errors lead to failure [102]. Lately, reliability issues have become more pronounced and their manifestations result in frequent error occurrence, as we rapidly adopt technological advancements [85]. Future systems need to have crucial reliability enhancements to have high trust and dependability on their computational correctness. Fault tolerance architectures have become more attractive than fault avoidance architectures as performance takes center stage.

The impact of soft errors and silicon defects on system reliability have been steadily rising as we progress towards 32\(\text{nm}\) technologies and beyond. Soft errors, which are transient in nature, and silicon defects, which lead to permanent failures, have intrigued researchers to formulate fault tolerance techniques with varied capabilities to improve system reliability. Soft errors, induced by high energy radiation and external noise, have become more frequent and may result in incorrect computation and silent data corruption. Intermittent faults are also a cause for concern [20]. Silicon defects resulting from silicon failure mechanisms such as transistor wear outs, gate breakdown, hot carrier degradation, and manufacturing limitations reduce lifetime and reliability of fabricated devices [57].
1.2.1 Transient Faults

A host of research work exists that characterizes the effects of transient faults on a high-performance processor pipeline. An analysis of soft error sensitivity for picoJava-II, a microprocessor core developed by Sun Microsystems, has been performed in [51]. This work asserts that reasonable prediction of soft error sensitivity can be made by means of deduction from the processor’s microarchitecture leading to efficient implementations of redundancy for various logic blocks. Similar characterizations have been performed on other commercial microprocessors [81, 101].

The fact that not all transient faults translate to system failure has led to concepts such as architectural vulnerability factor (AVF) [65]. AVF is calculated for different hardware structures based on their usage and presence of valid data in them, and it is defined as the probability that a fault occurring in a particular structure will result in an error. Calculating AVF for various structures enables selective switch-on of fault tolerance mechanisms, based on the AVF at that point of time. The reason for AVF to be different at various times is because a soft error happening, for instance, on dead code or wrong-path instructions may be inconsequential or may propagate without affecting the end result of a computation. Natural fault masking, resulting from logical masking, electrical masking, and latching window masking [85], also prevent a fault from manifesting as an error.

Prior research indicates that memories are more susceptible to faults than the processor pipeline itself [39, 98]. Microprocessor datapath protection has been well researched. However, control logic and random logic protection continues to be a major issue for high performance systems [77, 49]. Currently, a given single-bit is expected to flip in a RAM only once every many billions of hours of operation. However, with the growing sizes of RAMs and other hardware components, one can expect the error frequency to become much more noticeable in the near future. Also, processors deployed in hazardous environments will have a higher probability of getting affected by bit-flips.

1.2.2 Redundancy Techniques

Various components in a processor are protected using different types of redundancy techniques. Redundancy can be broadly classified as information redundancy, spatial redundancy and temporal redundancy.
Information redundancy involves generating extra code bits from data and appending them to the data before storing it. During data retrieval, the code is regenerated and cross-checked to detect errors. Error correcting codes that are capable of single error correction and double error detection are used to ensure high availability and dependability of memories [18].

Spatial redundancy is achieved by performing the same computation on multiple independent functional units at the same time. Concurrent error detection is achieved by duplication and comparison, and recovery is triggered whenever mismatch happens [62].

Temporal redundancy works by repeating the computation on the same hardware multiple times and comparing the execution results to flag errors. While spatial methods can tolerate both transient and permanent errors, temporal techniques only work for transient errors [72]. All such techniques offer fault tolerance by either performing fault detection coupled with recovery or by providing fault masking. The choice of redundancy is based on several parameters including cost, performance, and fault coverage.

1.2.3 Fault Mitigation Techniques

Soft errors are induced by alpha particle emissions from chip packaging, high energy particle strikes, which includes neutrons from cosmic rays, and electrical noise. Several factors, such as the energy of the incoming particle, the geometry of the impact, the location of the strike, and the design of the logic circuit, determine the occurrence of a soft error. The combination of capacitance and voltage at a transistor node, described by the critical charge parameter $Q_{\text{crit}}$, determines the minimum electron charge disturbance needed to change the logic level. With device scaling $Q_{\text{crit}}$ decreases, but the charge collection efficiency of transistors also goes down [82]. Radiation hardening techniques are used to minimize the impact of soft errors on digital circuits [43].

Soft error detection and recovery in the IBM Z990 processors is discussed in [56]. AR-SMT [76] proposes using the multi threading capability of modern processors to execute the program and a duplicate of the program in parallel as two threads. The two threads execute simultaneously and the results are compared after a fixed period of time. If the results mismatch, the system goes back to a checkpointed state and starts re-execution. Many such proposed techniques protect the core datapath against
faults. Schemes, which provide for fault detection and correction higher up in the design hierarchy usually, have higher penalties for recovery in comparison to fault masking and schemes implemented at a lower level. Techniques that propose architectural level solutions primarily use checkpointing for recovery, while circuit level schemes implement in-situ error detection and correction mechanisms.

Protection of FSM based control logic by error masking has been discussed in [21]. A signature caching scheme was proposed in [49] to detect SEUs in the control logic of complex microprocessors. The ReStore architecture [100] uses checkpointing and rollback to recover from soft errors. The rollback is done based on certain abnormal events such as exceptions and incorrect control flow. A scheme to protect the static output bits of the instruction decode logic in a RISC architecture during loop execution is proposed in [77].

Temporal schemes developed for processors are designed to provide tolerance to transient sources of errors, such as single event upsets caused by high energy particle strikes. A good example of temporal redundancy is REdundant Execution using Spare Elements (REESE) [68]. In this architecture, all instructions reaching the commit stage in a superscalar processor are re-executed by the same hardware in a redundant stream.

1.2.4 Exploiting Fault Tolerance to Improve Performance

Fault tolerance approaches strive to achieve high degree of fault coverage, while being as conservative as possible in terms of area and power overhead. They attempt to minimize performance degradation compared to a non fault-tolerant system. Commercial systems opt for low overhead approaches that provide limited fault coverage, while incurring a modest performance penalty. Mission critical systems with hard real-time constraints require extensive fault coverage and no compromise in performance. However, the performance degradation associated with high reliability solutions force systems requiring high performance to, sometimes, sacrifice reliability.

With the advent of Chip Multiprocessors (CMP), fault tolerance techniques that also improve performance have been developed [88, 95, 108]. 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. The speedup is achieved by exchanging control and data flow information between
the two cores. Here, execution is rolled backed to a checkpointed state and instructions are re-executed to recover from a fault.

Several of these proposed architectures apply fault tolerance with the goal of improving performance past worst-case limits. The Selective Series Duplex architecture [50] 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. DIVA [5] uses spatial redundancy by providing a separate, slower pipeline processor alongside the fast processor.

A body of work exists that utilize two cores to speedup performance and/or improve fault tolerance. Architectures such as, Slipstream [95], Dual-core Execution [107], Future Execution [31], and Reunion [88] exchange control and data flow information between the cores to speed up execution, while leveraging the redundancy to provide partial fault coverage. Paceline [32] performs overclocking of the cores to speed up execution.

### 1.3 Power/Thermal Aware Computing

The need for low power architectures that deliver high performance while consuming as less power as possible is increasingly being felt by embedded system designers as they try to pack more and more power intensive computational tasks, while curtailing their power budgets. One of the most effective and widely used techniques for power aware computing is dynamic voltage scaling (DVS). Supply voltage can be lowered during processor idle times. To reduce supply voltage, clock frequency needs to be reduced first. Dynamic voltage and frequency scaling together narrows the gap between high performance and low power requirements [69]. As dynamic energy scales quadratically with supply voltage, significant energy reduction is possible by lowering the supply voltage [64].

Industry standards such as Intel SpeedStep [75], AMD PowerNow [2], Transmeta Longrun [28] technologies alternate between a set of predefined voltage and frequency pairs and choose the best pair based on the worst-case voltage, temperature and process conditions. Several monitoring techniques are employed in modern microprocessors to achieve dynamic voltage and frequency scaling based on the current workload of the processor. These run time schemes use the outputs of performance counters, thermal sensors, and ring oscillators that are embedded at strategic locations on the chip.
Correlating voltage controlled oscillator approaches have been proposed wherein the oscillator speed automatically adapts based on the supply voltage and generates the fastest safe clock speed [16, 35]. More aggressive power reduction can be achieved by tuning the supply voltage of individual processor chip using embedded inverter delay chains [25].

The RAZOR architecture [22, 27] uses temporal fault tolerance by replicating critical pipeline registers in order to dynamically scale voltage past its worst-case limits. Razor achieves lower energy consumption by reducing supply voltage in each pipeline stage. In [83], a multiple clock domain approach for processor design is presented for improving energy efficiency. This work supports locally synchronous, globally asynchronous design technique. The goal is to run different parts of the processor with different clocks and use existing microprocessor queue structures for inter domain communication.

In recent years, thermal aware computing is becoming as important as power aware computing. The initial attempts, to bring down on-chip temperature, seek to minimize power. However, meeting power budget requirements is not sufficient and cooling mechanisms are not cost effective. This created a necessity for a control mechanism built within the processor chips, which is effective and economically viable. Designs began to include thermal sensors in various locations on a processor chip [74]. These dynamic control mechanisms effectively manage temperature, but suffer considerable overhead. Follow on research started to focus on the design of thermally aware high performance processors aiming to minimize performance impact for specific applications [14]. Dynamic Thermal Management (DTM) schemes are presented in [87] to make run-time decisions at different levels.

### 1.4 Contributions of this Thesis

The ideas in this thesis stem from the following observations:

- To provide reliable execution, traditional design methodologies ensure timing error avoidance by designing to accommodate the worst-case parameters. However, in practice the worst-cases are rare, leading to a large amount of exploitable performance improvement, if timing errors can be detected and recovered from. Several architectural design concepts have been recently proposed.
With techniques to speedup circuit operation, the overall performance of a digital system is going up, but on-chip thermal management is becoming an issue.

Semiconductor technology evolution enables high speed, low power systems, but the design costs and time-to-market are becoming worse with deep sub-micron technology generations. A supplementary approach that can provide stopgap relief is beneficial.

As the advances in semiconductor technology allow continued scaling of VLSI implementation, the possibilities for encountering a soft error also increases. As a result, most digital systems need to incorporate soft error tolerance mechanisms.

This thesis discusses the potential and limitation of reliable overclocking as a viable technique to enhance mainstream system performance. Potential solutions that can overcome the limitations are explored. This thesis brings forth the thermal issues related to overclocking. This thesis projects speculative reliable overclocking as a cost-efficient competitive stopgap alternative to technology scaling. The design of adaptive clocking mechanisms that aid in dynamic frequency switching without incurring any penalty cycles are also explored. This thesis also explores the possibility of coupling the timing error tolerance mechanism with transient, intermittent and permanent fault tolerance mechanisms to design highly reliable high performance systems.

In this chapter, we presented an introduction to various innovative techniques that have improved computer system performance over the years. We also presented an introduction to fault tolerant computing, and introduced the reliability issues that affect VLSI circuits. We surveyed the various techniques that have been proposed in the past to improve system performance and reliability. We also looked at the on-going research work in power and thermal aware computing. In this chapter, we also introduced the better-than-worst-case design methodology and reliable overclocking that form the basis of the research work presented in this thesis.

In Chapter 2, a brief description of relevant work pertaining to the research work reported in this thesis is presented. We analyze the issues presented by parameter variations in transistors, and how technology scaling exacerbates this problem. We then look at reliable overclocking in detail, explaining the timing error detection and recovery methodology, timing error rate based feedback control
system and timing speculation. We briefly present the Razor [27] framework, which first introduced circuit-level timing speculation, and look at how Razor enabled energy savings by exploiting the data dependence of circuit delay. We also briefly present the metastability problem, and look at the circuit-level metastability mitigation technique presented as part of the Razor circuit. Finally, we present the SPRIT³E framework for reliably overclocking a superscalar processor. The SPRIT³E framework developed by Mikel Bezdek, as part of his Master’s thesis [8], forms the primary basis on which the ideas for this dissertation work materialized.

The contributions of this dissertation are presented in the following chapters:

In Chapter 3, we present the limitation, imposed by short paths in the circuit, on reliable overclocking. We explore possibilities to manipulate the contamination delay of the circuit to maximize the performance gains achievable through reliable overclocking. To start with, we present a description of the clocking system used to generate the main clock and the backup clock that support speculative overclocking. We then present how contamination delay limits the extent of overclocking. To analyze the benefits of manipulating contamination delay in digital logic circuits to maximize the benefits of reliable overclocking, we did a case study on Carry Look Ahead (CLA) adders. CLA adders are chosen as they have very low contamination delay. We first increase the contamination delay, and then perform frequency scaling to operate the CLA adder at higher than worst-case speeds. Increasing contamination delay involves adding delays to the circuit. This increases the area and power overhead of the circuit. To minimize this overhead, we look at the optimal value for contamination delay, since increasing contamination delay results in higher number of timing errors for the same amount of overclocking performed when the contamination delay is not increased.

In Chapter 4, we characterize the extent of performance enhancement achievable in computer systems by dynamically varying the operating frequency past worst-case limits. We present an analysis framework and discuss in detail the nuances of designing a reliably overclocking system. One of our key objectives is to see the effect of overclocking on superscalar processors for various benchmark applications, and analyze the associated overhead in terms of extra hardware and error recovery penalty. We analyze the sensitivity of our technique’s adaptiveness by exploring the necessary hardware requirements for dynamic overclocking and dynamic frequency scaling schemes. We exploit the
data-dependent variance in circuit delay to achieve better-than-worst-case performance using adaptive reliable overclocking methodology. Experimental analysis based on integer and floating point SPEC2000 benchmarks running on a SimpleScalar Alpha processor simulator, augmented with error rate data obtained from hardware simulations of a superscalar processor, are presented.

In Chapter 5, we analyze the temperature pattern of reliably overclocked systems, and evaluate the lifetime reliability of such reliable aggressive clocking mechanisms. We monitor the on-chip temperature of aggressively overclocked systems that dynamically enhance single threaded application performance. We couple thermal monitoring techniques with reliable overclocking to alleviate lateral issues relating to system power and reliability.

In Chapter 6, we discuss the results we obtained for speculative reliable overclocking and technology scaling. We compare reliable overclocking with technology scaling, and evaluate its competitiveness vis-à-vis technology scaling. We start with an overview of technology scaling, and then present speculative reliable overclocking as a bridge between two successive technology generations. This alternative has the potential to reduce time-to-market, and act as a stopgap technique for performance enhancement in between two technology generations, or help skip a technology generation altogether.

In Chapter 7, we present a conjoined duplex system approach to provide tolerance for myriad hardware faults that plague modern computing systems. Our approach is capable of protecting both the datapath and control logic. Our conjoined pipeline system is capable of recovering from timing errors as well, 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 CPipe architecture, pipeline datapath and the error detection and recovery methodology is described in detail. We derive the relevant parameters that affect dynamic frequency scaling and the possible range of operating frequencies. Finally, we present the system implementation issues, and our implementation of a conjoined two-stage arithmetic pipeline and a conjoined five-stage DLX pipelined processor.

In Chapter 8, we conclude the research conducted in this dissertation and present the possible avenues, this research work presents for the future.
CHAPTER 2. BACKGROUND

In this chapter, the background material required for understanding the rest of this thesis is presented. This chapter covers parameter variations and the need for self-tuning systems. Timing error avoidance mechanisms tune frequencies until timing errors start to happen. Reliable overclocking techniques allow timing errors to happen, and have inbuilt error tolerance mechanisms to guarantee functional correctness. Timing speculation aids aggressive systems by allowing data to be used speculatively. Ernst, et al. in their work “Razor: A low-power pipeline based on circuit-level timing speculation” [27], introduced timing speculation and combined it with timing error tolerance methodology to achieve energy efficiency. Our work, SPRIT\textsuperscript{3}E [8, 92], extends reliable overclocking as a viable technique for performance enhancement in superscalar processors.

2.1 Parameter Variations

Ultra sub-micron process technologies force designers to adopt worst-case design methodologies that require safety margins to be added to individual system components to address parameter variations. Parameter variations include systematic and random variations in process, voltage, and temperature (PVT) [11]. These variations that affect propagation delay can be classified as physical variations, which are introduced during fabrication, and environmental variations, which are introduced during run-time [67]. Physical variations lead to both inter-die and intra-die variations. Environmental variations such as variations in temperature and power supply voltage also have an effect on the delay through any path.

During fabrication, the difficulties in minutely controlling the various processing steps contribute to process variations. Some of them, such as lithographic irregularities, are systematic in nature, while others, such as dopant fluctuations, lead to random effects [80]. Voltage fluctuations arise because
of variations in power supply, and also because of change in the capacitive load being driven during run-time [10]. Temperature variations are a direct effect of chip heating up and cooling down during operation.

Inter-die or die-to-die variations that affect two different chips, containing the same circuit, have led to the, now very common, semiconductor industry marketing technique called speed binning or frequency binning. Processor and memory manufactures employ speed binning to test their products for specific timing capabilities and bin them according to their tested frequencies. Process variations contribute to 30% variation in chip frequency [11, 13]. Intra-die or within-die variations, which are dependent on design implementation, are mostly caused by variations in effective gate length of transistors present in the same die. Die-to-die voltage fluctuations and within-die thermal variations also exacerbate this problem.

Parameter variations are becoming a key concern for circuit designers because they affect two key transistor parameters that dictate circuit performance: the threshold voltage, $V_T$, and effective gate length, $L_{\text{eff}}$. Gate length ($L$) is the physical distance between source ($S$) and drain ($D$) regions of a MOS transistor, shown in Figure 2.1, and when determined from actual transistor characteristics, it is referred to as “effective”. Threshold voltage is dependent on temperature and it affects both the frequency and leakage power.

Leakage power, also referred to as static power, in MOS transistors happens because of gate leakage current and sub-threshold leakage current. As the gate oxide thickness, $t_{\text{ox}}$, scales with newer process technologies, the magnitude of the gate leakage current increases, contributing to increased leakage power. Process level techniques such as the use of high-k dielectrics have alleviated the gate leakage problem. The other dominant leakage mechanism is due to the drain-source sub-threshold current. Sub-threshold current increases with increase in threshold voltage. As temperature varies, threshold voltage varies, leading to an exponential dependence of leakage power on operating temperature.

To account for these parameter variations, designers often assume delays $3\sigma$ from the typical delay. The deviation of process, voltage and temperature parameters from nominal specifications can only be statistically estimated when fixing the frequency of a circuit.
2.2 Reliable Overclocking

In a reliably overclocked processor (ROP), to tolerate timing errors, registers in the critical paths of every pipeline stage are augmented with a second time-delayed register. A typical pipeline stage in such a processor, along with local timing error detection and recovery circuit augmentation for critical path registers, is shown in Figure 2.2. Each combinational logic stage is a dense logic combination with multiple inputs and outputs, and possibly with more than one path from each input to output. The short paths in the logic can operate correctly even during extreme voltage and/or frequency scaling. The paths that are not likely to meet their timing requirements are categorized as critical paths and only their corresponding stage output registers are replaced with timing error detection and recovery circuits.

