2011

Techniques for Enhancing Reliability in VLSI Circuits

Ransford Morel Hyman Jr
University of South Florida, rhyman@cse.usf.edu

Techniques for Enhancing Reliability in VLSI Circuits

by

Ransford Hyman Jr.

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Nagarajan Ranganathan, Ph.D.
Srinivas Katkoori, Ph.D.
Hao Zheng, Ph.D.
Michael Weng, Ph.D.
Brendan Nagle, Ph.D.

Date of Approval:
October 31, 2011

Keywords: Soft Errors, Power, Variations, Architecture, Clustering

DEDICATION

To my family: Parents, Betty and the late Ransford Sr., and sister, Lauranda

ACKNOWLEDGEMENTS

I would like to acknowledge the ultimate creator and supreme being. To my major adviser Dr. Ranganathan for his guidance, from both a school and life's perspective. To my committee members Dr. Katkoori, Dr. Zheng, Dr. Weng, Dr. Brendan Nagle for their assistance throughout my Ph.D. process. To my loving parents Betty and Ransford Sr. I am forever grateful to you for raising me to be the man that I am today. To my sister Lauranda and her husband Thaddius along with their family. To my girlfriend Shakira and Denzel for always giving me encouragement and inspiration to continue. To my mentor Mr. Batson, Dr. Labrador and Dr. Perez, thank you for introducing me to USF. To Dr. Obeng, Mr. Douglass, Dr. Thompson, and the late Dr. Nicholson from BCU. To Dr. Morehouse, the FEF McKnight Doctoral Fellowship Program, National Science Foundation NSF S-STEM DUE #0807023, the NSF-FGLSAMP Bridge to the Doctorate HRD #021675 , the brothers of the Omega Psi Phi Fraternity Inc. I would also like to Thank Deaane Tran Vo and Thomas Bingel along with the East-West Innovation Corporation for allowing me to be a part of their innovative projects. To my family for their words of encouragement. To my research colleagues Koustav, Soumyaroop, Himanshu, Mali, Upuvan, Michael, Yue, Saurabh, Matt, Alfredo, Quenton, Al-Aakhir, Tony, Dorielle, Natasha, Frank, Eric, Brandon, Justin, Luis. To my dearest friends and family Lebron, John, Aidan, James, Renaldo, Tron, Stephen, Brian, Torvis, Rien, D'Jara, Sarah, Lamar, Fred, Abram, Markee, Alvin and Yvette.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Reliability is an important issue in very large scale integration(VLSI) circuits. In the absence of a focus on reliability in the design process, a circuit's functionality can be compromised. Since chips are fabricated in bulk, if reliability issues are diagnosed during the manufacturing of the design, the faulty chips must be tossed, which reduces product yield and increases cost. Being aware of this situation, chip designers attempt to resolve as many issues dealing with reliability on the front-end of the design phase (architecture or system-level modeling) to minimize the cost of errors in the design which increases as the design phase matures. Chip designers have been known to allocate a large amount of resources to reliability of a chip to maintain confidence in their product as well as to reduce the cost due to errors found in the design. The reliability of a design is often degraded by various causes ranging from soft errors, electro-migration, hot carrier injection, negative bias temperature instability (NBTI), crosstalk, power supply noise and variations in the physical design.

Given the continuing scaling down of circuit designs achievable by the advancement in technology, the issues pertaining to reliability have a greater impact within the design. Given this problem along with the demand for high-performance designs, chip designers are faced with objective to design reliable circuits, that are high performance and energy-efficient. This is especially important given the huge growth in mobile battery-operated electronic devices in the market. In prior research, there has been significant contributions to increasing the reliability of VLSI designs, however such techniques are often computationally expensive or power intensive.