2.2.1 Timing Error Detection and Recovery

A brief description, of how reliable overclocking is achieved, is presented from [8]: The main register is clocked ambitiously by the $MAIN_{CLK}$ at a frequency higher than that required for error-free operation. The backup register is clocked in such a way that it is prevented from being affected by
Figure 2.2 Typical pipeline stage in a ROP. Local timing error detection and recovery scheme for critical registers is shown in detail.

timing errors, and its output is considered “gold.” The clock for this register is phase shifted, shown as $P_{CLK}$, such that the combinational logic is effectively given its full, worst-case propagation delay time to execute. In case of a mismatch between the primary and backup registers, a recovery measure is taken by correcting the current stage data and stalling the pipeline for one cycle. In addition to local recovery, action is also taken on a global scale to maintain correct execution of the pipeline in the event of a timing error. The extent to which systems can be overclocked is limited by the penalty cycles needed to recover from timing errors. A balance must be maintained between the number of cycles lost to error recovery and the gains of overclocking. The achievable performance enhancement per cycle, compared to the worst-case clock, $W_{CLK}$, is shown in Figure 2.3 as $\Phi_2$. $W_{CLK}$ is shown only for comparison purposes, and is not required during operation. One important factor that needs to be addressed while phase shifting the $P_{CLK}$ is to limit the amount of phase shift within the fastest delay path of the circuit. In Chapter 3, how short paths limit frequency scaling and how we can overcome this limitation is dealt with in detail.
2.2.2 Timing Error Rate Based Feedback Control System

Reliably overclocking a processor may not yield an increase in performance at all times. The amount of aggressive overclocking is strongly influenced by the number of input combinations responsible for the longer timing-paths. As frequency is scaled higher dynamically, more number of input combinations would result in error. The percent of clock cycles affected by errors impact the performance. Each time an error occurs, additional time is required to recover from that error. In addition, the occurrence of a timing error is highly dependent on the workload and the current operating conditions. It is beneficial to have an adaptive clock tuning system, which increases or decreases the clock frequency based on a set target error rate.

In a ROP, dynamic clock frequency tuning is controlled by a global feedback system based on the total number of timing errors that occur in a specified time interval. The number of errors occurring at each timing error counter sampling interval is continuously monitored. As long as the number of errors is within the target limit, the frequency is scaled up, else scaled down. One can apparently construe that the error rate is a monotonically increasing function with respect to frequency. In Chapter 4, adaptive clocking techniques for reliably overclocked systems is discussed in detail.

2.2.3 Timing Speculation

The most significant aspect that is exploited by reliable overclocking is the input data dependency of the worst-case delays. The worst-case delay paths are sensitized only for specific few input combinations and data sequences [3]. Allowing a tolerable number of errors to occur, and incorporating an
efficient mechanism to detect and recover from those errors are the key elements [27, 92]. Based on this observation, numerous architectures have been proposed over the years. One of the earliest works on aggressive clocking, TEATIME [99] scales the frequency of a pipeline using dynamic timing error avoidance. This technique attempts to achieve better-than-worst-case performance by realizing typical delay operation rather than assuming worst-case delays and operating conditions.

When reliably overclocked, performance benefits can be seen only if computed data is used at overclocked speeds. Timing speculation is a technique wherein data generated at aggressive speeds is sent forward speculatively assuming error free operation, and when an error is detected, the forwarded data is voided and the computation is redone. Circuit level speculation is proposed initially in Razor, and have been applied to superscalar processors in SPRIT$. Figure 2.4 shows timing waveforms that depict pipeline stage level timing speculation. In the figure, $inst_0$ moves forward speculatively without any timing errors. However, $inst_1$ encounters a timing error in $Stage \ i$, indicated by corrupted data “terr”. This error is detected by the error detection mechanism, and the stage error signal is asserted. This stage error signal triggers a local and global recovery. Timing error recovery flushes the data sent forward speculatively, indicated in the figure as “xxx”, and voids the computation performed by $Stage \ i + 1$. Once the timing error is fixed, the pipeline execution continues normally. The values $\Phi_1$, $\Phi_2$, $\Phi_3$ and $\Phi_4$ are explained in Figure 2.3.

![Figure 2.4 Timing diagram showing pipeline stage level timing speculation](image-url)
Apart from the run-time schemes, there are static methods that are specifically developed for better-than-worst-case architectures. BlueShift [33] proposes a design methodology from ground up for timing speculation. The main idea is to identify and optimize the frequently used critical paths, called the ‘overshooters’ at the expense of the lesser frequent ones. Timing speculation has been well studied in the chip multiprocessors as well. Generally, these techniques couple two cores, such that one of them is sped-up with the help of the other [32, 95, 88]. Other works in the domain seek to improve reliability and common case performance through functionally incorrect design [5, 54, 60].

The desire for better than worst-case designs is much more serious in nanoscale technology. Process, voltage and temperature variations within and across the die are causing the bottleneck while selecting the worst-case frequency. ReCycle uses additional registers and clock buffers to apply cycle time stealing in the pipeline from faster stages to the slower ones [97]. Another technique, EVAL, has been proposed to maximize performance with low power overhead in the presence of timing induced errors [79].

### 2.3 Razor Architecture

Razor employs dynamic voltage scaling (DVS) technique along with timing speculation, error detection and correction mechanism to recover from timing errors, thereby eliminating the need for voltage margins and exploiting the data dependence of circuit delay. Razor permits operation at sub-critical voltages by tolerating circuit timing errors and guaranteeing correct operation of the processor pipeline. As voltage is scaled lower and lower, the number of errors increase resulting in increased power consumption and decreased instruction throughput because of the associated error recovery penalty. To counter this, Razor tunes the supply voltage based on error monitoring and feedback control achieving significant power savings. Also, an error recovery technique based on counterflow pipeline methodology is proposed.

In Razor, pipeline registers are augmented with a shadow latch, instead of a register. The working of a razor flip-flop, consisting of a main register and shadow latch, is similar to the description provided in Section 2.2.1. The shadow latch is clocked by a delayed version of the main register clock. The clock timing requirements guarantee that the shadow latch is not corrupted with incorrect value when
operating at better-than-worst-case conditions. This scheme is most suitable to deal with multiple bidirectional (0 to 1 and 1 to 0) errors [73]. To eliminate clock distribution overhead for the delayed clock, the shadow latch is clocked by the negative edge of the main register clock. Also, the minimum path constraints needs to be taken care of and to this end buffers are inserted during synthesis stage with a power overhead of around 3%.

It is important for clocked flip-flops to respect setup and hold time criteria. Otherwise, metastability might occur, during which the flip-flop’s output may take an indefinite period of time to settle down to its correct state; possibly oscillating multiple times between the stable states 0 and 1. To reduce power overhead and handle metastability issues in the main register the Razor flip-flop has special circuit-level implementations as shown in Figure 2.5.

![Figure 2.5 Reduced overhead Razor flip-flop and metastability detection circuits](Figure reproduced from [27])

In a 64-bit Alpha processor only 192 flip-flops out of 2048 flip-flops required Razor augmentation. This amounted to a power overhead of 1%. Razor architecture was analyzed at various levels: First, an 18x18 bit multiplier is implemented and analyzed in FPGA. Then a C-level timing model of a Kogge-Stone adder is implemented with Razor timing details from SPICE analysis. This C-model is then integrated into the execution stage of simplescalar simulator [4]. Overall, substantial energy savings of up to 62% was observed with less than 3% impact on performance due to error recovery.
In [22], by generating an asymmetric clock, the duration of the positive clock phase is varied. Also an internal core frequency generator is available which is capable of generating clocks at different frequencies from 60 MHz to 400 MHz in steps of 20 MHz, and the duration of the positive clock phase is configurable from 0 ns to 3.5 ns in steps of 500 ps.

Though a prototype circuit was fabricated, the hardware was only verified with simple programs. Architectural simulations reflect only the energy savings for a Razor augmented adder. The clock frequency is not changed and it is fixed before processor operation. The minimum path constraints problem is not fully addressed.

2.4 SPRIT$^{3}$E Framework

The Superscalar PeRformance Improvement Through Tolerating Timing Errors (SPRIT$^{3}$E) framework allows the clock frequency of a superscalar processor to be dynamically tuned to its optimal value, beyond the worst-case limit. SPRIT$^{3}$E framework mainly exploits the dependence of critical path on input data, and makes the common case faster. Because the frequency is dynamically modified as the processor is running, variations in the environmental conditions, such as temperature and voltage, as well as variations present from fabrication, are automatically adjusted for. As frequency scales to higher values, timing errors will begin to occur. To prevent these errors from corrupting the execution of the processor, fault tolerance in the form of temporal redundancy is used. Specifically, pipeline stages are augmented with a local fault detection and recovery (LFDR) circuit. As frequency is scaled higher dynamically, more number of input combinations would result in error. Each time an error occurs, additional time is required to recover from that error. The error rate is monitored during run time, and based on a set tolerable error rate that does not affect the performance, the clock frequency is adjusted dynamically. The timing error mitigation scheme is similar to the one used in Razor [27].

SPRIT$^{3}$E technique applied to a superscalar processor is shown in Figure 2.6. To mitigate the timing errors, every pipeline stage is augmented with a second, time-delayed register. The LFDR circuit is highlighted in the figure. The main register is clocked ambitiously by the MAIN$^{3}$CLK at a frequency higher than that required for error-free operation. The backup register is clocked in such a way that it is prevented from being affected by timing errors, and its output is considered “gold.” The
clock for this register is phase shifted, shown as $PS_{CLK}$, such that the combinational logic is effectively given its full, worst-case propagation delay time to execute.

In addition to local recovery, action must be taken on a global scale to maintain correct execution of the pipeline in the event of a timing error. Mechanisms are identified to recover errors in all pipeline stages [92]. Because errors are detected quickly and the recovery technique utilizes many existing paths through the processor, the area and performance overhead incurred from allowing timing errors to occur is kept to a minimum. The current approach keeps area overhead for timing error detection low by reusing the combinational logic and by duplicating only critical pipeline registers. Circuitry added to perform global error recovery is modest, since the logic involved is not complex and reuses already existing signals in the pipeline. Overall, SPRIT$^3$E provides a viable means of tolerating timing errors.

To gauge the performance improvements provided by the SPRIT$^3$E framework, an initial sequence of experiments has been performed. The first experiment was done on a simple multiplier circuit that established that significant room for improvement does indeed exist. Using a method such as dual latching to tolerate a small amount of timing errors allowed the multiplier circuit to run at almost half the period, a speedup of 44%. SPRIT$^3$E framework was also evaluated in a DLX superscalar processor. The evaluation was done for three different benchmarks. Experimental results show that on an average a performance gain up to 57% across all benchmark applications is achievable.

Figure 2.6   SPRIT$^3$E framework
As demonstrated by successful overclocking, the current practice to set the frequency for synchronous circuits is far too conservative. At the same time, fault tolerance is necessary to ensure reliability if timing errors are not avoided with worst-case margins. The SPRIT$^3$E framework addresses these problems with relatively simple additions to the superscalar pipeline. Only the pipeline registers are duplicated and the large combinational logic blocks making up the stages are reused by utilizing temporal redundancy. Additional overhead comes from the error recovery logic, but this too may be kept modest by reusing existing pipeline signals whenever possible. All in all, the performance gained by operating at the optimal, sub-worst-case period more than justifies the overhead of the detection and recovery logic.
CHAPTER 3. MANIPULATING SHORT-PATHS FOR PERFORMANCE

The cardinal factor that limits frequency scaling, as described in the previous chapter for Razor and SPRIT$^3$E frameworks, is the contamination delay of the circuit. Contamination delay 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. The major hurdle imposed by the short-paths in the circuit is to limit the phase shift of the backup clock by the contamination delay of the circuit. The phase shift of the delayed clock is restricted below the contamination delay to prevent incorrect result from being latched in the backup register. Reliable execution can be guaranteed only if the contents of the redundant register is considered “golden”. To overcome this limitation, it is important to increase the contamination delay of the circuit.

In this chapter, we evaluate the impact of short paths on reliable overclocking, and explore ways to manipulate the contamination delay of the circuit to maximize the performance gains achievable through reliable overclocking. To serve our purpose, we did a case study, to evaluate the performance improvement in Carry Look Ahead (CLA) adders using our technique. CLA adders have very low contamination delay. We need to first increase their contamination delay to get any meaningful improvement in performance by operating them at higher than worst-case speeds. To minimize the area and power overhead associated with adding additional buffers to increase the contamination delay of the circuit, we look at the optimal value for contamination delay, since increasing contamination delay results in higher number of timing errors for the same amount of overclocking performed when the contamination delay is not increased. We further built an experimental setup to estimate the performance improvement of the new CLA adders with increased contamination delay. We observed that for circuits with higher propagation delay, there was a significant performance gain using our technique.
3.1 Impact of Short-paths

In order to support reliable overclocking, we aggressively decrease the clock period, while ensuring that the backup register is timing error free. This is achieved by time delaying the backup register clock signal, $P_{SCLK}$, by the same amount as the difference between the worst-case clock, $W_{CCLK}$, period and the aggressive clock, $MAIN_{CLK}$, period. To guarantee the integrity of data latched by the backup register, it is important to ensure that the input of the backup register changes only because of valid timing error free data. In order to accomplish this, we need to increase the delay of all the paths in the circuit to at least the desired maximum phase shift of the $P_{SCLK}$. To better understand this, let us look at in detail the clock timing requirements for adaptive reliable overclocking. The following discussion assumes dynamic overclocking based on timing error based feedback mechanism. It is possible to consider aggressive operation at one particular frequency. The timing requirements for such an operation can be obtained directly from the equations derived for adaptive reliable overclocking.

3.1.1 Timing Constraints

Figure 3.1 Clock timing waveforms showing governing requirements, for $MAIN_{CLK}$ and $P_{SCLK}$, over the full range of overclocked aggressive frequencies ($F_{MIN}$ $\Rightarrow$ $F_{MAX}$)
To be able to reliably overclock a system dynamically using SPRIT$^3$E framework, the foremost requirement is to generate the $MAIN_{CLK}$ and $PS_{CLK}$. The two clocks are governed by certain timing requirements that are to be met at all times. Figure 3.1 depicts the two clocks, with respect to the $WC_{CLK}$, for the full range of frequencies, $F_{MIN} \Rightarrow F_{MAX}$, that are possible when a system is dynamically overclocked beyond the worst-case operating frequency, $F_{MIN}$. The two clock signals, $MAIN_{CLK}$ and $PS_{CLK}$, have the same frequency at all times, but they are out-of-phase by an amount determined by the extent of overclocking.

We define the following parameters to analyze the clock timing constraints for adaptive reliable overclocking:

- Let $T_{PD}$ denote the worst-case propagation delay of the circuit.
- Let $T_{CD}$ denote the contamination delay of the circuit.
- Let $T_{WCCLK}$, $T_{MAINCLK}$ and $T_{PSCLK}$ represent the clock periods of $WC_{CLK}$, $MAIN_{CLK}$ and $PS_{CLK}$ respectively.
- Let $T_{PS}$ represent the amount of phase-shift between $MAIN_{CLK}$ and $PS_{CLK}$.
- Let $T_{OV}$ denote the overclocked aggressive time period.

At all times, the following equations hold.

\[
T_{WCCLK} = T_{PD} = \frac{1}{F_{MIN}} \tag{3.1}
\]

\[
T_{MAINCLK} = T_{PSCLK} = T_{OV} \tag{3.2}
\]

\[
T_{PD} = T_{OV} + T_{PS} \tag{3.3}
\]

From Figure 3.1, we observe and understand the following:

In $F_{MIN}$ setting, there is no overclocking. $T_{OV} = T_{PD}$. In this case, there is no need to phase shift the $PS_{CLK}$. $T_{PS} = 0$. The two clock signals are identical to the $WC_{CLK}$.
The maximum possible frequency, $F_{\text{MAX}}$ permitted by reliable overclocking is governed by $T_{CD}$. This is because short paths in the circuit, whose delay determine $T_{CD}$, can corrupt the data latched in the backup register. From the $F_{\text{MAX}}$ setting shown in Figure 3.1, we observe the following: Data launched by the $\text{MAIN}_{\text{CLK}}$, at clock edge $A$, is destined to be captured aggressively by the $\text{MAIN}_{\text{CLK}}$ edge $C$ and to be captured timing error free by the $\text{PS}_{\text{CLK}}$ edge $D$. However, if the phase shift $T_{PS}$ is greater than the $T_{CD}$, then the data launched at $A$ can corrupt the backup register at $\text{PS}_{\text{CLK}}$ edge $B$. If such a corruption happens, then the backup register may get incorrect result and cannot be considered “golden”. Hence, it is not possible to overclock further than $F_{\text{MAX}}$. The following equations should hold at all times to guarantee reliable overclocking.

\[
T_{PS} \leq T_{CD} \tag{3.4}
\]

\[
F_{\text{MAX}} \leq \frac{1}{T_{PD} - T_{CD}} \tag{3.5}
\]

For any intermediate overclocked frequency, $F_{INT}$, between $F_{\text{MIN}}$ and $F_{\text{MAX}}$, $T_{PS} \leq T_{CD}$. During operation, $F_{INT}$ is determined dynamically based on the number of timing errors being observed during a specific duration of time.

The dependence of phase shift on contamination delay leads directly to the limitation of the aggressive frequency scaling. A simplistic notion of the maximum speedup that is achievable through reliable overclocking is given by equation 3.6.

\[
\text{Maximum Speedup} = \frac{T_{PD}}{T_{PD} - T_{CD}} \tag{3.6}
\]

3.1.2 Variable or Fixed Phase Shift

Until now, we have been considering a variable phase shift to generate the $\text{PS}_{\text{CLK}}$. In other words, to generate a new frequency of operation, we change both the frequency, as well as the phase shift. However, it is possible to maintain a fixed phase shift between the $\text{MAIN}_{\text{CLK}}$ and the $\text{PS}_{\text{CLK}}$, while generating the aggressive frequencies of operation.
Under fixed phase shift technique, the following changes are required:

- Equation 3.3 changes to $T_{PD} \leq T_{OV} + T_{PS}$.
- For $F_{MIN}$ setting, $T_{OV} = T_{PD}$ and $T_{PS} = T_{CD}$.
- For $F_{MIN}$ setting, $T_{PD} - T_{CD} \leq T_{OV} \leq T_{PD}$ and $T_{PS} \leq T_{CD}$.
- For $F_{MAX}$ setting, $T_{OV} = T_{PD} - T_{CD}$ and $T_{PS} = 0$.

![Figure 3.2 Examples of Main and PS clocks with variable and fixed phase shifts](image)

Figure 3.2 shows three possible ways of generating the two clocks, either with variable phase shift or fixed phase shift, when the worst-case propagation delay is 10ns, and the contamination delay is 3ns. In Case I, there is no frequency scaling, and the clock period of the $MAIN_{CLK}$ is equal to the propagation delay. In Case II, the frequency of the $MAIN_{CLK}$ is scaled to 9ns. Case III shows the maximum possible overclocking. In this case, the clock period of the $MAIN_{CLK}$ is 7ns.

When variable phase shift technique is used, the phase shift of the $PS_{CLK}$ varies from 0 in Case I, to $\Phi_1 = 1ns$ in Case II, and finally to $\Phi_2 = 3ns$ in Case III. In fixed phase shift approach, the phase shift of the $PS_{CLK}$ is fixed at $\Phi = 3ns$ for all three cases.