In this dissertation, we develop a set of new techniques to generate reliable designs by minimizing soft error, peak power and variation effects. Several techniques at the architectural level to detect soft errors with minimal performance overhead, that make use of data, information, temporal and spatial redundancy are proposed. The techniques are designed in such a way that much of their latency overhead can be hidden by the latency of other functional operations. It is shown that the proposed methodologies can be implemented with negligible or minimal performance overhead hidden by critical path operations in the datapath. In designs with large peak power values, high current spikes cause noise within the power supply creating timing issues in the circuit which affect its functionality. A path clustering algorithm is proposed which attempts to normalize the current draw in the circuit over the circuit's clock period by delaying the start times of certain paths. By reducing the number of paths starting at a time instance, we reduce the amount of current drawn from the power supply is reduced. Experimental results indicate a reduction of up to 72% in peak power values when tested on the ISCAS '85 and OpenCores benchmarks. Variations in VLSI designs come from process, voltage supply, and Temperature (PVT). These variations in the design cause non-ideal behavior at random internal nodes which impacts the timing of the design. A variation aware circuit level design methodology is presented in this dissertation in which the architecture dynamically stretches the clock when the effect of an variation effects are observed within the circuit during computations. While previous research efforts found are directed towards reducing variation effects, this technique offers an alternative approach to adapt dynamically to variation effects. The design technique is shown to increase in timing yield on ITC '99 benchmark circuits by an average of 41% with negligible area overhead.

CHAPTER 1

INTRODUCTION

Reliability is a important issue in digital integrated circuits. Semiconductor corporations employ engineers to verify and validate the reliability of the design at all levels of the design phase. According to [51], seventy percent of the efforts in the production of a design is dedicated to verifying the design's functionality. Reliability engineers focus on both correct functionality and the lifetime of the circuit. The lifetime of the circuit can be hindered by issues such as hot carrier injection, electro-migration, negative bias temperature instability, and electrostatic discharge. Functionality on the other hand can be affected by soft errors, power supply noise, and variations.

The trends in technology scaling have led to an exponential growth in the number of on-chip transistors and significant reductions in the voltage levels of a chip. Due to technology scaling, with the increase in chip densities and clock frequencies, the demand for the design of low power integrated circuits has increased. This trend of increasing chip density and clock frequency has made reliability a major issue for the designers mainly because of the high on-chip electric fields [71, 75]. Several factors such as the demand of portable systems, thermal considerations and environmental concerns have further driven the area of low power design [71]. The power-performance trade-off has only been exacerbated with the inception of parameter variations in nanometer technology. Variation parameters comprise of process deviation due to doping concentration, temperature fluctuations, power supply voltage variations and noise due to coupling.

Some major issues in the reliability of a circuit are keeping the soft error rate (SER) low, minimizing noise on the power supply, and timing the circuit such that it is variation tolerant. A majority of transient faults in modern processors is due to radiation induced soft errors. Soft errors occur when the energetic neutrons coming from space or the alpha particles arising out of packaging materials hit the transistors. Power supply noise manifests from the large instantaneous current demand by logic units. This instantaneous demand causes an IR drop within the power supply and increases the delay at internal nodes which affects timing of the design. Variations caused by process, voltage supply, and temperature can cause frequency and power dissipated to vary from the specified target and hence can result in parametric yield loss.

1.1 Motivation

Soft errors have become a critical issue in nano-scale VLSI circuits. It has been shown that the growth of the soft error rate(SER) increases as the technology size shrinks. The authors in [70] state that the SER is predicted to increase by nine orders of magnitude from 1992 to 2011. In earlier technologies, the effects of soft errors in combinational circuits were too minute to have an effect. As the scaling of transistors progressed, the possibility of soft errors has increased. Due to the power wall and the demand for high performance, the era of multi-core designs was born. Multi-core processors are composed of a number of small processing cores which allow higher throughput through parallelism. Efficient solutions focused on today's architecture are needed. Solutions have been proposed to mitigate soft errors in processors but some these techniques either require a lot of resources or are power intensive. The problem of implementing detection techniques with low overhead is still open. Solutions that are energy efficient that can detect soft errors with minimal overhead will be of significant benefit. As the demand for high performance designs continues to grow and the growth in the smartphone and tablet market, power efficient solutions are needed to prolong the battery life in these devices. There is also a continuous growth in the cloud computing arena which requires fast and correct communication between millions of users. In high performance designs, high current demands from the circuit causes power supply noise known as *Power and Ground Bounce*. Given high current density, the effects of electro-migration, hot carrier injection and electric static discharge are possible which leads to permanent failure of the design. Since the power supply cannot accommodate the current demand of the design at the time necessary, an increase in delay is seen which has an effect on the timing of the circuit. To mitigate this high current demand, designers have allocated more I/O pins to power and ground rails. There have been techniques proposed in research that have approached this problem by taking advantage of the clock skew between registers [26, 32, 86], however in this work, the peak power reduction is achieved through path clustering.