### 3.1.3 Manipulating Contamination Delay

From the above discussion, we understand that short paths severely limit the extent of reliable overclocking in circuits. This problem is compounded, as most circuits have a significantly lower con-
tamination delay. For instance, a 32-bit CLA adder circuit, implemented in 0.18µm Cadence Generic Standard Cell Library (GSCLib), has a propagation delay of 3.99ns, but an insignificant contamination delay of 0.06ns, thus allowing almost no performance improvement through reliable overclocking.

Since contamination delay limits performance improvement, it might be worthwhile to redesign the logic and increase the contamination delay. Redesigning circuits from ground-up for reliable overclocking is beyond the scope of this thesis. In this thesis, we look at manipulating short paths by increasing their delay to a threshold value determined by the performance requirements.

Increasing the delay of all the paths in the circuit above a desired lower bound, while not affecting the critical path of the circuit is one of the steps performed during synthesis of sequential circuits to fix hold time violations. For a signal to be latched correctly in a flip-flop on the clock edge, the signal should be stable for at least a specific amount of time before and after the clock edge, called the setup time and the hold time respectively. Clock skew, which is the difference in arrival times at the source and destination flip-flops, also exacerbates hold time requirements in sequential circuits. Hold time violations occur when the previous data at the input of the destination flip-flop is not held long enough to be latched properly. The data can change during the hold time window, if the contamination delay of the circuit is less than the hold time requirements at the destination flip-flop. The hold time requirement for a sequential circuit is normally a very small fraction of the propagation delay of the circuit. Hence, adding buffers to short-paths that violate hold time criteria is a step that is done without too much of a concern regarding area and power overheads.

However, increasing the contamination delay of a logic circuit significantly, sometimes as high as half the propagation delay, without affecting its propagation delay is not a trivial issue [84]. At first glance, it might appear that adding delay by inserting buffers to the shortest paths will solve the problem. However, delay of a circuit is strongly input dependent, and several inputs play a role in deciding the value of an output in a particular cycle. Current synthesis tools support increasing the delay of short paths through their hold violation fixing option; in a broader sense, what we essentially want to do is that to extend the hold time of the backup register.

To guarantee the correct working of SPRIT³E framework, the phase shift, $T_{PS}$, cannot be more than half the propagation delay of the circuit. This restriction comes from Equations 3.2 and 3.3. The phase
shift, $T_{PS}$, is at most equal to the overclocked time period, $T_{OV}$. If $T_{PS}$ is greater than $T_{OV}$ and $T_{OV}$ is less than half the propagation delay of the circuit, then Equations 3.2 and 3.3 cannot hold together. The following equation should hold at all times to guarantee reliable overclocking.

$$\frac{T_{PD}}{2} \leq T_{PS} \leq T_{OV} \leq T_{PD}$$  \hspace{1cm} (3.7)

![Figure 3.3 Timing waveforms after increasing contamination delay to half the propagation delay for the full range of overclocked aggressive frequencies ($F_{MIN} \iff F_{MAX}$)](image)

Overclocking to half the original clock period is possible only if the contamination delay is more than half the propagation delay of the circuit. From Equations 3.4 and 3.7, we see that to support reliable overclocking contamination delay of the circuit can be increased up to half the propagation delay of the circuit. Figure 3.3 shows the clock waveforms for $MAIN_{CLK}$ and $PS_{CLK}$ after increasing the contamination delay to its highest beneficial value.

### 3.2 Increasing Contamination Delay of a CLA Adder Circuit - A Case Study

To show that it is possible to increase contamination delay without affecting the propagation delay, we experimented on a CLA adder circuit. Our experiments indicate that by carefully studying the input-output relationship of a given circuit, it is possible to overcome the limitation imposed by contamination
delay on our technique. The following case study presents our experiments and results we achieved for a CLA adder circuit.

Let us first consider an 8-bit CLA adder shown in Figure 3.4. The propagation delay of the circuit is estimated to be 1.06\textit{ns}, and the contamination delay 0.06\textit{ns}. We synthesized the circuit using Cadence BuildGates Synthesis tool in Physically Knowledgeable Synthesis (PKS) mode. We used the 0.18\textit{um} Cadence Generic Standard Cell Library (GSCLib) for timing estimation.

Figure 3.5 shows the delay distribution of all possible timing paths from any input to any output in an 8-bit CLA adder circuit, considering both rising and falling transitions. From Figure 3.5 it can be seen that just about 20\% of the paths have a delay more than 0.75\textit{ns}. Though this is highly motivating and provides a strong reason to apply our technique, a 0.06\textit{ns} contamination delay acts as a dampener and we risk incorrect operation if the clock period is reduced beyond 1\textit{ns}. This is because, once the
input to the adder stabilizes, the output starts changing after 0.06\textit{ns}, the contamination delay of the circuit. This will cause an incorrect sum to be latched by the redundant register at the first rising edge of the $PS_{CLK}$ following the rising edge of the $MAIN_{CLK}$. Reliable execution is guaranteed only if the data latched in the redundant register is considered “golden”.

To overcome the limitation imposed by the contamination delay, we developed a technique to increase the contamination delay without affecting the propagation delay of the circuit. As each output of the CLA adder depends on several inputs, and more than one path to each output exists, with both shorter and longer paths overlapping, adding buffer delays to shorter paths resulted in increasing the overall propagation delay of the circuit. After carefully studying the delay pattern, we observed that it is possible to distribute the additional delay to either the input side or the output side, or both. By doing so, we are able to increase the contamination delay. More importantly, the overall propagation delay remained unchanged.

![Figure 3.6 8-bit CLA adder with additional delay blocks to increase contamination delay](image)

Figure 3.6 shows the new CLA adder circuit. A chain of inverters form the delay block. As seen in the figure, there are now three different circuits that compute the Sum, Propagate, and Generate bits, and they are called L-Type, I-Type, and M-Type. The main difference is where the additional delay is placed. The amount of delay can also be different.

By varying the number of I-Types, and increasing or decreasing the L-Types and M-Types, the increase in contamination delay can be controlled. Each inverter has a delay in the range of 0.06 —
0.08\,ns. For an 8-bit adder, ten inverter delay blocks are sufficient to increase the contamination delay to a significantly higher value. Also increasing contamination delay beyond a certain point is not beneficial, as it will push most of the timing paths to higher delay values, and when frequency is scaled higher more timing errors will occur.

Figure 3.7 shows the new delay distribution. The contamination delay of the circuit now is 0.37\,ns, while the propagation delay remains unchanged at 1.06\,ns. Now 31\% of the timing paths have a delay value greater than 0.75\,ns. Though we can possibly phase shift our $P_{S\,CLK}$ by 0.35\,ns, reducing the clock period by that amount results in higher number of errors. Having a control over the increase in contamination delay gives us an advantage to tune the circuit’s frequency to the optimal value depending on the application and the frequency of occurrence of certain input combinations. Introducing delay to increase contamination delay increases the area of the circuit. Therefore, judiciously increasing contamination delay makes sure that the increase in area is kept minimal.

Using the same technique, to add additional delays, we increased the contamination delay of 32-bit and 64-bit CLA adders.

Table 3.1 provides all relevant details about the implementation of our technique in 8-bit, 32-bit and 64-bit CLA adder circuits. Since enhancing contamination delay without affecting propagation delay was our main goal, our initial implementation did not take into consideration any technique for minimizing the area.
Table 3.1 Implementation details of CLA adder circuits

<table>
<thead>
<tr>
<th>Adder Width</th>
<th>Original</th>
<th>Delay Added</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T_{CD}$ (ns)</td>
<td>$T_{PD}$ (ns)</td>
</tr>
<tr>
<td>8</td>
<td>0.06</td>
<td>1.06</td>
</tr>
<tr>
<td>32</td>
<td>0.06</td>
<td>3.99</td>
</tr>
<tr>
<td>64</td>
<td>0.06</td>
<td>7.89</td>
</tr>
</tbody>
</table>

3.2.1 Analysis of Reliable Overclocking Performance

To estimate the performance improvement achievable using our technique, we performed a series of experiments in ALU circuits. An initial study of a multiplier circuit [8] provided compelling reasons to further explore the possibilities of implementing the technique in circuits with low contamination delay. Also, the study reinforced our conviction that significant performance improvement is possible using overclocking.

![Experimental setup to estimate performance improvement of CLA adder circuits](image)

Figure 3.8 Experimental setup to estimate performance improvement of CLA adder circuits

Not all circuits are as amenable to overclocking as the multiplier circuit. The CLA adder circuit discussed in Section 3.2 provides a good platform to study the various factors that influence overclocking. A different experimental setup, as seen in Figure 3.8, was built to estimate the performance improvement of the adder. This time FPGAs were not used because of the routing delays and the difficulty involved in adding delays to the circuit. As before, two linear feedback shift registers provide random
inputs to the adder. The MAIN$_{CLK}$ and PS$_{CLK}$ have the same frequency. As the frequency of the clocks is scaled higher and higher, the phase shift also increases to provide full propagation delay before the result gets latched in the redundant register. The outputs of the primary and redundant registers are compared every cycle, and the error counter is incremented for each error observed. It is also necessary to verify the correctness of the result stored in the redundant register. The test bench internally computes the sum and that value is compared with that of the redundant register. Another counter is used to log the number of incorrect results computed. Only the adder circuit is simulated with timing information obtained from the Cadence GSCLib. The experiment is then run for 10,000 cycles with different frequencies, and the error rate is observed after every run.

![Figure 3.9](image.png)

**Figure 3.9** Percent of error cycles versus clock period for an 8-bit delay added CLA adder circuit

Figure 3.9 shows the percentage of cycles affected by errors as frequency is scaled higher for an 8-bit CLA adder circuit. The figure also highlights the worst-case clock period, and the clock period at which incorrect results are detected for the first time. The worst-case delay of the circuit is 1.06 ns and the contamination delay is 0.35 ns. When the clock period is 0.8 ns, 33.26% of clock cycles are affected by errors. Until this point there is no incorrect result stored in the redundant register. As we scale further, the phase shift exceeds the contamination delay. The number of incorrect results latched in the redundant register is almost 98% when the clock period is 0.7 ns. Frequency scaling beyond this point will result in unreliable execution.
Figure 3.10 Percent of error cycles versus clock period for a 32-bit delay added CLA adder circuit (Contamination delay 1.21ns)

Figure 3.11 Percent of error cycles versus clock period for a 32-bit delay added CLA adder circuit (Contamination delay 1.38ns)
Figure 3.10 and Figure 3.11 present the error rate at various frequencies for 32-bit CLA adder circuits with different contamination delay values. From the figures, we observe that increasing contamination delay beyond a point, results in higher error rate. As contamination delay is increased from 1.21\(ns\) to 1.38\(ns\), the number of cycles affected by errors at 3\(ns\) clock period goes from 10.07\% to 55.19\%. This illustrates the need to optimally increase contamination delay.

![Figure 3.12 Percent of error cycles versus clock period for a 64-bit delay added CLA adder circuit](image)

Finally, Figure 3.12 shows the percentage of errors detected in a 64-bit CLA adder operating at higher than worst-case speeds.

From the different adder experiments, we observe that as propagation delay of a circuit goes up, the clock period can be reduced further and further. In the case of 64-bit CLA adder, even after the phase shift is increased beyond the contamination delay (1.82\(ns\)) there is no incorrect sum detected. Only when the clock period is reduced to 5\(ns\), incorrect results are observed. The reason for this is that the shorter paths responsible for the contamination delay are not being exercised by the inputs to the adder. However, increasing the phase shift beyond the contamination delay does not guarantee reliable execution for all possible inputs.
CHAPTER 4. CHARACTERIZING ADAPTIVE RELIABLE OVERCLOCKING

In this chapter, we explore timing speculation based reliable overclocking and evaluate the various factors that impact reliable overclocking. We present a design methodology that provides dynamically controllable knobs to designers to balance the parameters of interest.

Figure 4.1 Alpha 21264 integer and floating point pipeline showing timing error detection and recovery circuit for critical registers

In a reliably overclocked processor (ROP), to tolerate timing errors, registers in the critical paths of every pipeline stage are augmented with a second time-delayed register. Figure 4.1 shows an enhanced Alpha 21264 pipeline that is capable of supporting speculative reliable overclocking. A typical pipeline stage in such a processor, along with local timing error detection and recovery circuit augmentation for critical path registers, is shown in the figure. Reliably overclocking a processor may not yield an increase in performance at all times; the reason being that the occurrence of a timing error is highly dependent on the workload and the current operating conditions. The amount of frequency scaling is
strongly influenced by the number of input combinations responsible for the longer timing-paths. As frequency is scaled higher dynamically, more number of input combinations would result in error. Each time an error occurs, additional time is required to recover from that error.

In this chapter, we analyze target error rate values and evaluate adaptive clock tuning systems, which increase or decrease the clock frequency based on a set target error rate. We monitor the error rate during run time, using various error sampling techniques and based on a set tolerable error rate that does not affect the performance, we adjust the clock frequency dynamically. The fact that dynamic clock tuning comes at a significant runtime cost has lead to the use of two clock generators enabling clock tuning in near zero time. In this chapter, we analyze the benefits of using single and dual clock generators, and various clock tuning techniques. We further evaluate the importance of a faster memory alongside a faster processor. We present our results for SPEC2000 integer and floating point benchmarks executing on a SimpleScalar Alpha processor, augmented with error rate profiles obtained from a Alpha processor hardware model.

### 4.1 Evaluating Speculative Reliable Overclocking

In Chapter 3, we determined the limits of the frequency scaling, and the importance of manipulating short paths to maximize gains. Let us now look at the other important factor that affects reliable overclocking. The number of errors that occur at overclocked frequencies play a significant role in determining the extent of overclocking. As frequency is scaled higher, the number of input combinations that result in delays greater than the new clock period also increases. Each error takes additional cycles to recover, resulting in diminishing returns at higher operating frequencies. Hence, deciding on the target error rate is the foremost step in speculative reliable overclocking.

Let us first analyze the impact of error rate:

Let $T_{WC}$ denote the original worst-case clock period.

Let $T_{OV}$ denote the clock period after aggressive frequency scaling.

Let $T_{GAIN}$ be the time difference between the original clock period and the aggressive clock period.

\[
T_{GAIN} = T_{WC} - T_{OV}
\]  (4.1)
Let $\delta$ denote the frequency scaling factor.

$$\delta = \frac{T_{WC}}{T_{OV}} \quad (4.2)$$

If a particular application takes $C_{TOT}$ clock cycles to execute, then the total execution time is reduced by $T_{GAIN} \times C_{TOT}$, if there is no error.

Let $C_{REC}$ be the number of cycles needed to recover from an error.

Let $\varepsilon$ denote the fraction of clock cycles, out of a total $C_{TOT}$ cycles, affected by errors, due to overclocking.

To achieve any performance improvement Equation 4.3 must be satisfied. Equation 4.3 states that as long as the error recovery overhead is less than the reduction in execution time, a reliably overclocked system performs better than a non-overclocked system.

$$\varepsilon \times C_{TOT} \times C_{REC} \times T_{OV} < C_{TOT} \times T_{GAIN} \quad (4.3)$$

Equation 4.4 provides an upper bound for the number of errors that can be tolerated. This upper bound is inversely proportional to the error recovery penalty. Equation 4.4 is for a system that is always overclocked at a constant frequency.

$$\varepsilon < \frac{T_{GAIN}}{C_{REC} \times T_{OV}} \quad (4.4)$$

For adaptive systems, in addition to the timing error recovery overhead that scales along with frequency scaling, there is also a frequency switching time, $T_{TUNE}$, consumed by the clock generator’s frequency synthesizer to generate the new frequency. Therefore, it is necessary to make sure that the time gained through overclocking is more than the total losses incurred. Considering the frequency switching penalty, Equation 4.4 becomes Equation 4.5. Here, $C_{SAMP}$ refers to the number of cycles in an error sampling interval before frequency switching is triggered.

$$\varepsilon < \frac{T_{GAIN}}{C_{REC} \times T_{OV}} - \frac{T_{TUNE}}{C_{SAMP} \times C_{REC} \times T_{OV}} \quad (4.5)$$
An application can now be partitioned into smaller time intervals, and the frequency for every subsequent interval be determined by the number of errors being observed at the current overclocked frequency. In order to maximize gains, $T_{TUNE}$ should be minimized or the error sampling interval should be made much longer. If we consider an application that takes $C_{TOT}$ clock cycles to execute, and partition it into $n$ sampling intervals, each consisting of $C_{SAMP}$ cycles, then Equation 4.6 presents the overall aggressive execution time, $EX_{OV}$. Here, each sampling interval has a different clock period and error rate. If we assume that the system starts running initially at worst-case clock period, then $T_{OV1} = T_{WC}$. Each sampling interval includes a frequency switching time.

$$EX_{OV} = (C_{SAMP} + \epsilon_1 \times C_{SAMP} \times C_{REC}) \times T_{OV1} + \ldots + (C_{SAMP} + \epsilon_n \times C_{SAMP} \times C_{REC}) \times T_{OVn} + n \times T_{TUNE}$$  \hspace{1cm} (4.6)$$

The total execution time for a non-overclocked system that executes an application from start to finish at the worst-case clock period is given by Equation 4.7. The term $T_{TUNE}$ is included to account for the one-time clock generation time.

$$EX_{WC} = C_{TOT} \times T_{WC} + T_{TUNE}$$  \hspace{1cm} (4.7)$$

The execution cycles for a pipelined processor are mainly divided into cycles for instruction execution, memory and branches. During overclocking, the number of execution cycles may go up depending on timing errors, as already discussed. Also, in a computation, it is possible that when the clock frequency is scaled there is an increase in the total number of execution cycles.

For instance, in a pipelined processor, when the processor accesses memory, the number of clock cycles taken for that memory operation increases when the frequency is scaled, if the clock frequency of the memory remains constant. Consider a processor whose clock period is 10ns, and a memory access which took 20 CPU cycles. If after scaling, the clock period is reduced to 5ns, then the same memory access would take 40 CPU cycles. Now, if the clock frequency of the memory is not scaled corresponding to that of CPU, there will be an increase in the memory cycles. However, the branch penalty cycles remain unaffected.
Analytically, if each memory operation takes $C_M$ cycles at $T_{WC}$, then by scaling the clock by a factor of $\delta$ each memory operation will now take $\delta \times C_M$ cycles. However, this is strongly dependent on the number of memory bound instructions.

Let $\gamma$ denote the fraction of memory access per cycle.