As shown in [28], the manufacturing cost increases as the technology scaling decreases. With demand for high performance, timing of the circuits should be as aggressive as functionally possible. Many designs are timed conservatively which has an effect on performance and energy efficiency. Techniques at the layout level have been proposed but are complex and can be computationally expensive to generate a solution. There have been techniques that suggest the use of multi-voltage domains but these can be power hungry.

1.2 Contributions of This Work

- A series of techniques are proposed that detect soft errors in combinational design in multi-core processors. The technique focuses on combinational logic operations in the pipeline. Soft errors are detected using data value, information-based, temporal and spatial redundancy. The multi-core architecture is used to help minimize perfor-

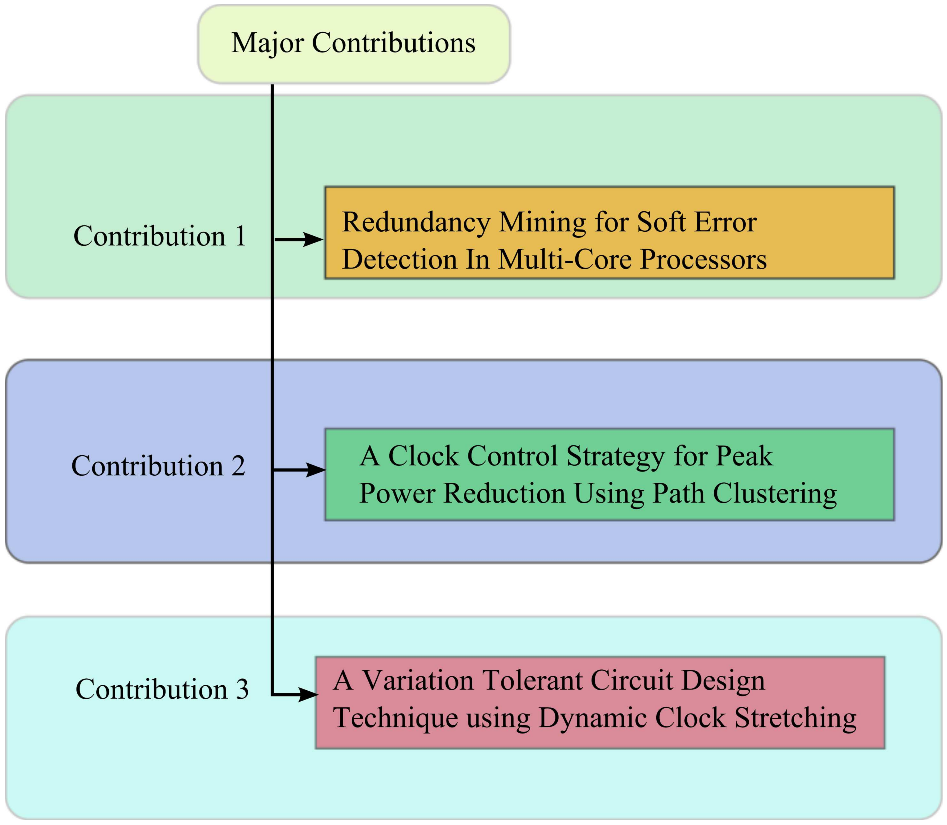

Major Contributions
Contribution 1
Redundancy Mining for Soft Error Detection In Multi-Core Processors
Contribution 2
A Clock Control Strategy for Peak Power Reduction Using Path Clustering
Contribution 3
A Variation Tolerant Circuit Design Technique using Dynamic Clock Stretching

Figure 1.1. List of Research Contributions

mance overhead in the spatial redundancy. Other techniques take advantage of the resources currently available during computation.

- A path clustering algorithm is proposed to minimize peak power values in combinational circuits. The focus of the peak power reduction methodology is geared towards combinational logic between registers. The objective is to cluster those paths that possess slack relative to its path delay and the circuit's critical path delay. By reducing the number of paths starting at a time instance, we reduce the amount of current drawn from the power supply is reduced.

- A variation-tolerant technique is proposed based on using dynamic clock stretching. A variation aware circuit level design methodology is presented in which the architecture dynamically stretches the clock when the effect of an variation effects are observed within the circuit during computations. A dynamic clocking strategy is used to provide additional delay to the wire driving the adjacent register circuit if conflicting values are seen at the *critical interconnect transition* which is discussed later.

1.3 Outline of Dissertation

In Chapter 2, a discussion on background information and the related work pertaining to this research is given. In Chapter 3, redundancy techniques used to detect soft-errors in multi-core designs are presented. The proposed architecture and algorithm to detect soft-errors with low performance overhead is illustrated in this chapter. In Chapter 4, a discussion is given on peak power minimization techniques through the use of path clustering. Circuit-level techniques are implemented on the combinational logic to minimize the amount of current drawn from the power supply at a given time instance. In Chapter 5, an architectural-level solution to dynamically perform clock stretching to mitigate the effects of process variations is shown. By inserting a small block of logic on all near critical paths, the timing yield can be increased for performance optimized chips that have affected by variations. In Chapter 6, conclusions are given for the contributions presented in this dissertation.

CHAPTER 2

BACKGROUND

2.1 Transistor Theory

Transistors are the building blocks of digital circuits. There are two types of transistors that can be created in Complementary Metal Oxide Semiconductor (CMOS) technology. These types are called p-type and n-type. N-type silicon is created by doping a silicon body with large concentrations of phosphorus or arsenic such that the majority carriers are electrons. P-type silicon can be created by doping silicon with large concentrations of boron such that the majority carriers are holes (places where an electron can reside). p-type transistors are designed by doping concentrations of p-type silicon within a n-type substrate. The reverse method generates an n-type transistor. These two devices are called Metal-Oxide Semiconducting Field-Effect Transistors (MOSFET). An illustration of a n-type transistor is shown in Figure 2.1 Lead wires are located on the gate, source and drain terminals. The drain of a n-type transistor is connected to ground and the source terminal is connected to the output node. As the voltage of the gate increases with respect to the drain terminal(V_{gd}), an electric field is created between the gate and the majority carriers are repelled in the substrate which creates an empty region called the depletion region. As the gate voltage increases and reaches the threshold voltage, V_t, it starts to attract electrons under the gate. The voltage between the source and the drain terminals(V_{ds}) creates a lateral electric field which causes conduction of current to flow from the source to the drain. The

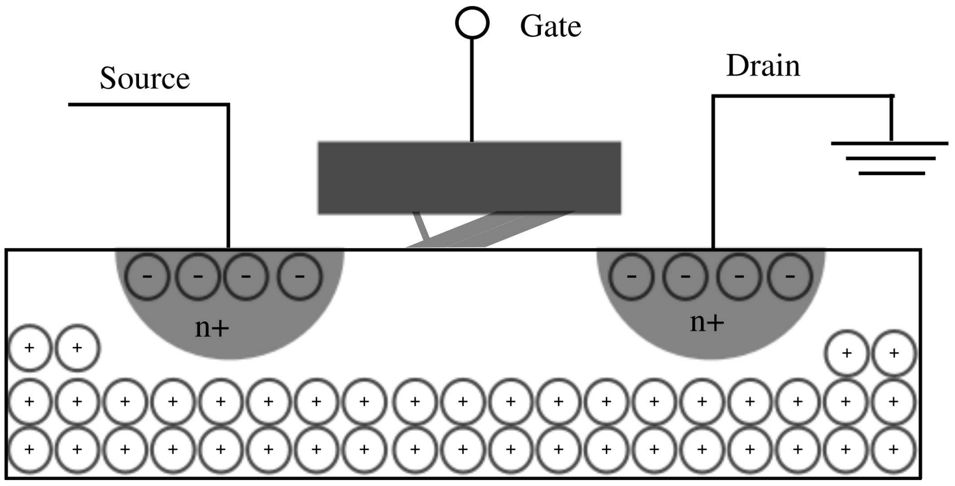


Figure 2.1. Two Dimensional View of NMOS Transistor

same technique applies for the p-type transistor except the carriers are holes instead of electrons.

The transistor in CMOS designs acts as a switch to charge or discharge the load capacitance on the output node. This capacitive load is made up of the source and drain terminal capacitors near the output as well as the wire capacitances driving the connecting gates(or pins). The CMOS transistor design is a widely chosen method due to that it only consumes power when it switches states (ideally) and it also has large noise margins which allows for well defined logic ’1’ and logic ’0’ states.