If we consider that $C_{TOT}$ excludes memory cycles, then the aggressive execution time for any particular sampling interval is given by Equation 4.8. Equation 4.8 accounts for frequency switching penalty for that sampling interval. Equation 4.9 presents the overall execution time.

$$EX_{OV\ INT} = C_{SAMP} \times T_{OV} + \gamma \times C_{SAMP} \times \delta \times C_M \times T_{OV} + \varepsilon \times C_{SAMP} \times C_{REC} \times T_{OV} + T_{TUNE}$$ (4.8)

$$EX_{OV} = EX_{OV\ INT1} + EX_{OV\ INT2} + \ldots + EX_{OV\ INTn}$$ (4.9)

By replacing $T_{OV}$ by $T_{WC}$ and substituting $\delta = 1 \& \varepsilon = 0$ in Equation 4.8, we get the worst-case runtime for a sampling interval, as given by Equation 4.10. By replacing $C_{SAMP}$ by $C_{TOT}$, we get the overall worst-case execution time, $EX_{WC}$. The one-time clock generation overhead can be added to $EX_{WC}$ estimation.

$$EX_{WC\ INT} = C_{SAMP} \times T_{OV} + \gamma \times C_{SAMP} \times C_M \times T_{OV}$$ (4.10)

Although overclocking improves performance, it also increases the switching activity of the circuits. This causes more dynamic power dissipation. As Equation 4.11 illustrates, we see a factor of $\delta$ increase in the dynamic power consumed because of overclocking. Here, $\alpha, C$ and $V$ are switching activity factor, circuit capacitance and system voltage respectively. Leakage power is discussed in Section 4.2.2.

$$P_{OV} = \frac{\alpha \times C \times V^2}{T_{OV}} + P_{LEAK} = \frac{\alpha \times C \times V^2}{\delta \times T_{WC}} + P_{LEAK}$$ (4.11)
4.1.1 Performance Metrics

In order to evaluate the performance of speculative reliable overclocking, we derive the following performance metrics:

The speed up achieved from reliable overclocking in a sampling interval is obtained by dividing $EX_{OVI T}$ by $EX_{WC INT}$. This is given by Equation 4.12. The overall speedup is given by Equation 4.13.

\[
Speedup = \frac{EX_{WC INT}}{EX_{OVI T}} = \frac{\delta \times (1 + \gamma \times C_M)}{(1 + \gamma \times \delta \times C_M + \varepsilon \times C_{REC}) + T_{TUNE}} \tag{4.12}
\]

\[
OverallSpeedup = \frac{EX_{WC}}{EX_{OV}} = \frac{EX_{WC}}{EX_{OVI T1} + EX_{OVI T2} + \ldots + EX_{OVI Tn}} \tag{4.13}
\]

Normally, for pipeline stage level timing speculation designs, the recovery is made the following cycle, i.e., $C_{REC} = 1$. From Equation 4.12, we can understand that for optimal performance enhancement, the workload should have a right mixture of memory bound and CPU bound instructions. Practically, $\gamma$ is quite small, as there exists several methods in literature for shadowing the memory operations, such as caching and buffering.

Traditionally, power and delay have been the two most important specifications for digital systems and microprocessor designs. There are numerous metrics in use presently, all of which are primarily based on these two factors. Each metric assigns different weights to the two terms based on the design goals. Power-Delay Product (PDP), or the energy is a widely used metric for older technologies (> 180nm). With increase in leakage power in the deep submicron era, PDP may not prove to be the best option. Also, PDP is technology dependent; necessitating newer metric for comparing across different technologies. An improved metric called the Energy-Delay Product (EDP) was later developed. This gives a quadratic weightage for delay compared to energy. In other words, EDP measures how fast a given circuit operates, while consuming minimal energy.

The energy or PDP for an overclocked system is given by Equation 4.14. EDP for an overclocked system is calculated from Equation 4.15. By replacing $T_{OV}$ with $T_{WC}$ and $EX_{OV}$ with $EX_{WC}$ in Equations 4.11, 4.14, 4.15 the corresponding metrics for a non-overclocked system are calculated.
\[ PDP_{OV} = \frac{\alpha \times C \times V^2}{T_{OV}} \times EX_{OV} + P_{LEAK} \times EX_{OV} = P_{OV} \times EX_{OV} \quad (4.14) \]

\[ EDP_{OV} = \frac{\alpha \times C \times V^2}{T_{OV}} \times EX_{OV}^2 + P_{LEAK} \times EX_{OV}^2 = PDP_{OV} \times EX_{OV} \quad (4.15) \]

### 4.2 Analysis Framework

To evaluate the effectiveness of speculative reliable overclocking, we develop an analysis framework that allows us to analyze the impact of various factors on the extent of reliable overclocking. We modify an existing model of a microprocessor, add the necessary features that help us to study the benefits of speculative reliable overclocking and provide an effective framework for understanding the performance gains.

Figure 4.2 presents the entire analysis methodology. The figure depicts a SimpleScalar Alpha processor simulator, in combination with a Wattch power model and HotSpot thermal model. The individual components are explained below in detail. The power model takes in technology specific parameters to compute active and leakage power every cycle. The thermal model provides block level temperature based on chip floor plan and run-time power consumption information. A hardware model of an Alpha processor implemented in 45nm technology is used to enable overclocking dependent error injection. During normal execution, without any overclocking, clock controller provides a singular technology dependent base frequency. When reliable overclocking is enabled, timing error based feedback control is activated. Adaptive clock tuning techniques are employed to adapt system behavior based on workload characteristics.

Our base processor, which is an out-of-order 64-bit Alpha EV6 processor, is derived from the SimpleScalar-Alpha tool set [17]. This processor executes the Alpha AXP ISA. Our base processor configuration resembles that of Alpha EV6 processor. Table 4.1 provides the configuration details of the principal features of the base processor. The processor configuration remains consistent with technology scaling, and we evaluate the same base processor across different technologies. However, the area estimate varies, and this is captured by the floor plan, which is provided as an input to the thermal model.
4.2.1 Modeling a Reliably Overclocked Processor (ROP)

To evaluate the capabilities of speculative reliable overclocking, we modeled a reliably overclocked processor using a functional simulator, which incorporates a random timing error injector based on error profiles obtained by running application binary on a hardware model. When reliably overclocked, we dynamically tune the clock frequency based on the number of errors happening during a pre-determined time interval and target error rate.

In order to bring in the aspects of timing error in the SimpleScalar Alpha simulator, which is cycle accurate, but not timing accurate, we analyzed the number of timing errors occurring in the hardware model of a superscalar processor. For this purpose, we analyzed the error rate in the different pipeline stages of a superscalar, dynamically scheduled integer pipeline similar in complexity to the Alpha 21264 [101] that executes a subset of the Alpha instruction set. Our analysis was performed at each stage in isolation because the hardware model of the processor is not fully synthesizable, and we require a synthesizable model to get the gate-level timing information. As a result, we synthesized individual pipeline stages using the 45nm OSU standard cell library [91]. Once we got the synthesized blocks for a particular stage, we replaced the RTL model for that block with the synthesized model, and evaluated each pipeline stage independently. We annotated the timing information, extracted in standard delay format (SDF), on the blocks to run timing accurate simulations. We ran the instruction
<table>
<thead>
<tr>
<th>Feature</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch/decode/issue/commit width</td>
<td>4/4/4/4 instructions/cycle out of order execution</td>
</tr>
<tr>
<td>Functional units</td>
<td>4 integer arithmetic and logic units</td>
</tr>
<tr>
<td></td>
<td>1 integer multiplier/divider unit</td>
</tr>
<tr>
<td></td>
<td>4 floating point arithmetic and logic units</td>
</tr>
<tr>
<td></td>
<td>1 floating point multiplier/divider unit</td>
</tr>
<tr>
<td>Branch predictor</td>
<td>8-K entry bimodal</td>
</tr>
<tr>
<td>Branch target buffer</td>
<td>512 sets, four-way set associativity</td>
</tr>
<tr>
<td>L1 instruction cache size</td>
<td>64 KB</td>
</tr>
<tr>
<td>L1 data cache size</td>
<td>64 KB</td>
</tr>
<tr>
<td>L2 unified cache size</td>
<td>2 MB</td>
</tr>
</tbody>
</table>

Table 4.1 Processor specifications

profile of various integer benchmarks obtained from the SimpleScalar simulator through the various stages. We used random data values for other inputs, filled the memory with random data. We explain below in detail, how we obtained the integer pipeline error profile. For measuring error rate in floating point computation, we evaluated a floating point ALU obtained from opencores.org, and measured the error rate at varying frequencies and incorporated it as part of the floating point error profile shown in Figure 4.2. For the rest of the floating point pipeline, we used the average of integer pipeline error profiles.

<table>
<thead>
<tr>
<th>Pipeline Stage</th>
<th>$T_{PD}$ (ns)</th>
<th>$T_{CD}$ (ns)</th>
<th>% Critical Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>3.90</td>
<td>0.06</td>
<td>2.1</td>
</tr>
<tr>
<td>Decode</td>
<td>2.76</td>
<td>0.10</td>
<td>0</td>
</tr>
<tr>
<td>Rename</td>
<td>2.88</td>
<td>0.06</td>
<td>0</td>
</tr>
<tr>
<td>Issue</td>
<td>4.89</td>
<td>0.10</td>
<td>89.17</td>
</tr>
<tr>
<td>Execute</td>
<td>6.65</td>
<td>0.08</td>
<td>11.86</td>
</tr>
<tr>
<td>Memory</td>
<td>5.21</td>
<td>0.10</td>
<td>3.21</td>
</tr>
<tr>
<td>Commit</td>
<td>1.94</td>
<td>0.07</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.2 Synthesis report of major pipeline stages

The hardware model of the processor has altogether 12 pipeline stages. Table 4.2 reports the synthesis results for the major pipeline stages. In the hardware model, the fetch stage, for instance, is divided into three stages. The table reports only the propagation delay, $T_{PD}$, and the contamination delay, $T_{CD}$, for the slowest among the three fetch stages. The timing values, reported in $ns$, are obtained
from static timing analysis reports. In the table, we report the percentage of registers that have path terminating at them with delay values greater than or equal to 3.5\(\text{ns}\). We fixed the worst-case delay at 7\(\text{ns}\) to allow the maximum propagation delay of 6.65\(\text{ns}\) in the execute stage.

Figure 4.3 shows the error rate for the issue stage, execute stage and the cumulative error rate for all stages of the processor. We noticed around 89.17\% of the paths fail in the issue stage at 3.5\(\text{ns}\), which causes a sudden rise in error rate, as observed in the Figure 4.3. From the distribution of critical registers and as correlated by the error rate results for various benchmarks, we observe that the primary contributors for timing errors are the issue stage and the execute stage. Since, the issue stage performs computation every cycle, overclocking it results in significant increase in errors, as shown in Figure 4.3. However, the execute stage performs computations that exercise the critical paths rarely resulting in moderate increase in error rate with overclocking.

The error rate at different frequencies were measured by running the experiment for 100,000 cycles, and repeating the experiment with different sequences of 100,000 instructions for each benchmark. Average values are reported in the chart. Based on the error profile, the random timing error injector in Figure 4.2 injects appropriate number of errors. Whenever an error occurs, the pipeline stalls for a cycle.

As explained earlier, it is necessary to augment critical registers with error detection and recovery circuit, and also increase contamination delay of paths terminating at critical registers to a value greater than the desired extent of overclocking. Our simulator overclocks up to 55\% of the worst-case clock period. This requires increasing contamination delay to over 45\% the clock period. We obtain the power overhead values from Razor [27] and incorporate them in our power and thermal model. In Razor, the contamination delay was increased to 50\% of the clock period for paths terminating in critical registers. The power overhead reported was 3\%, which came from the extra buffers padded to improve contamination delay and the backup registers.

Figure 4.4 shows the error profiles for three benchmarks executing five different instruction and data sets. The variability seen in these plots indicate the need for adaptive clocking techniques. The performance of a ROP is significantly enhanced by a dynamic overclocking mechanism. Traditional dynamic frequency scaling techniques stall the system clock during the frequency switching phase.
Figure 4.3 Cumulative error profile for all pipeline stages at overclocked operating frequencies for SPEC2000 integer benchmarks. Also shown separately are error profiles for issue stage and execute stage.
Figure 4.4 Error profile for three SPEC2000 integer benchmarks executing five different instruction and data sets
The entire process of switching from one frequency to another may take upwards of 100 clock cycles depending upon the speed at which the voltage controlled oscillator (VCO), and the delay lock loops (DLLs) or phase-locked loops (PLLs) can generate the new stable clock signal. This frequency switching penalty becomes a bottleneck, and prevents adjusting the clock frequency frequently. To overcome this limitation, we may consider using two clock generators, and have a control mechanism that switches between these two clock generators. This provides the capability to adjust concurrently the clock frequency, while the system is running. Adaptive clocking mechanisms are discussed in detail in Section 4.3.

Table 4.3 specifies the various simulation parameters we incorporated into our ROP simulation model.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology node</td>
<td>45nm</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>1V</td>
</tr>
<tr>
<td>Threshold Voltage</td>
<td>0.2398V</td>
</tr>
<tr>
<td>Worst-case frequency</td>
<td>1536MHz</td>
</tr>
<tr>
<td>Maximum Overclocked frequency</td>
<td>2792MHz</td>
</tr>
<tr>
<td>No. of frequency levels</td>
<td>32</td>
</tr>
<tr>
<td>Frequency sampling interval</td>
<td>100000 cycles</td>
</tr>
<tr>
<td>PLL locking time</td>
<td>10μs</td>
</tr>
<tr>
<td>Frequency switching penalty</td>
<td>Single PLL: 10μs</td>
</tr>
<tr>
<td></td>
<td>Dual PLL: 0μs</td>
</tr>
<tr>
<td>Temperature sampling interval</td>
<td>1ms</td>
</tr>
</tbody>
</table>

Table 4.3 Simulator parameters

4.2.2 Power and Thermal Modeling

Wattch [15] is an accurate, architecture level power tool that is embedded within the SimpleScalar simulator. Wattch categorizes the various processor units to be one among the following four types: array structures, fully-associative content-addressable memory structures, combinational logic blocks and clock resources. This classification enables modeling the power for each processor functional block based on its categorization, input configuration that determines the size of the block, access pattern, which is workload dependent and implementation technology parameters. Wattch estimates
the worst-case cycle power based on voltage, frequency and process technology. During runtime, this worst-case power consumption value is scaled based on accesses and resources used in a cycle. Wattch calculates instantaneous power that includes both active and leakage power at every cycle, and outputs the total power accumulated over a simulated period of time.

We chose the in built linear clock gating with 10% turnoff power, as this resembles the industry reported leakage power values. Even as leakage power tends to get worse with scaling, several new techniques have come up to minimize leakage power within acceptable levels. Some of the popular leakage reduction techniques include architecture level techniques, such as power gating, transistor level techniques, such as multi-threshold transistors, and material changes, such as the latest hafnium base high-$k$ dielectric for gate oxide. One important fact that makes leakage power so important with scaling is its exponential dependence on on-chip temperature. Active power, however, is largely independent of temperature, even as circuit switching is temperature dependent because of the effect of temperature on the threshold voltage of a CMOS device.

We model this temperature dependence of leakage power, using the empirical relationship presented in Equation 4.16 [40]. Here, $\beta$ is technology dependent constant ($\beta$ is 0.036 and 0.017 for 180nm and 70nm respectively), $T_0$ is the temperature of a reference point and $T_i$ is the temperature at $i^{th}$ instant with respect to the reference point. We obtain on-chip temperature values for each of the functional blocks using the HotSpot tool [42], which acts a thermal sensor.

$$P_{\text{LEAK}} \propto e^{\beta(T_i - T_0)} \tag{4.16}$$

We modified Wattch to track instantaneous power for each functional block. The instantaneous power trace is provided to the HotSpot RC model to calculate temperature. Thermal modeling requires the chip floor plan. We obtain the Alpha EV6 floor plan for 45nm technology node from 130nm floor plan provided as part of the HotSpot distribution by assuming scaling is proportional to square of technology. The thermal model, which was originally designed for 0.18um technology, includes die, heat spreader, and heat sink models.
4.3 Adaptive Clocking

Dynamic clock frequency tuning is controlled by a global feedback system, which is based on the total number of timing errors that occur in a specified time interval. The number of errors occurring at each timing error counter sampling interval is continuously monitored. As long as the number of errors is within the target limits, the frequency is scaled up, else scaled down. One can apparently construe that the error rate is a monotonically increasing function with respect to frequency. This allows the use of efficient search algorithms to select the next tuned frequency, starting from the base frequency. The maximum frequency for performance enhancement is theoretically limited by the contamination delay of the circuit. If the time period of the new frequency is less than the contamination delay of the circuit, timing errors certainly occur during every cycle and the error rate goes to 100%. Clock periods that are greater than the propagation delay do not cause any timing errors (0% error rate). Earlier studies have indicated that fixing a non-zero target error rate improves performance significantly.

The dynamically tuned frequency is achieved through the global feedback system pictured in Figures 4.5 and 4.6. Figure 4.5 depicts a feedback mechanism with single clock generator and variable phase shift option for the $PS_{CLK}$. Figure 4.6 shows the presence of two clock generators and the $PS_{CLK}$ has a fixed phase shift from the $MAIN_{CLK}$. Both variable and fixed phase shift options can be used either with single or dual clock generators.
Figure 4.5 shows the ROP, which is SPRIT\textsuperscript{3E} in this case, along with the dynamic clock tuning mechanism, wherein a single programmable clock generator is used. Before operation begins, a small, non-zero, error rate is programmed as the set point. The clock controller is initialized with the worst-case delay parameters of the pipeline. The initial frequency of the clocks is determined by the worst-case propagation delay, and the $PS_{CLK}$ begins with no phase shift ($\Delta \Phi = 0$). These values are sent to the clock generator block. This block consists of the clock generator, which includes a voltage controlled oscillator (VCO) and a PLL. The VCO is able to generate a variable frequency clock that meets the $\Delta F$ value given by the clock controller. The PLL locks the output of the VCO to provide the $MAIN_{CLK}$ to the pipeline. The clock generator then phase shifts the $MAIN_{CLK}$ by the value requested by the clock controller and produces the $PS_{CLK}$. Once the clocks are stable, the clock generator states that the signals are locked. During the period in which the clocks are being adjusted, the pipeline is stalled. To avoid a high overhead from frequent clock switching, the number of timing errors in the pipeline must be sampled at a large interval and a new frequency determined after that interval.

Figure 4.6 depicts a clock tuner, which is composed of a clock controller, two programmable clock generators, and a phase shift control block. In the figure, the parameters that control the operation of each of the blocks is highlighted. The operation of this clock tuner is as follows:

![Feedback control system to dynamically tune clock frequency: Dual clock generators with fixed phase shift](image)

Figure 4.6  Feedback control system to dynamically tune clock frequency: Dual clock generators with fixed phase shift

The programmable clock generators generate clocks that have frequencies within the range prescribed by $F_{MIN}$ and $F_{MAX}$. The exact frequency to be generated is based on the frequency tuning value
that is estimated by the clock controller based on the current error rate. The frequency steps that are possible in the given frequency range is determined by the number of bits used to represent the two tuning values $Tuner_{Gen1}$ and $Tuner_{Gen2}$. For instance, if we use 5-bits, then 32 different frequencies can be generated between $F_{MIN}$ and $F_{MAX}$, with 0 corresponding to $F_{MIN}$ and 31 corresponding to $F_{MAX}$.

If the number of steps, $N_{Steps}$, is fixed, then the step increase or decrease in clock time period is given by Equation 4.17.

$$T_{Step} = \frac{T_{MAX} - T_{MIN}}{N_{Steps}}; \quad T_{MAX} = \frac{1}{F_{MIN}} \quad \text{and} \quad T_{MIN} = \frac{1}{F_{MAX}}$$  \hspace{1cm} (4.17)

Based on the values of $Tuner_{Gen1}$ and $Tuner_{Gen2}$, $T_{PS}$ for each generator is calculated as $T_{Step} \times Tuner_{Gen1}$ and $T_{Step} \times Tuner_{Gen2}$, respectively. Once $T_{PS}$ is known, using Equation 3.3, the clock frequencies of $Clk1$ and $Clk2$ are decided.

The clock select block switches between the clocks generated by the two generators. The clock controller generates the signal $ClkSwitch$ to select between the two clocks. Generator selection toggles whenever a new frequency is generated, and is in sync with the newest frequency being generated. The actual switching happens after allowing sufficient time for the outputs of the clock generators to stabilize. Once the $MAINCLK$ is selected, the fixed phase shift block phase shifts the $MAINCLK$ to generate the $PSCLK$. The clocks to the ROP are stalled for few cycles to ensure that the system functionality is not affected by unwanted glitches in any of the two clocks.

### 4.3.1 Clock Tuning Schemes

The operation of the clock controller is dependent on two parameters: $TargetErrRate$, which specifies the tolerable error rate, and $SwitchRate$, which specifies the rate at which frequency switching happens. For example, if we specify $SwitchRate$ as 100,000, then frequency switching happens every 100,000 clock cycles. If $TargetErrRate$ is specified as 1% (an error rate of 1000 per 100,000 cycles), then frequency is increased if the number of errors is below 1000 after 100,000 cycles, otherwise it is decreased. Frequency is decreased immediately if the error rate exceeds the $TargetErrRate$ anytime during the 100,000 cycles. Also, a tolerance band for the error rate can be used, during which the frequency will not be changed. As mentioned earlier, frequency switching takes several cycles once
Algorithm 1  Binary Search Algorithm for Calculating Tuning Values

1: Initial: $T_i = T_j = T_{hi} = T_{hj} = 0$; $j = 1$; $i = 2$
2: Initial: $\text{LowerBound} = 0$; $\text{UpperBound} = 2^{\text{Tuner} \_ \text{Gen} \_ \text{length}} - 1$
3: if cycles $\geq \text{SwitchRate}$ then
4:   $x \leftarrow \text{MAX}(T_j, T_{hj}, T_{hi})$
5:   $y \leftarrow \text{MIN}(T_j, T_{hj}, T_{hi})$
6:   $T_i \leftarrow T_i$
7:   if $\text{ErrRate} \leq \text{TargetErrRate}$ then
8:      if $T_i \geq x$ then
9:         $T_j \leftarrow \frac{T_i + \text{UB}}{2}$
10:    else
11:       if $T_i < y$ then
12:          $T_j \leftarrow \frac{T_i + y}{2}$
13:       else
14:          $T_j \leftarrow \frac{T_i + x}{2}$
15:     end if
16:   end if
17: else
18:      if $T_i \leq y$ then
19:         $T_j \leftarrow \frac{T_i + \text{LB}}{2}$
20:     else
21:       if $T_i > x$ then
22:          $T_j \leftarrow \frac{T_i + x}{2}$
23:       else
24:          $T_j \leftarrow \frac{T_i + y}{2}$
25:     end if
26: end if
27: $T_{hi} \leftarrow \text{Tuner} \_ \text{Gen}_i$; $T_{hj} \leftarrow \text{Tuner} \_ \text{Gen}_j$
28: $\text{Tuner} \_ \text{Gen}_i \leftarrow T_i$; $\text{Tuner} \_ \text{Gen}_j \leftarrow T_j$
29: $i \leftarrow j$; $j \leftarrow i$
30: cycles $\leftarrow 0$
31: else
32:   cycles $\leftarrow$ cycles + 1
33: end if
initiated. The ERateClr signal is asserted for one cycle just before asserting ClkSwitch to reset the system error rate.

Frequency tuning is implemented in an ad-hoc fashion. Frequency is increased or decreased by a single step based on the history of error during the preceding sampling interval. A binary search can be implemented too. Simple binary search does not work well, as frequency switches by a large amount for every switch and does not settle down at the right value. We use a modified binary search algorithm for deciding on the tuning values. Algorithm 1 runs once to calculate the new tuning values whenever frequency switching is called for. Our search alternates between TunerGen1 and TunerGen2. We also record the current tuning value before changing it, so that whenever it is required to find the midpoint between the current value and the lower bound or the upper bound, our search algorithm first looks at the last three tuning values to make a decision. This allows the dynamic clock tuner to change frequencies by smaller values in the beginning, and if error rate is still high or low, drop to the midpoint based on upper or lower bound, as the case may be.

4.3.2 Comparing Adaptive Clocking Techniques

In this section, we compare the benefits of having a single clock generator or two clock generators to support reliable overclocking. We compare the performance of the two clocking schemes, shown in Figures 4.5 and 4.6, against a base system that does not support any overclocking. The base processor operates at the worst-case clock frequency, and does not see any timing errors during its operation. The base system theoretically has an error rate target of 0% and does not have a need to adapt its frequency during runtime. The two overclocked systems adapt themselves during runtime based on the specified target error rate. The single clock generator system incurs a clock switching penalty of around 100µs, since the processor has to be stalled when the system frequency is changing. The dual clock generator system tunes its frequency using the second generator, while the first generator provides the clock signal to the processor. When the need arises the processor alternates between the two generators and the switching happens within a few cycles.

Table 4.4 compares various performance metrics between a base non-overclocked processor, a reliably overclocked processor that uses a single clock generator for dynamic clock tuning and a reliably
overclocked processor that uses two clock generators to adapt its frequency quickly to the changing runtime requirements. The comparison results are presented for three error rate targets. We report results for six integer benchmarks, namely, bzip2, crafty, gap, gzip, mcf, and vpr. From the results, we see that there are significant runtime benefits, as a result of reliable overclocking. The dual clock generator system achieves the best runtime across all benchmarks. However, the power consumption is much higher for the dual clock generator system. Still, the dual clock generator system is beneficial if we look at the energy-delay product.

For an error rate target of 1%, the single clock generator system achieves, on an average, 11.34% improvement in runtime over the base system. The dual clock generator system outperforms the single clock generator system by almost 20%, and betters the base system by almost 30%, across all benchmarks. Even for 3% and 5% error rate targets the dual clock generator is 20% faster than the single clock generator system, even as the single clock generator outperforms the base system by about 20%.

With respect to power, the base system consumes the lowest power, followed by the single and dual clock generator based overclocked systems. If we consider the energy-delay product metric, the dual clock generator based overclocked system turns out to be much better as it achieves around 30% improvement over the base system, and outperforms the single clock generator based overclocked system by about 25%, across all benchmarks.

Table 4.5 presents results for six SPEC2000 floating point benchmarks. The benchmarks we considered are applu, apsi, equake, galgel, mesa, and mgrid. For the 1% error rate target, the single clock generator based overclocked system achieves, on an average, only about 4% improvement in runtime, while the dual clock generator based overclocked system achieves almost 24%. For 3% and 5% error rate targets the system with two clock generators does better by 20% than the system with single clock generator. The energy-delay product values still indicate that having a dual clock generator based tuning system for a reliably overclocked processor is a good idea, and we can achieve more benefits by doing so.
Table 4.4 Comparing various performance metrics between a base non-overclocked processor, a reliably overclocked processor tuned using a single clock generator and a reliably overclocked processor tuned using dual clock generators. All the systems execute SPEC2000 integer benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>Target Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Base</td>
</tr>
<tr>
<td>bzip2</td>
<td>RUN TIME (ms)</td>
<td>40.08</td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>2.55</td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>4.09</td>
</tr>
<tr>
<td>crafty</td>
<td>RUN TIME (ms)</td>
<td>28.30</td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>3.48</td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.10</td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>2.79</td>
</tr>
<tr>
<td>gap</td>
<td>RUN TIME (ms)</td>
<td>20.04</td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>4.37</td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.09</td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>1.75</td>
</tr>
<tr>
<td>gzip</td>
<td>RUN TIME (ms)</td>
<td>52.25</td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>2.75</td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>7.52</td>
</tr>
<tr>
<td>mcf</td>
<td>RUN TIME (ms)</td>
<td>26.03</td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>5.36</td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>3.63</td>
</tr>
<tr>
<td>vpr</td>
<td>RUN TIME (ms)</td>
<td>19.56</td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>3.41</td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>1.31</td>
</tr>
</tbody>
</table>
Table 4.5 Comparing various performance metrics between a base non-overclocked processor, a reliably overclocked processor tuned using a single clock generator and a reliably overclocked processor tuned using dual clock generators. All the systems execute SPEC2000 floating point benchmarks.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Metric</th>
<th>BASE</th>
<th>SINGLE</th>
<th>DUAL</th>
<th>SINGLE</th>
<th>DUAL</th>
<th>SINGLE</th>
<th>DUAL</th>
<th>SINGLE</th>
<th>DUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RUN TIME (ms)</td>
<td>41.22</td>
<td>40.29</td>
<td>31.34</td>
<td>35.57</td>
<td>29.13</td>
<td>34.12</td>
<td>27.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>3.79</td>
<td>4.18</td>
<td>5.02</td>
<td>4.74</td>
<td>5.49</td>
<td>5.05</td>
<td>5.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.16</td>
<td>0.17</td>
<td>0.16</td>
<td>0.17</td>
<td>0.16</td>
<td>0.16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>6.44</td>
<td>6.78</td>
<td>4.93</td>
<td>5.99</td>
<td>4.65</td>
<td>5.87</td>
<td>4.44</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RUN TIME (ms)</td>
<td>12.26</td>
<td>11.62</td>
<td>9.31</td>
<td>10.10</td>
<td>8.14</td>
<td>9.96</td>
<td>7.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>5.69</td>
<td>6.31</td>
<td>7.55</td>
<td>7.43</td>
<td>8.84</td>
<td>7.66</td>
<td>9.21</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td>0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>0.86</td>
<td>0.85</td>
<td>0.65</td>
<td>0.75</td>
<td>0.58</td>
<td>0.75</td>
<td>0.57</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RUN TIME (ms)</td>
<td>27.17</td>
<td>25.74</td>
<td>20.68</td>
<td>22.18</td>
<td>17.85</td>
<td>21.84</td>
<td>17.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>4.86</td>
<td>5.41</td>
<td>6.43</td>
<td>6.42</td>
<td>7.61</td>
<td>6.65</td>
<td>7.98</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
<td>0.14</td>
<td>0.14</td>
<td>0.14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>3.59</td>
<td>3.58</td>
<td>2.75</td>
<td>3.14</td>
<td>2.41</td>
<td>3.16</td>
<td>2.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RUN TIME (ms)</td>
<td>41.24</td>
<td>39.58</td>
<td>31.31</td>
<td>34.26</td>
<td>27.81</td>
<td>33.31</td>
<td>26.76</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>2.72</td>
<td>3.11</td>
<td>3.62</td>
<td>3.63</td>
<td>4.15</td>
<td>3.82</td>
<td>4.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.11</td>
<td>0.12</td>
<td>0.11</td>
<td>0.12</td>
<td>0.12</td>
<td>0.13</td>
<td>0.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>4.63</td>
<td>4.87</td>
<td>3.54</td>
<td>4.25</td>
<td>3.20</td>
<td>4.23</td>
<td>3.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RUN TIME (ms)</td>
<td>27.12</td>
<td>25.69</td>
<td>20.64</td>
<td>22.14</td>
<td>17.81</td>
<td>21.80</td>
<td>17.30</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>4.84</td>
<td>5.38</td>
<td>6.40</td>
<td>6.38</td>
<td>7.57</td>
<td>6.61</td>
<td>7.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
<td>0.14</td>
<td>0.13</td>
<td>0.14</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>3.56</td>
<td>3.55</td>
<td>2.72</td>
<td>3.12</td>
<td>2.39</td>
<td>3.13</td>
<td>2.36</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RUN TIME (ms)</td>
<td>40.31</td>
<td>39.14</td>
<td>30.63</td>
<td>34.21</td>
<td>27.93</td>
<td>33.20</td>
<td>26.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>POWER (W)</td>
<td>3.90</td>
<td>4.32</td>
<td>5.18</td>
<td>4.97</td>
<td>5.77</td>
<td>5.25</td>
<td>6.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENERGY (J)</td>
<td>0.16</td>
<td>0.17</td>
<td>0.16</td>
<td>0.17</td>
<td>0.16</td>
<td>0.17</td>
<td>0.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EDP (ms*J)</td>
<td>6.34</td>
<td>6.62</td>
<td>4.86</td>
<td>5.80</td>
<td>4.50</td>
<td>5.78</td>
<td>4.36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.4 Reliable Overclocking Analysis

The Razor architecture is among the first in which the voltage was decreased below worst-case settings without altering the frequency during run time [27]. In a 64-bit Alpha processor only 192 flip-flops out of 2048 flip-flops required Razor augmentation. This amounted to a power overhead of 1%. Razor architecture was analyzed at various levels: First, an $18 \times 18$ bit multiplier is implemented and analyzed in FPGA. Then a C-level timing model of a Kogge-Stone adder is implemented with Razor timing details from SPICE analysis. This C-model is then integrated into the execution stage of SimpleScalar simulator. Overall, substantial energy savings of up to 62% was observed with less than 3% impact on performance due to error recovery.

In contrast to Razor’s energy saving goals, the SPRIT$^3$E framework varied the frequency dynamically at a fixed voltage level [92]. To gauge the performance improvements provided by the SPRIT$^3$E framework, an initial experiment on multiplier circuits indicated that using a method such as dual latching to tolerate a small amount of timing errors allowed the multiplier circuit to run at almost half the period, a speedup of 44%. SPRIT$^3$E framework was also evaluated in a DLX superscalar processor. Experimental results show that on an average a performance gain up to 57% across all benchmark applications is achievable.

The most important thing that makes speculative reliable overclocking so attractive is the possibility of allowing and tolerating timing errors. An important design constraint is fixing the target error rate. We evaluated the performance of ROP with different error rate targets. For this analysis, we used 45nm technology parameters. Also, the results presented are for a reliably overclocked system that tunes its frequency using the two clock generator system shown in Figure 4.6.

Figures 4.7 and 4.8 shows the performance trends for SPEC2000 integer and floating point benchmarks as we vary target error rate from 0% to 20%. The processor operates in normal mode without any speculative reliable overclocking for 0% error rate target. From the plots, we can see that all benchmarks show significant improvement in run time with reliable overclocking. However, as the error rate target is increased, the error recovery penalty plays a role and offsets some of the improvement in performance. This is seen from stagnating or decreasing improvement in run time for more than 5% error rate target.
The energy curves show that even as run time decreases, the total energy consumption increases modestly with higher error rate targets. This increase comes from the significant increase in power consumption for reliably overclocked systems. However, the energy-delay product metric shows that reliably overclocking is beneficial for systems, which require both energy-efficiency and high performance. From the plots, we understand that 5% is a good target error rate, as it minimizes both run time and energy-delay product.

Tables 4.6 and 4.7 present the percent increase or decrease for run time, power and energy consumption, energy-delay product and temperature when reliably overclocked, as compared to the base processor. A positive value for percent difference indicates a favorable improvement because of reliable overclocking, while a negative value indicates a decline. Power consumption increases by an average of 38% across all benchmarks, while energy consumption increased on an average by only 3%. The significant improvement in run time and energy delay product validates the reliable overclocking scheme. Change in the maximum temperature reached is also within 3%.

The comparison results presented until now assumes that the memory is also reliably overclocked. If we assume that only the processor is overclocked, then memory intensive applications may not see significant benefits. This is reflected in Tables 4.8 and 4.9. Integer benchmarks bzip2 and gzip, and floating point benchmark mgrid are memory intensive benchmarks and benefit minimally with overclocking, as the memory latency plays a key role in the run time. A non-memory intensive benchmark, crafty for instance, is more computation oriented and we notice that there is no difference in performance with or without memory overclocking. We observe the same behavior for gap, mcf, vpr, apsi, equake, and galgel. applu is moderately memory intensive and suffers a 6.25% performance degradation with no memory overclocking.
Figure 4.7 Run time, energy and energy-delay product trends for SPEC2000 integer benchmarks as target error rate varies from 0% to 20%. All values are normalized to 0% target error rate (no overclocking mode).
Figure 4.8  Run time, energy and energy-delay product trends for SPEC2000 floating point benchmarks as target error rate varies from 0% to 20%. All values are normalized to 0% target error rate (no overclocking mode).
### Table 4.6 Comparing various performance metrics for non-overclocked and reliably overclocked processors executing SPEC2000 integer benchmarks

<table>
<thead>
<tr>
<th>METRIC</th>
<th>bzip2</th>
<th></th>
<th>crafty</th>
<th></th>
<th>gap</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>ROP</td>
<td>% Diff</td>
<td>Base</td>
<td>ROP</td>
<td>% Diff</td>
</tr>
<tr>
<td>RUN TIME (ms)</td>
<td>40.08</td>
<td>24.67</td>
<td>38.45</td>
<td>28.30</td>
<td>17.89</td>
<td>36.76</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>2.55</td>
<td>4.27</td>
<td>-67.39</td>
<td>3.48</td>
<td>5.82</td>
<td>-67.14</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.10</td>
<td>0.11</td>
<td>-2.88</td>
<td>0.10</td>
<td>0.10</td>
<td>-5.11</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>4.09</td>
<td>2.59</td>
<td>36.68</td>
<td>2.79</td>
<td>1.85</td>
<td>33.53</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>325</td>
<td>330</td>
<td>-1.52</td>
<td>329</td>
<td>337</td>
<td>-2.33</td>
</tr>
</tbody>
</table>

### Table 4.7 Comparing various performance metrics for non-overclocked and reliably overclocked processors executing SPEC2000 floating point benchmarks

<table>
<thead>
<tr>
<th>METRIC</th>
<th>applu</th>
<th></th>
<th>equake</th>
<th></th>
<th>mgrid</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>ROP</td>
<td>% Diff</td>
<td>Base</td>
<td>ROP</td>
<td>% Diff</td>
</tr>
<tr>
<td>RUN TIME (ms)</td>
<td>52.25</td>
<td>32.21</td>
<td>38.36</td>
<td>26.03</td>
<td>16.58</td>
<td>36.30</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>2.75</td>
<td>4.64</td>
<td>-68.46</td>
<td>5.36</td>
<td>8.82</td>
<td>-64.48</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.14</td>
<td>0.15</td>
<td>-3.65</td>
<td>0.14</td>
<td>0.15</td>
<td>-4.31</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>7.52</td>
<td>4.80</td>
<td>36.12</td>
<td>3.63</td>
<td>2.41</td>
<td>33.55</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>329</td>
<td>337</td>
<td>-2.30</td>
<td>333</td>
<td>343</td>
<td>-3.06</td>
</tr>
</tbody>
</table>

### Table 4.8 Comparing various performance metrics for non-overclocked and reliably overclocked processors executing SPEC2000 floating point benchmarks

<table>
<thead>
<tr>
<th>METRIC</th>
<th>apsi</th>
<th></th>
<th>galgel</th>
<th></th>
<th>mesa</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Base</td>
<td>ROP</td>
<td>% Diff</td>
<td>Base</td>
<td>ROP</td>
<td>% Diff</td>
</tr>
<tr>
<td>RUN TIME (ms)</td>
<td>12.26</td>
<td>7.94</td>
<td>35.24</td>
<td>41.24</td>
<td>26.76</td>
<td>35.12</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>5.69</td>
<td>9.21</td>
<td>-61.82</td>
<td>2.72</td>
<td>4.40</td>
<td>-61.51</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.07</td>
<td>0.07</td>
<td>-3.72</td>
<td>0.11</td>
<td>0.12</td>
<td>-4.49</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>0.86</td>
<td>0.57</td>
<td>32.83</td>
<td>4.63</td>
<td>3.14</td>
<td>32.21</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>333</td>
<td>344</td>
<td>-3.08</td>
<td>328</td>
<td>334</td>
<td>-1.84</td>
</tr>
</tbody>
</table>
Table 4.8 Effect of memory overclocking on the performance benefits of a ROP executing SPEC2000 integer benchmarks

<table>
<thead>
<tr>
<th>METRIC</th>
<th>bzip2</th>
<th>crafty</th>
<th>gap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem*RO</td>
<td>Percent</td>
<td>Mem*RO</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RUN TIME (ms)</td>
<td>24.67</td>
<td>36.51</td>
<td>-48.01</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>4.27</td>
<td>3.39</td>
<td>20.52</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.11</td>
<td>0.12</td>
<td>-17.86</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>2.59</td>
<td>4.52</td>
<td>-74.43</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>330</td>
<td>328</td>
<td>0.56</td>
</tr>
</tbody>
</table>

Table 4.9 Effect of memory overclocking on the performance benefits of a ROP executing SPEC2000 floating point benchmarks

<table>
<thead>
<tr>
<th>METRIC</th>
<th>applu</th>
<th>equake</th>
<th>mgrid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem*RO</td>
<td>Percent</td>
<td>Mem*RO</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RUN TIME (ms)</td>
<td>27.40</td>
<td>29.11</td>
<td>-6.25</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>5.93</td>
<td>5.72</td>
<td>3.61</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.16</td>
<td>0.17</td>
<td>-2.44</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>4.44</td>
<td>4.83</td>
<td>-8.85</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>344</td>
<td>343</td>
<td>0.35</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>METRIC</th>
<th>apsi</th>
<th>galgel</th>
<th>mesa</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mem*RO</td>
<td>Percent</td>
<td>Mem*RO</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RUN TIME (ms)</td>
<td>7.94</td>
<td>7.95</td>
<td>-0.16</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>9.21</td>
<td>9.20</td>
<td>0.09</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.07</td>
<td>0.07</td>
<td>-0.07</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>0.57</td>
<td>0.58</td>
<td>-0.23</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>344</td>
<td>344</td>
<td>0.00</td>
</tr>
</tbody>
</table>
CHAPTER 5. THERMAL IMPACT OF RELIABLE OVERCLOCKING

The design for worst-case settings provides us an opportunity to improve processor performance to a greater extent through overclocking. Reliable overclocking mechanisms employ proven fault tolerance techniques to detect and recover from any timing errors that occur at better-than-worst-case speeds. Although aggressive clocking mechanisms facilitate in improving performance, they adversely impact on-chip temperatures, leading to hotspots. Overclocking enthusiasts invest heavily in expensive cooling solutions to protect the chip from overheating, and such overclocked systems typically have significantly lower lifetime. Additionally, reliable overclocking techniques necessitate additional circuitry, leading to an increase in power consumption. Higher clock speeds and power densities invariably lead to accretion of on-chip temperature over a period of time. As system operates faster, on-chip temperatures quickly reach and exceed the safe limits. This poses a serious threat to the lifetime reliability of these systems [90]. In this chapter, we perform a comparative analysis on the thermal behavior of reliably overclocked systems with non-accelerated systems. Our aim in this work is to establish a safe acceleration zone for such ‘better than worst-case’ designs by efficiently balancing the gains of overclocking and the impact on system temperature.

We must emphasize that current products from both the leading microprocessor vendors, Intel and AMD, have dynamic thermal monitoring techniques that take necessary corrective action to maintain on-chip temperature [2, 55, 75]. The corrective actions, in most cases, shut down the system or reduce system voltage and frequency, leading to considerable performance degradation. Our goal in this study is to analyze the temperature pattern of reliably overclocked systems, and evaluate the lifetime reliability of such reliable aggressive clocking mechanisms. Furthermore, we monitor the on-chip temperature of aggressively overclocked systems that dynamically enhance single threaded application performance. We couple thermal monitoring techniques with reliable overclocking to alleviate lateral
issues relating to system power and reliability. While taking feedback from an integrated thermal monitor, we observed an average performance increase of 25%, while operating within temperature 355K. To the best of our knowledge, this is the first work that analyzes the impact of reliable overclocking on on-chip temperature.

5.1 Thermal and Reliability Management

Over the last decade, thermal awareness has gained importance distinguishing itself from power awareness. Processor chips began to have thermal sensors in various locations to regularly sample the temperature and to shut down the operation in case of overheating. However, rapid heating and cooling of processor chips create thermal cycles affecting the lifetime reliability of the system [90].

The power consumed by a VLSI chip consists of two parts: dynamic and static. Dynamic power is dependent on capacitance (C), voltage (V), frequency (f) and switching factor (α), and is given by

\[ P_{dyn} = \alpha CV^2 f. \]

Since dynamic power is directly proportional to the frequency at which the circuit operates, this causes overclocked systems to consume more power, which in turn causes systems to overheat. However, solving the thermal problem is not as simple as bringing down the overall power consumed [86].

The thermal problem becomes much more noticeable in designs under 90nm technology, where leakage power grows significantly. The leakage power grows exponentially with temperature as given by the empirical relationship in Equation 4.16 [40]. Leakage power suffers from a positive feedback, wherein, increase in temperature leads to further leakage and increased total power consumption, which in turn leads to increase in temperature. Due to non-uniform switching and leakage, temperature is not distributed uniformly across the chip, creating localized heating in parts leading to hotspots.

Higher temperatures not only increase power budget, but also affect the lifetime reliability of the devices. To improve the overall reliability and lifetime of the systems, the thermal performance should be monitored and the average degradation of transistors managed. RAMP model [90] relates thermal cycling to mean time to failure due to various factors such as, electromigration, stress migration and dielectric breakdown and brings the importance of keeping the on-chip temperature within critical limits. Table 5.1 summarizes five critical failure mechanisms, namely, electromigration, stress migration, time
71

<table>
<thead>
<tr>
<th>Wear out Mechanism</th>
<th>Proportional Model (MTTF) and Fitting Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electromigration (EM) [70]</td>
<td>$(J)^{-n}e^{\frac{E_{aEM}}{kT}}$; $J=$Current Density; $n=1.1$, $E_{aEM}=0.9$eV</td>
</tr>
<tr>
<td>Stress Migration (SM) [70]</td>
<td>$</td>
</tr>
<tr>
<td>Time dependent dielectric breakdown (TDB) [104]</td>
<td>$(\frac{1}{T})^{(a-bT)}e^{\frac{[X+Y/T]+Z]}{kT}$; $a=78$, $b=-0.081$, $X=0.759$eV, $Y=66.8$eV/K, $Z=-8.37$eV/K</td>
</tr>
<tr>
<td>Thermal Cycling (TC) [90]</td>
<td>$(\frac{T}{T_{ambient}})^{q}$; $T_{ambient}$=Ambient Temperature; $q=2.35$</td>
</tr>
<tr>
<td>Negative Bias Temperature Instability (NBTI) [106]</td>
<td>$\left{ \ln \left( \frac{A}{1+2e^{B/T}} \right) - \ln \left( \frac{A}{1+2e^{B/T}} - C \right) \right} \times \frac{T}{e^{-D/T}} \beta_1$; $A=1.6328$, $B=0.07377$, $C=0.01$, $D=0.06852$, $\beta_1=0.3$</td>
</tr>
</tbody>
</table>

Table 5.1 Mean Time To Failure (MTTF) for critical wear out models

Dependent dielectric breakdown, thermal cycling and negative bias temperature instability as specified in [90], with their respective mean time to failure (MTTF). Here, $k$ is Boltzmann’s constant and $T$ is temperature in Kelvin. These wear out phenomena create impedance in the circuits, gradually leading to permanent device failures.

5.2 Analysis Framework for Estimating On-chip Temperature

The analysis framework for estimating on-chip temperature is similar to the one presented in Chapter 4. In order to demonstrate the full extent of the overheating problem, we disabled clock gating. Clock gating is a low power technique that is used to minimize dynamic power consumption during idle time. The benefits from low power techniques complement the benefits achieved through thermal throttling. The analysis framework requires few additions in order to support thermal throttling. Figure 5.1 depicts both timing error based feedback control, and thermal throttle. For our initial evaluation of how on-chip temperatures vary when reliably overclocked, we only observe the temperature, without employing any thermal throttle. We employ dynamic clock tuning beyond worst-case limits, using timing error based feedback control, to adapt system behavior based on workload characteristics. The number of timing errors occurring at a given time is based on the workload being executed by the processor.
5.2.1 Thermal Throttling

Thermal throttling is a technique in which system operation is throttled when on-chip temperature exceeds a critical value. In Figure 5.1, the HotSpot thermal model estimates the on-chip temperature during run-time. The current on-chip temperature is compared with an user or manufacturer defined critical value, and the control unit takes corrective actions. Our implementation of thermal throttle reduces system operating frequency, whenever the on-chip temperature exceeds the critical value. We restrict reliable overclocking based on the on-chip temperature. Frequency is increased only if the timing error rate value is below the target error rate. However, frequency is decreased if either the timing error rate exceeds the timing error set point value or the on-chip temperature exceeds the critical temperature value.

5.2.2 Simulation Parameters

Table 5.2 presents the simulation parameters. We evaluate the system temperature while running at 1.25V. From Figure 3.1 in Chapter 3, we can see that clock period can be scaled only up to 50% of the original cycle time. We assume up to 45% overclocking. Table 5.2 provides the worst-case frequency and the maximum overclocked frequency we considered for our simulations. We perform a binary
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology node</td>
<td>45nm</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.25V</td>
</tr>
<tr>
<td>Minimum frequency</td>
<td>1024MHz</td>
</tr>
<tr>
<td>Maximum frequency</td>
<td>1862MHz</td>
</tr>
<tr>
<td>No. of frequency levels</td>
<td>32</td>
</tr>
<tr>
<td>Area</td>
<td>10mm²</td>
</tr>
<tr>
<td>Temperature sampling interval</td>
<td>1ms</td>
</tr>
<tr>
<td>Frequency sampling interval</td>
<td>100µs</td>
</tr>
<tr>
<td>Frequency penalty</td>
<td>Single PLL: 10µs</td>
</tr>
<tr>
<td></td>
<td>Dual PLL: 0µs</td>
</tr>
</tbody>
</table>

Table 5.2 Simulator parameters

search between 32 frequency levels within the allowed range, based on error rate and also temperature, when employing thermal throttle. We assume the presence of two phase-locked loops (PLLs), so that there is no performance penalty involved, while switching between frequencies. If there is only one PLL, it takes up to 10µs to change from one frequency to another.

5.3 On-chip Temperature Trends in Reliably Overclocked Processors

![Graph showing temperature and MTTF trends](image)

Figure 5.2 On-chip temperature trends and MTTF results for bzip2 benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling.
We simulated six SPEC INT 2000 benchmarks, namely bzip2, crafty, gcc, gzip, mcf and parser to analyze and compare the on-chip temperature trends and MTTF behavior for a non-overclocked processor, a reliably overclocked processor and a reliably overclocked processor with thermal throttling. We calculate MTTF based on the on-chip temperature at that given instant of time. We obtain the proportionality constant for our calculations from the baseline MTTF at 337K [90]. The MTTF values are obtained from the formulas mentioned in Table 5.1.

![Figure 5.3 On-chip temperature trends and MTTF results for crafty benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling.](image)

Figure 5.2 compares the transient temperature trends of a reliably overclocked processor with a non-overclocked processor for bzip2 benchmark. We evaluate ROP performance with and without thermal throttling. From the plots, we can clearly see that there is up to 15K difference between a reliably overclocked processor and a non-overclocked processor. Also, we see that the reliably overclocked processor reaches and exceeds 360K on executing around 3 million instructions.

Based on the cooling solution used, the system will reach a steady state temperature and remain there. In our experiments, a non-overclocked processor settles at 347K for the same cooling solution. We start our experiments at a steady state temperature of 340K. This initial temperature is based on the assumption that the system has already performed certain operations, before it executes the benchmark of interest.
Figure 5.4 On-chip temperature trends and MTTF results for gzip benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling.

Figure 5.5 On-chip temperature trends and MTTF results for mcf benchmark. The plots show how on-chip temperature and MTTF varies for a non-overclocked processor, a reliably overclocked processor, and a reliably overclocked processor with thermal throttling.
When incorporating thermal throttle, we find that the temperature gets clamped at the desired choice of operating temperature. Since thermal sensor outputs are available once every ms, it is good to choose a temperature 3K below the critical temperature, so that even if the system temperature overshoots before getting a thermal measurement, it will not exceed the critical temperature.

Figure 5.2 shows the mean time to failure of reliably overclocked system, with and without thermal throttle, as compared to the non-overclocked system. We observe that a non-overclocked system has a longer lifetime, of about 30 years, as its on-chip temperature does not exceed 347K. However, a reliably overclocked system has a much shorter lifetime of about 9 years. Applying thermal throttling at about 355K increased the system lifetime to about 14 years. We understand from the figure that running the \textit{bzip2} benchmark at lower temperatures over a long period of time improves the MTTF significantly. This motivates the need for having efficient dynamic thermal management techniques, alongside reliable overclocking, to achieve performance gain and reliability. Also, thermal management techniques alleviate the need for having an expensive cooling solution, making it cost effective to have high performance systems.

Figure 5.3, 5.4 and 5.5 show the on-chip temperature trends and MTTF results for \textit{crafty}, \textit{gzip} and \textit{mcf} benchmarks. As can be seen from the plots, the thermal characteristics and MTTF trends for \textit{crafty}, \textit{gzip} and \textit{mcf} benchmarks have similar nature to the ones recorded for \textit{bzip2} benchmark. Other two benchmarks, \textit{gcc} and \textit{parser}, have similar thermal and MTTF trends.

The relative speed-up for the six benchmarks, running $10^7$ instructions is illustrated in Figure 5.6. Reliable overclocking, on an average, achieves 35% increase in performance over a non-overclocked system. When a thermal throttle is applied, the performance gain drops to 25%.
Figure 5.6 Relative performance for SPEC2000 integer benchmarks
CHAPTER 6. RELIABLE OVERCLOCKING AND TECHNOLOGY SCALING

The continued and progressive scaling of the minimum feature size of the metal-oxide-semiconductor field-effect transistors (MOSFETs) has played a key role in the design of spectacular low-cost high performance computing systems [46]. ITRS uses the term “technology node” to indicate overall industry progress in integrated circuit (IC) feature scaling [29]. Even though, technology scaling reduces cost in the long run, the research, development and production cost involved in getting the next generation fabrication plant, at this level of sub-nm scaling, has resulted in diminishing return-on-investments, forcing many IC designers to adopt a fableless asset-lite business model. Unlike earlier practices, present and future technology generations call for more technology and design interactions very early in the design cycle to minimize cost and maximize yield. This required interaction adversely affects the fabless IC vendors resulting in ineffective time-to-market.

The race towards the next technology node has become a key part in retaining market share and competitive advantage in the semiconductor business, and in the design of high performance energy efficient systems [24, 36, 44]. The time-to-market difference gives a significant advantage to the bigger players in the semiconductor industry. To facilitate competitiveness and to create a level playing field for everyone involved, it becomes important to look at cheaper alternative solutions that enable the rest of the industry to compete without the need to immediately embrace technology scaling. Though it is not possible to avoid technology scaling altogether, in this paper, we strive to present a possible approach, which will either enable to bridge the gap between porting to the next technology node by extending the lifetime of a technology generation, or to skip a process technology generation altogether.

Our work in this chapter evaluates the competitiveness of timing speculation based adaptive reliable overclocking with respect to technology scaling. We make a convincing case for gains of reliable overclocking, quantifying such gains at different technology nodes, developing a methodology and
framework to evaluate such systems by estimating likely error rates using a synthesizable hardware model and applying them to the newly developed complete evaluation framework for realistic simulations.

We present the results of our experimental analysis based on integer and floating point SPEC2000 benchmarks running on a SimpleScalar Alpha processor simulator, augmented with error rate data obtained from hardware simulations of a superscalar processor. We compare the performance and energy management of the reliably overclocked systems to the non-overclocked systems that are implemented using different process technology nodes. Our evaluation results quantify the comparative gains achievable with reliable overclocking, and our substantive and significant results make a case for this approach to be a worthwhile technique to pursue in mainstream processor design.

6.1 Technology Scaling

The key driving force behind technology scaling is the market’s need for cost-efficient high performance, energy efficient computing systems. New process methods and materials have continuously emerged to surmount the seemingly impossible technology barriers, such as lithography and oxide scaling limits [36]. The semiconductor industry pursues technology scaling, irrespective of the cost involved, for the following reasons, as hypothesized by the scaling theory [24, 9]:

For every subsequent generational change in technology, gate delay reduces proportionally, contributing to the performance improvement seen in the operating frequency of a system that is ported to the scaled technology node. However, the increase in active and leakage power, observed in sub-100nm nodes, have subdued the quest for higher operating frequencies.

Until the 1990s, constant voltage scaling was practiced resulting in increased performance and higher active power. As power became an important design constraint owing to thermal design point requirements, semiconductor manufactures switched to constant electric field scaling to scale down supply voltage to minimize power dissipation. Constant field scaling requires threshold voltage to be scaled proportional to the feature size. However, threshold voltage scaling is limited by the sub-threshold slope, which in turn is limited to the thermal voltage, $V_T = \frac{kT}{q}$, where $k$ is the Boltzmann’s constant and $q$ is the electron charge.
As manufacturers reduce supply voltage with subsequent technology nodes, significant energy and active power savings are expected. One thing that worsens with sub-100nm technologies is the leakage power [44]. In MOS technology, leakage power comes from two sources: sub-threshold leakage and gate leakage. Sub-threshold leakage happens when current flows through the MOS transistors during their turn-off state, which happens when the gate-to-source ($V_{GS}$) voltage is lower than the sub-threshold voltage ($V_{TH}$).

When $V_{TH}$ is lowered proportionally with $V_{DD}$ to improve performance, since a large gate overdrive ($V_{GS} - V_{TH}$) is required to enable high speed switching, this results in 5x increase in sub-threshold leakage current. Gate leakage happens as the gate oxide between the metal gate and the channel becomes thinner and thinner resulting in tunneling current through the gate dielectric. Even as active power is independent of temperature, leakage power has an exponential dependence on device operating temperature. With scaling, keeping the devices cool becomes much more important, as leakage power becomes a significant part of the total power consumption.

Technology scaling complicates the on-chip communication as interconnects do not scale as well as logic gates [9]. Even as area and fringe capacitance decreases proportionally with downscaling, interconnect resistance and capacitance increases with scaling of wire width and thickness. Wire delays have started dominating the overall delay and in modern microprocessors pipeline stages are dedicated only for moving signals across the chip. More interconnect layers are added to subsequent technology to account for the increased density and complexity that comes with reduced area. Even as interconnect scaling is considered to be one of the bottlenecks that hinder future scaling, over the years changes in interconnect materials and architectures have favored interconnect performance to be in accordance with the transistor scaling trends.

In a nutshell, for every 30% downscaling of technology node, transistor density doubles, gate delay reduces by 30%, operating frequency improves by 43%, active power consumption halves, and energy savings of 65% is observed. However, recent technology generations are unable to scale clock frequency as desired because of low power requirements, process variations and reliability concerns. The trade-off between maintaining high speed switching and low-leakage remains an important design constraint and plays a key role in deciding the scaling trends of supply voltage and threshold voltage.
6.2 A Reliable Overclocking Approach

As discussed in the previous section, technology scaling does present a strong reason for the continued investment by semiconductor manufacturers. We do not dispute technology scaling. Our goal is to present an option to designers to use an enhancement in the current technology node and enable the longevity of the process technology node until the adoption of a new one. We believe that reliable overclocking presents an opportunity to VLSI designers to achieve significant performance improvements by considering it as an enhancement in every process technology node.

In this chapter, we make a viable case for a reliable overclocking approach that is at least as effective and competitive as technology scaling, albeit at a lower cost. Our technique either acts as an intermediate point between two technology nodes or presents an alternative to technology porting, a technique which is commonly done before moving on to the next architecture that consumes the extra silicon area created by scaling. Technology porting performs a die shrink on an existing design and achieves significant improvements in speed and power consumption. Our goal is to show that by reliably overclocking a VLSI chip design implemented in the current technology node, we can match and exceed the performance of the same design that is ported to the next technology node.

It is important to remember that even as scaling shrinks the device sizes, the die size tends to increase with improvements in yield enhancing manufacturing techniques. This allows multiple cores and hardware co-processors to be added to the system enabling high throughput parallel operation. Hence, technology scaling is necessary to accommodate significant changes in design architecture. We present reliable overclocking as a stopgap alternative and as a supplement to technology scaling.

Our approach is based on timing speculative reliable overclocking. Timing speculation is a technique by which a computing system performs aggressive computation, albeit incorrectly during a small fraction of instances. As the name suggests, the technique involves using data speculatively, and deploying efficient checking mechanisms to detect and correct erroneous computations. Timing errors resulting from accelerated computation and premature use of data are tolerated by exploiting proven fault tolerance techniques to ensure functional correctness. Many techniques have been proposed earlier that utilize different error tolerance mechanisms to tolerate timing errors. These techniques show significant improvement in performance as they enable faster execution of typical computations, and
suffer an error recovery penalty for rare occurrences of worst-case delay involving computations. Simply put, these techniques take advantage of Amdahl’s law, making the common case faster.

### 6.3 Analysis Framework

<table>
<thead>
<tr>
<th>PARAMETER</th>
<th>TECHNOLOGY NODE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>90nm</td>
</tr>
<tr>
<td>Supply Voltage, $V_{DD}$ (V)</td>
<td>1.2</td>
</tr>
<tr>
<td>Threshold Voltage, $V_T$ (V)</td>
<td>0.2943</td>
</tr>
<tr>
<td>Base Frequency, $f_{base}$ (MHz)</td>
<td>768</td>
</tr>
<tr>
<td>Overclocking Frequency Range (MHz)</td>
<td>768 - 1396</td>
</tr>
</tbody>
</table>

Table 6.1 Technology scaling parameters

In order to prove that speculative reliable overclocking can be considered as a viable stop-gap alternative for technology scaling, we adopt the analysis framework described in Chapter 4. For our analysis, we look at three technology nodes. Several IC designers are now either implementing their products in 45nm node or making the transition to 45nm node. The predecessors of 45nm node are 65nm and 90nm nodes. We obtained the scaling values for each of the three nodes from the Ss-PPC tool developed by University of Texas at Austin. Ss-PPC is a SimpleScalar simulator for the PowerPC instructions set architecture [78]. In Ss-PPC simulator, the technology dependent parameters are scaled from the values provided for 0.8um technology, which were originally presented as part of the Wattch power simulator. The technology nodes we chose are scaled approximately by 30% from one another and are currently used by industry. Other technology nodes in the vicinity of these nodes exist. However, performance characteristics do not change drastically with minor scaling, and our choice of technology nodes enables us to investigate the performance trends in state of the art semiconductor process technologies.

With technology scaling, resistance, capacitance, voltage and circuit delay values are scaled in accordance with the scaling theory. As discussed earlier in this chapter in Section 6.1, supply voltage and threshold voltage are scaled minimally to guarantee higher performance. Table 6.1 presents the scaled supply voltage and threshold voltage values across technologies. Because of the difficulty involved in changing the heat sink characteristics, the processor frequency for 90nm technology is fixed.
at 768 MHz, so that technology scaling and overclocking result in frequencies that maintain the temperature within acceptable limits. We assume that the processor is operating at room temperature (300K) before executing a benchmark. For 65nm and 45nm technologies, we scale the 90nm base frequency proportionally to 1063 MHz and 1536 MHz, respectively. Table 6.1 indicates the range of overclocking for the three technology nodes. While operating the simulator at much higher frequencies and at different technology nodes, as reported in Table 6.1, we use the error rates, discussed in detail in Section 4.2.1, relatively.

One important factor that needs to be taken care of with technology scaling is memory performance, since memory does not scale as well as logic. Over the years, for every 60% reduction in feature size for logic transistors, memory is scaled by 10% [53]. We have taken this into account while scaling from one technology to another.

Table 6.2 Comparing various performance metrics across different technology nodes for a non-overclocked processor executing SPEC2000 integer benchmarks

<table>
<thead>
<tr>
<th>Metric</th>
<th>BZIP2</th>
<th>CRAFTY</th>
<th>GAP</th>
<th>Scaling Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN TIME (ms)</td>
<td>57.20</td>
<td>48.11</td>
<td>40.08</td>
<td>40.08</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>2.48</td>
<td>2.55</td>
<td>2.73</td>
<td>2.73</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.14</td>
<td>0.10</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>8.11</td>
<td>5.81</td>
<td>4.09</td>
<td>4.09</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>321</td>
<td>322</td>
<td>325</td>
<td>322</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>GZIP</th>
<th>MCF</th>
<th>VPR</th>
<th>Scaling Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>RUN TIME (ms)</td>
<td>80.56</td>
<td>65.28</td>
<td>52.25</td>
<td>51.63</td>
</tr>
<tr>
<td>POWER (W)</td>
<td>2.61</td>
<td>2.69</td>
<td>2.75</td>
<td>4.45</td>
</tr>
<tr>
<td>ENERGY (J)</td>
<td>0.21</td>
<td>0.18</td>
<td>0.14</td>
<td>0.23</td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>16.94</td>
<td>11.47</td>
<td>7.52</td>
<td>11.87</td>
</tr>
<tr>
<td>TEMPERATURE (K)</td>
<td>322</td>
<td>324</td>
<td>329</td>
<td>323</td>
</tr>
</tbody>
</table>
Table 6.3 Comparing various performance metrics across different technology nodes for a non-overclocked processor executing SPEC2000 floating point benchmarks

<table>
<thead>
<tr>
<th>Metric</th>
<th>90nm</th>
<th>65nm</th>
<th>45nm</th>
<th>90nm</th>
<th>65nm</th>
<th>45nm</th>
<th>90nm</th>
<th>65nm</th>
<th>45nm</th>
<th>90nm</th>
<th>65nm</th>
<th>45nm</th>
<th>Better or Worse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time (ms)</td>
<td>78.84</td>
<td>58.02</td>
<td>41.22</td>
<td>54.33</td>
<td>39.26</td>
<td>27.17</td>
<td>68.84</td>
<td>53.22</td>
<td>40.31</td>
<td>Better</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power (W)</td>
<td>3.08</td>
<td>3.47</td>
<td>3.79</td>
<td>3.98</td>
<td>4.42</td>
<td>4.86</td>
<td>3.49</td>
<td>3.75</td>
<td>3.90</td>
<td>Worse</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy (J)</td>
<td>0.24</td>
<td>0.20</td>
<td>0.16</td>
<td>0.22</td>
<td>0.17</td>
<td>0.13</td>
<td>0.24</td>
<td>0.20</td>
<td>0.16</td>
<td>Better</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDP (ms*J)</td>
<td>19.15</td>
<td>11.67</td>
<td>6.44</td>
<td>11.74</td>
<td>6.81</td>
<td>3.59</td>
<td>16.53</td>
<td>10.61</td>
<td>6.34</td>
<td>Better</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature (K)</td>
<td>323</td>
<td>326</td>
<td>335</td>
<td>323</td>
<td>326</td>
<td>332</td>
<td>323</td>
<td>326</td>
<td>334</td>
<td>Worse</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.4 Performance at Different Technology Nodes

In this section, we evaluate the impact of technology scaling on performance. Tables 6.2 and 6.3 present the results for six integer and six floating point benchmarks, respectively, executing on the base processor implemented in 90nm, 65nm and 45nm technology. From the results, we find that going from 90nm to 65nm node, which is a 27.7% reduction in feature size, on an average, across both integer and floating point benchmarks, decreases run time by 25.47%, while increasing power consumption by about 10.09%. Because of the significant improvement in run time and moderate increase in power consumption, energy efficiency improved on an average by 18.09%. The energy-delay product drops by almost 38.90% indicating the significant benefits of technology scaling.

A switch from 65nm to 45nm node, which is a 30% reduction in feature size, on an average, across all twelve benchmarks, decreases run time by 27.98%, increases power consumption by about 8.77%, improves energy efficiency by 21.83%, and minimizes energy-delay product by 43.59%. Overall, the performance trends are very much in accordance with the scaling theory.
### 6.5 Comparing Technology Scaling with Reliable Overclocking

Having seen that both technology scaling and reliable overclocking improve run time and energy efficiency, let us now compare how they fare against each other. From Tables 4.6, 4.7, 6.2 and 6.3 we notice that technology scaling increases power moderately as compared to reliable overclocking. This is reflected in Figures 6.1 and 6.2. Power consumption values increase by almost 60% for reliable overclocking. A ROP implemented in 90\textit{nm} technology consumes more power than a non-overclocked processor implemented in the same technology, as well as the subsequent technologies. Temperature trends, as shown in Figures 6.3 and 6.4, for reliable overclocking are much better than power consumption trends. The on-chip temperatures of the ROP are higher than the non-overclocked processor implemented in the same technology. However, it is lower than the next technology generation.

Figures 6.5 and 6.6 show the run time trends for SPEC2000 integer and floating point benchmarks, respectively. Reliable overclocking results in better run time than technology scaling. This is because by setting an error rate target of 5% reliable overclocking allows operation at higher frequencies than possible with technology scaling.

Even as reliable overclocking outperforms technology scaling in terms of run time, the increase in power consumption is a worry. However, the significant decrease in run time, helps in keeping the increase in energy consumption within 3% for the ROP, as compared to the non-overclocked processor. The energy trends are shown in Figures 6.7 and 6.8.

Reliable overclocking scores over technology scaling when energy-delay product is considered. Figures 6.9 and 6.10 show that the energy-delay product for ROP is better or comparable to the non-overclocked processor implemented in the next technology node.

Overall, reliable overclocking presents a compelling reason to be looked at seriously by chip designers alongside technology scaling. Even as technology scaling is important for continued progress in the design of integrated circuits, speculative reliable overclocking presents a convincing case as a supplement to technology scaling. A reliable overclocking approach presents a stopgap alternative for porting to the next technology node, or it possibly enables certain products to skip a technology node altogether.
Figure 6.1 Technology scaling vs. speculative reliable overclocking: Power consumption trends for SPEC2000 integer benchmarks.

Figure 6.2 Technology scaling vs. speculative reliable overclocking: Power consumption trends for SPEC2000 floating point benchmarks.
Figure 6.3 Technology scaling vs. speculative reliable overclocking: Temperature trends for SPEC2000 integer benchmarks.

Figure 6.4 Technology scaling vs. speculative reliable overclocking: Temperature trends for SPEC2000 floating point benchmarks.
Figure 6.5  Technology scaling vs. speculative reliable overclocking: Run time for SPEC2000 integer benchmarks. All values are normalized to 90nm run time.

Figure 6.6  Technology scaling vs. speculative reliable overclocking: Run time for SPEC2000 floating point benchmarks. All values are normalized to 90nm run time.
Figure 6.7 Technology scaling vs. speculative reliable overclocking: Energy consumption for SPEC2000 integer benchmarks. All values are normalized to 90nm energy values.

Figure 6.8 Technology scaling vs. speculative reliable overclocking: Energy consumption for SPEC2000 floating point benchmarks. All values are normalized to 90nm energy values.
Figure 6.9 Technology scaling vs. speculative reliable overclocking: EDP for SPEC2000 integer benchmarks. All values are normalized to 90nm EDP.

Figure 6.10 Technology scaling vs. speculative reliable overclocking: EDP for SPEC2000 floating point benchmarks. All values are normalized to 90nm EDP.
CHAPTER 7. FAULT TOLERANT AGGRESSIVE SYSTEMS

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 32\textit{nm} technologies and beyond. Soft errors, which are transient in nature, and silicon defects, which lead to permanent failures, have intrigued researchers to formulate fault tolerance techniques with varied capabilities to improve system reliability. Soft errors, induced by high energy radiation and external noise, have become more frequent and may result in incorrect computation and silent data corruption. Intermittent faults that persist for a short duration of time at one particular location are also a cause for concern [20]. Silicon defects resulting from silicon failure mechanisms such as transistor wear outs, gate breakdown, hot carrier degradation, and manufacturing limitations degrade lifetime and reliability of fabricated devices. Transient, intermittent and permanent fault classes constitute the three major reasons for hardware failure.

In this chapter, we develop a conjoined duplex system approach to provide tolerance for myriad hardware faults that plague modern computing systems. Our approach is capable of protecting both the datapath and control logic. With minor additions to the error recovery procedure used in SPRIT\textsuperscript{3}E, our fault tolerant aggressive system is capable of recovering from soft errors and timing errors. 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.

Our Conjoined Pipeline (in short, \textit{CPipe}) system 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 systems. The \textit{CPipe} system builds on the better-than-worst-case design method-
ologies [3] proposed in Razor [27] and SPRIT$^3$E [92] performance enhancement techniques. In CPipe, both the pipeline registers and the pipeline stage combinational logic are replicated. The term ”conjoined” implies the intertwining of the two pipelines and their constant and continued dependency on each other.

SPRIT$^3$E 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, SPRIT$^3$E demonstrated that considerable performance improvement can be achieved through reliable overclocking. However, in the presence of faults, the redundant register cannot be relied upon, and this necessitates spatial redundancy of combinational logic to ensure that the value stored in the redundant register is “gold”. Our CPipe system is designed to tolerate transient and intermittent faults along with timing errors, and implements a robust error detection and recovery mechanism.

The contamination delay of the redundant pipeline is increased to support the operation of the CPipe system. Depending on the extent of overclocking desired, the contamination delay is increased accordingly. The contamination delay of the primary pipeline is not increased, and this allows the CPipe approach to have less timing errors at a particular better-than-worst-case frequency, as compared to Razor or SPRIT$^3$E approach. This is because, increasing contamination delay affects non-short path delays too, even though the overall worst-case propagation delay is not increased. An input data combination, which did not cause an error, originally at a particular frequency, may result in an error after the contamination delay is increased.

The CPipe system benefits from a dynamic clock tuning mechanism that is capable of adapting the system clock frequency to the optimal value based on the current executing application and the environmental conditions. The range of frequencies at which the CPipe system operates reliably is estimated based on the implementation of the CPipe datapath and error recovery logic. The frequency is tuned during run-time in the range to maximize performance.

We performed a series of experiments to evaluate the fault tolerance and overclocking capability of the CPipe technique. We designed and implemented a two stage conjoined arithmetic pipeline for this purpose. The first stage performs 64-bit carry look ahead addition, and the second stage performs 32-bit multiplication of the most significant and least significant words of the adder output. Separate
experiments were carried out to verify detection and recovery from soft errors, timing errors, and intermittent faults. Permanent fault detection is also verified. Our fault injection campaign indicated fault masking in case of soft errors. The output of the pipeline was verified for correctness, and it was made certain that all randomly injected faults were detected and recovered from.

To prove that our CPipe technique is viable in the presence of feedback signals from subsequent pipeline stages, we implemented our technique in a five-stage in-order pipeline processor supporting DLX instruction set architecture. The implemented processor supports data forwarding and hazard detection. We subjected the conjoined processor to faults, and performed analysis for three different microbenchmarks. Our results demonstrate sufficient confidence in the correct working of our technique, and indicate the possibility of extending our technique in out-of-order systems too.

7.1 Conjoined Pipeline Architecture

The basic principle behind the CPipe system architecture is to replicate the entire pipeline, and interlink the two pipelines in a way so as to provide capability to tolerate various fault types. Both primary and redundant pipelines are susceptible to faults that are uniformly distributed in time and space. Timing errors occur, if 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. Since, the redundant pipeline can be corrupted because of a fault occurrence, the error detection and recovery process is more complex than that described in Razor and SPRIT\textsuperscript{3}E techniques. 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.

7.1.1 Conjoined Pipeline Datapath Description

The organization of redundancy in CPipe is illustrated in Figure 7.1. 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
Figure 7.1 Conjoined Pipeline Architecture. Shaded region represents the L-Pipeline. Dotted line encompasses the Local Fault Detection and Recovery (LFDR) circuit.
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 is 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 7.1 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.

The CPipe architecture requires three clocks for proper operation. The three input clocks are the leader clock, \( L_{\text{Clk}} \), the error clock, \( E_{\text{Clk}} \), and the shadow clock, \( S_{\text{Clk}} \). \( E_{\text{Clk}} \) and \( S_{\text{Clk}} \) are phase shifted versions of the \( L_{\text{Clk}} \). These three clocks, along with the error signals from all the pipeline stages, control \( L_{\text{GClk}} \) and \( S_{\text{GClk}} \). The clocks \( L_{\text{GClk}} \) and \( S_{\text{GClk}} \) are gated versions of \( L_{\text{Clk}} \) and \( S_{\text{Clk}} \). The L-Pipeline registers are clocked by \( L_{\text{GClk}} \), while the S-Pipeline registers are clocked by \( S_{\text{GClk}} \). \( E_{\text{Clk}} \) is required to precisely control when \( L_{\text{Clk}} \) and \( S_{\text{Clk}} \) 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 \( \text{Load}_{SP} \) in Figure 7.1, is asserted for a cycle. This entire control mechanism is performed in the clock stall control module, shown as \( \text{CLK STALL CNTRL} \) in Figure 7.1.

### 7.1.2 Error Detection and Recovery

As mentioned earlier, the results computed by the S-Logic are free from timing errors, but susceptible to soft errors. This complicates the error detection and recovery process. It is very important
to ensure that the S-PIPELINE register is not corrupted with incorrect result; otherwise recovery will not be possible. Considering this complication, in the CPipe architecture error detection is performed before storing the results in the S-PIPELINE register. Only if the results computed by the S-LOGIC matches the values registered in the L-PIPELINE register, then the S-LOGIC outputs are written into the S-PIPELINE register. The E-DETECT module incorporates metastability detection, similar to the one described in [22], for the L-PIPELINE register, 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 of the S-LOGIC needs to be increased to a value more than the delay between the $L_{GClk}$ and the $S_{GClk}$. 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. It is not possible to differentiate between these errors. The transient fault tolerance mechanism is overloaded to detect and recover from any timing errors that might occur because of overclocking.

Figure 7.2 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 7.1). 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
Figure 7.2 Waveforms highlighting error detection and recovery in a Conjoined Pipeline system
the mismatch between the L-P 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 $E_{\text{Clk}}$ in the CLK STALL CNTRL module. The latched signal is referred to as $GError$ in Figure 7.2. The global error signal ensures correct execution of the pipeline, and helps in global recovery. The $GError$ signal is asserted or deasserted only when the recovery counter ($RCounter$) 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 7.2 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 7.2): Error flag is asserted by the E-DETECT module, and $GError$ goes high at the rising edge of $E_{\text{Clk}}$. Immediately after $GError$ 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 $S_{\text{GCclk}}$ is low, and value from S-Pipeline registers are loaded into the corresponding L-Pipeline registers. $RCounter$ is incremented at the end of the cycle.

- **SECOND**: $L_{\text{ClkStall}}$ goes low at the negative edge of $L_{\text{Clk}}$. This stalls $L_{\text{GCclk}}$ 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. $RCounter$ 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. $RCounter$ is incremented at the end of the cycle.

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 7.2, 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, and all possible combination of errors. Table 7.1 shows the possible error scenarios that can happen in a CPipe system.

<table>
<thead>
<tr>
<th>Case</th>
<th>L-PIPELINE</th>
<th>S-PIPELINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Soft Error</td>
<td>No Error</td>
</tr>
<tr>
<td>2.</td>
<td>Soft Error</td>
<td>Soft Error</td>
</tr>
<tr>
<td>3.</td>
<td>Timing Error</td>
<td>No Error</td>
</tr>
<tr>
<td>4.</td>
<td>Timing Error</td>
<td>Soft Error</td>
</tr>
<tr>
<td>5.</td>
<td>No Error</td>
<td>Soft Error</td>
</tr>
</tbody>
</table>

**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 thesis, this possibility is not pursued further. However, 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 de-asserted, indicating that there will be no timing errors in the L-PIPELINE. In this case, error recovery takes two cycles, as $L_{Clk}$ 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 signals, consequently affecting several flip-flops in the registers. 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 corrupting the S-PIPELINE register values. 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 GError signal go high. In essence, the CPipe architecture is capable of providing very high degrees of fault coverage.

7.2 Timing Requirements

For proper operation of systems implementing the CPipe architecture, it is of paramount importance to respect the timing relationship between the three clocks, namely, $L_{Clik}$, $E_{Clik}$ and $S_{Clik}$. To support reliable dynamic overclocking, certain governing conditions need to be met at all times. Figure 7.3
shows the parameters that control the full range of frequencies, $F_{\text{Min}} \sim F_{\text{Max}}$, that are possible when a 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$^3$E [92]. In a CPipe system, there are three clocks and two of those, $E_{\text{Clk}}$ and $S_{\text{Clk}}$, are phase shifted versions of the $L_{\text{Clk}}$. 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 the generation of the global error signal from the pipeline stage error signals.
- Let $T_{\text{SStall}}$ represent the time required to stall $S_{\text{Clk}}$ 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 $\text{LoadSP}$ 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 $T_{\text{FminCD}}$ represent the minimum contamination delay of the S-LOGIC of all the pipeline stages.

Figure 7.3 shows the time available for the above operations under $F_{\text{Min}}$ and $F_{\text{Max}}$ settings. $PS_{\text{Min}}$, defined by Equation 7.1, represents the minimum required phase shift to ensure correct operation, and it should satisfy Equation 7.2.

\[
PS_{\text{Min}} = T_{\text{Err}} + T_{\text{SStall}} \quad (7.1)
\]

\[
PS_{\text{Min}} \leq T_{\text{FminCD}} \quad (7.2)
\]

**Clock Timing Requirements:** $L_{\text{Clk}}$ active edge occurs first, followed by $E_{\text{Clk}}$ edge, and then $S_{\text{Clk}}$ edge. $E_{\text{Clk}}$ time lag should at least be equal to $T_{\text{Err}}$, and the $S_{\text{Clk}}$ phase shift amount should at least
be $PS_{Min}$. Fixing the phase shift between $E_{Clk}$ and $S_{Clk}$ as $T_{SS\text{Stall}}$ makes dynamic frequency operation easier, since only the phase shift between $L_{Clk}$ and $E_{Clk}$ needs to be controlled. Also, maintaining the phase shift value between $E_{Clk}$ and $S_{Clk}$ 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 $T_{Err}$, $T_{LSP}$ and $T_{SS\text{Stall}}$.

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

- Let $T_{Min}$ 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 $PS_{Max}$ represent the maximum phase shift required to ensure correct operation.
- Let $T_{F\text{maxCD}}$ represent the minimum contamination delay of the S-LOGIC of all the pipeline stages.

As seen in Figure 7.3, the only parameter that becomes critical because of frequency scaling is $T_{LSP}$. $T_{Err}$ and $T_{SS\text{Stall}}$ are taken care by the clock timing requirements, and their criticalness continue to remain the same as in $F_{Min}$ settings. Depending on the extent of overclocking required, the value $T_{F\text{maxCD}}$ is fixed at any value within the range given by Equation 7.3. 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.

$$PS_{Min} \leq T_{F\text{maxCD}} \leq T_{Min} \tag{7.3}$$

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, $T_{Min}$, is given by Equation 7.4. The corresponding phase shift, $PS_{Max}$, is given by Equation 7.5, and should satisfy Equation 7.6.

$$T_{Min} \leq \frac{T_{\text{Max}} + T_{Err} + T_{LSP}}{2} \tag{7.4}$$
\[ PS_{\text{Max}} = T_{\text{Max}} - T_{\text{Min}} + PS_{\text{Min}} \] (7.5)

\[ PS_{\text{Max}} \leq T_{F_{\text{maxCD}}} \] (7.6)

Let \( T_{PS} \) represent the adjustable phase shift value. Equation 7.7 defines the range of phase shift values, while Equation 7.8 defines the range for \( T_{PS} \).

\[ PS_{\text{Min}} \leq T_{PS} + T_{Err} + T_{SStall} \leq PS_{\text{Max}} \] (7.7)

\[ 0 \leq T_{PS} \leq T_{\text{Max}} - T_{\text{Min}} \] (7.8)

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 \( T \) given by Equation 7.9 and the phase shift \( PS \) given by Equation 7.10 are found.

\[ T = T_{\text{Max}} - T_{PS} \] (7.9)

\[ PS = PS_{\text{Min}} + T_{PS} \] (7.10)

**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.
7.3 Implementation Considerations

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 data bus 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 7.4 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 7.2 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.
We implemented and evaluated our clock generation methodology on a Xilinx Virtex 5 FPGA [105]. The Virtex 5 FPGA has support for digital clock managers (DCMs) and phase lock loops (PLLs).

For our purpose, we used the PLL in frequency synthesis mode and generated the three clocks with the required phase shifts in between them. The PLL has support for six output clocks with different phase shifts and frequencies. Using the dynamic reconfiguration port, it is possible to reconfigure during run-time the clock frequency, as well as the phase shift. Figure 7.5 shows the programmable part of the PLL. The multiplier and divider values are varied to get the required frequency.

![Figure 7.5 Clock generation circuitry](image)

We use three of the output clocks. Each of the output clocks are programmed to have the required phase shift. Based on information provided by Xilinx, we adjusted the phase shift in increments of 11.25 degrees, where 0 degree corresponds to no phase shift, and 360 degrees corresponds to one full clock period shift. This corresponds to 32 different clock frequencies between minimum and maximum possible frequencies. Finer phase shifts are also possible. Also the $S_{C_{lk}}$ is phase shifted a constant value from the $E_{C_{lk}}$ based on the value of $T_{S_{nall}}$, which for experimentation was fixed at 1ns.

When the output of the VCO is close to 1GHz the PLL locks in approximately 1400 cycles, measured at a reference clock frequency of 100MHz. The output of VCO is further divided to generate each of the output clocks. The PLL takes upwards of 1500 cycles when the VCO output frequency is less than 1GHz. We used a look up table for reconfiguring during run-time the values of the multiplier, divider, and the values of output clock dividers and phase shifts.
7.3.1 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, $E_{Clk}$, is used only to clock the error register. The duty cycle of $L_{Clk}$ 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 $L_{Clk}$. The assertion and global routing delay of the two stall signals, $L_{Clk}\text{Stall}$ and $S_{Clk}\text{Stall}$, 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 $E_{Clk}$ is kept below $T_{High} - T_{SStall}$. We implemented and evaluated this approach too on the Virtex 5 FPGA. The duty cycle for each of the output clocks is as easily programmable as the phase shift of the output clocks.

7.4 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 [91]. From static timing analysis reports, we estimated the values of $T_{Max}$ as 9.1ns, $T_{Err}$ as 1.7ns, $T_{SStall}$ as 0.67ns, and $T_{LSP}$ as 1.85ns. Then, using the equations derived in Section 7.2 we calculated the values of $T_{Min}$ as 6.33ns and $PS_{Max}$ 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 $T_{\text{Min}}$, 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 circuit and used the delay values for locking that we obtained from the Xilinx Virtex 5 experimentation.

In our experiment, for an 1ms 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.

Table 7.2 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>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Injected</td>
<td>Detected</td>
<td>Injected</td>
</tr>
<tr>
<td>NOOC</td>
<td>1</td>
<td>109818</td>
<td>981</td>
<td>131</td>
<td>27</td>
</tr>
<tr>
<td>NOOC</td>
<td>2</td>
<td>109892</td>
<td>937</td>
<td>105</td>
<td>26</td>
</tr>
<tr>
<td>NOOC</td>
<td>3</td>
<td>109772</td>
<td>941</td>
<td>124</td>
<td>27</td>
</tr>
<tr>
<td>MAXOC</td>
<td>1</td>
<td>141054</td>
<td>913</td>
<td>216</td>
<td>27</td>
</tr>
<tr>
<td>MAXOC</td>
<td>2</td>
<td>140976</td>
<td>953</td>
<td>193</td>
<td>27</td>
</tr>
<tr>
<td>MAXOC</td>
<td>3</td>
<td>140879</td>
<td>919</td>
<td>214</td>
<td>31</td>
</tr>
<tr>
<td>DYNOC</td>
<td>1</td>
<td>132975</td>
<td>925</td>
<td>207</td>
<td>26</td>
</tr>
<tr>
<td>DYNOC</td>
<td>2</td>
<td>133053</td>
<td>914</td>
<td>177</td>
<td>25</td>
</tr>
<tr>
<td>DYNOC</td>
<td>3</td>
<td>133031</td>
<td>933</td>
<td>190</td>
<td>26</td>
</tr>
</tbody>
</table>

We repeated the experiment with three different random seeds, and performed the experiments in three different modes. The three modes were, no overclocking (NOOC), $T_{\text{Min}} = 9.1\text{ns}$, $T_{\text{Max}} = 9.1\text{ns}$, maximum overclocking (MAXOC), $T_{\text{Min}} = 7\text{ns}$, $T_{\text{Max}} = 7\text{ns}$, and dynamic overclocking (DYNOC), $T_{\text{Min}} = 7\text{ns}$, $T_{\text{Max}} = 9.1\text{ns}$. Table 7.2 reports results for the three types of faults injected, and it also presents the number of correct operations performed in 1ms time duration.

In [92], 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 CPipe is limited. Even while running at maximum possible frequency, for randomly generated inputs we observed extremely less timing errors, as reported in Table 7.3.

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 the modified binary search algorithm, described in Algorithm 1, 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 at MAXOC. DYNOC offers about 21% performance increase over NOOC.

Table 7.3 Timing errors

<table>
<thead>
<tr>
<th>Mode</th>
<th>Run</th>
<th>Operations</th>
<th>Timing Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOOC</td>
<td>1</td>
<td>109818</td>
<td>0</td>
</tr>
<tr>
<td>NOOC</td>
<td>2</td>
<td>109892</td>
<td>0</td>
</tr>
<tr>
<td>NOOC</td>
<td>3</td>
<td>109772</td>
<td>0</td>
</tr>
<tr>
<td>MAXOC</td>
<td>1</td>
<td>141054</td>
<td>13</td>
</tr>
<tr>
<td>MAXOC</td>
<td>2</td>
<td>140976</td>
<td>10</td>
</tr>
<tr>
<td>MAXOC</td>
<td>3</td>
<td>140879</td>
<td>13</td>
</tr>
<tr>
<td>DYNOC</td>
<td>1</td>
<td>132975</td>
<td>13</td>
</tr>
<tr>
<td>DYNOC</td>
<td>2</td>
<td>133053</td>
<td>11</td>
</tr>
<tr>
<td>DYNOC</td>
<td>3</td>
<td>133031</td>
<td>12</td>
</tr>
</tbody>
</table>

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 CPipe 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 bubble sort on 5,000 half-word variables.
The performance of the three modes is shown in Figure 7.6. The fault injection campaign is similar to the arithmetic pipeline case. From static timing analysis reports, we estimated the values of $T_{Max}$ as 6ns, $T_{Err}$ as 1.4ns, $T_{SStall}$ as 0.67ns, and $T_{LSP}$ as 1.7ns. We estimated $T_{Min}$ to be 4.55ns. From the chart, we see that when overclocked, all three application show significant performance gains while operating at DYNOC and MAXOC modes, even when subjected to a severe fault campaign. For applications running for a longer time, the performance benefits achieved through reliable overclocking is huge.

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

Figure 7.6 Execution time for three different applications running on Conjoined Processor in various modes

For our approach, there are no timing overheads on the leading pipeline except for the MUXing-delay. 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 $1.72E5 \text{ mm}^2$, 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.
CHAPTER 8. CONCLUSIONS AND FUTURE WORK

Advances in computing technologies have transformed society and have allowed formation and growth of many communities that were beyond imagination two decades ago. To sustain this growth, advances in microprocessor architectures are critical. The continued shrinking of VLSI circuits has complemented architectural innovations ever since silicon transistors began to revolutionize our world. As device scaling reaches its limits, reliable overclocking has the capability to extend the quest for high performance further, until suitable device alternatives are found. As more and more people get their hands on computers, and more and more day to day activities get automated, it becomes important to have techniques that adapt to the environment, and limit power consumption as and when possible, while not making the end user unhappy because of lack of performance.

This thesis takes into account the wide range of applicability of digital systems, which subject them to diverse demands, in terms of performance, power consumption and dependability, as they perform a plurality of tasks and run in a multitude of operating environments. Considering that these demands are interrelated and need to be addressed cohesively, as improving one metric alone is counterproductive for another, we developed schemes that combines fault tolerance, overclocking and thermal throttling techniques to dynamically enhance computer system performance, reliability and thermal management.

As demonstrated by the successful timing error tolerant overclocking methodology, the current way of estimating the operating frequency for synchronous circuits is far too conservative. The SPRIT$^3$E framework reuses existing superscalar pipeline logic whenever possible, resulting in a modest error detection and recovery logic overhead. However, as silicon feature size decreases, architects have increasingly large silicon real estate available to them. As a result, this trade-off to achieve high performance is acceptable. This work extends the SPRIT$^3$E framework by taking advantage of the margins produced by worst-case design mentality.
In this work, we studied the various factors that limit overclocking. Contamination delay had a serious impact on the extent of overclocking. We looked at ways to manipulate the contamination delay of logic circuits to favorably benefit reliable overclocking. In this thesis, we developed an analysis framework that enables the understanding of several nuances of reliable overclocking. We explored the benefits of reliable overclocking, and looked at ways to make it better. Our results indicate that setting a target error rate of 5% yields significant run-time benefits, while minimizing energy-delay product.

We also presented an initial study of the effects of reliably overclocked systems on on-chip temperatures [93]. In addition, we also analyzed the consequent effects on lifetime reliability of these systems. We considered a reliable overclocking framework and studied its thermal behavior compared to worst-case design. Our work in this dissertation is an initial exploration of dynamic thermal management in reliably overclocked systems. We are continuing this work by developing a powerful thermal management scheme that enhances performance as much as possible while operating well within the thermal limits, guaranteeing an extended system lifetime. The results we have obtained at this juncture are very promising, opening up many different directions for the near future.

Our thermal throttling approach can be extended by adding a dynamic voltage-frequency control technique. Based on the work done in this thesis, we are developing a scheme called DVARFS, which explores a new direction to manage on-chip thermal conditions and improves energy efficiency for processors, especially for battery operated devices [71]. The DVARFS mechanism facilitates reliable overclocking under thermal bounds. Our technique currently relies on an ad-hoc scheme for switching frequencies, a prediction based scheme that makes wise decisions, will reduce the time taken by the clock controller to decide on the best frequency of operation for the current executing application.

This thesis makes significant research contributions as improving performance generation after generation is becoming difficult, as technology scaling in the ultra deep sub-micron region is both expensive, as well as, time consuming. Challenges from process variations can possibly wipe out the benefits of an entire technology node. Overclocking has become mainstream, and several hardware vendors are allowing high-performance enthusiasts to overclock their systems. Reliable overclocking has the capability to enhance the lifetime of a technology node by extending the performance gains achievable with that generation.
The research presented in this thesis can enable hardware vendors to provide overclocking in their mainstream chips. Also, in this expensive semiconductor business, our technique presents smaller players a possibility to meet design goals without resorting to expensive process technology upgrades. We compared speculative reliable overclocking with technology scaling. With every new generation of technology, performance improves by 30%. Our results indicate that reliable overclocking improves performance more than the switch to the next technology node. Even as power and energy consumption increase with reliable overclocking, the energy-delay product metric indicates that reliable overclocking is a good choice for high performance energy conscious systems.

System dependability is a core issue, which is often ignored due to its impact of performance. Most often than not, system designers need to make difficult trade off choice between reliability and high performance. In this dissertation, we proposed 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 [94]. 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, to minimize the area overhead of CPipe architecture, new low-overhead high reliability high performance architectures can be developed. In that direction, our work in this dissertation laid the foundation for developing two techniques, namely Soft Error Mitigation (SEM) and Soft and Timing Error Mitigation (STEM), for protecting combinational logic blocks from soft errors [6]. The 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. The 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.
As part of future work, it will be interesting to see how reliable overclocking applies in the case of superpipelined cores. The important issues of concern are the global propagation of the stall signal on error detection to initiate proper recovery, the precise control of the phase shift between clocks at high frequencies, handling multiple cycle execution in a pipeline stage, and the distribution of the two clock signals to all the pipeline registers in the design.

Another interesting aspect that can be explored is independent overclocking of individual cores in a chip-based multi-core system. Scheduling various tasks based on on-chip temperature, core speed, and ability to overclock can allow multi-cores to enhance single threaded application performance. The work can be extended to real-time systems too, allowing faster energy-efficient execution of various tasks.

Overall, our research looked at the possibility of integrating performance, reliability and energy-efficiency. There have been solutions in the past, which address one or the other issue. An integrated solution becomes attractive, as it presents the best, while solving the primary issues that plague modern computing systems. A unified solution to these problems could render significant cost and performance benefits. As computing machines become ubiquitous, our research helps to make them highly reliable and energy efficient, without compromising on performance. The techniques we introduced and explored in our work holds good prospect for further research.
Bibliography