2.2 Soft Errors

Soft errors were initially discovered within dynamic memories by the authors in [40]. Soft errors are caused by random collisions from alpha-particles found in traces of uranium, high energy neutron particles, and low-energy cosmic neutron interactions with Boron which is a doping element used in IC manufacturing. These particles can be found in large volumes in extra-terrestrial environments and the packaging materials used to enclose the chip. When these particles strike the active region of a transistor, a group of free

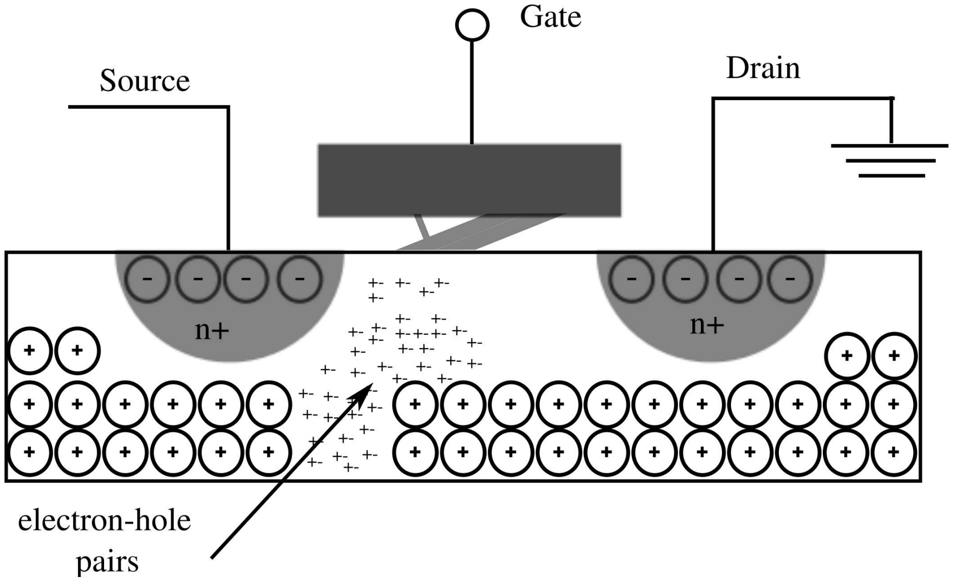


Figure 2.2. Effects of an High-Energy Particle Strike

electron-pairs are formed by the collision of particles. If the excess majority carriers of the transistor are collected into the drain terminal of the transistor this may offset the voltage reading on the node (called a single-event transient (SET)), causing an incorrect value at the output. Once this single-event transient gets stored on a memory element, it becomes a single-event upset (SEU). The critical charge, Q_{crit}, defines the amount of charge needed to cause a disturbance in the state of the output. As our devices continue to shrink, this critical charge decreases making circuits more susceptible to soft errors. There are multiple metrics used in research to measure soft errors. The Soft Error Rate(SER) calculates the number soft errors occurring within a time measurement given the density of high-energy particles. The architectural vulnerability factor(AVF) measures the probability that a soft error causes an error in the architectural state. Failures in Time(FIT) defines the number of failures every 10^9 hours. Mean Time Between Failure (MTBF) or Mean Time to Failure (MTTF) measures the amount of time between consecutive errors on average.

2.3 Pipeline Design

A datapath in computer architecture is the process flow of executing instructions. The main operations that must be performed in any datapath are instruction fetch, instruction decode, instruction issue, execute, write back, and commit. These operations may be executed in a single-cycle, multi-cycle or pipeline implementation. The pipeline implementation is the most common approach in today's architectures due to its increase in information throughput. A pipeline dataflow is shown in Figure 2.3. Instructions are fetched from the instruction memory cache in the fetch stage. Instructions represented as 32-bits(known as a word) or 64-bits(double word) are translated so that the architecture knows what actions are necessary in the decode stage of the pipeline. The decode stage determines the type of instruction, the location of the registers for the source operands, and what register to store the result of the instruction. The issue stage sends the source operands to the functional unit needed to perform the operation. Source operands are sent to functional units as soon as the source operands are ready(all dependencies have been resolved). Depending on the architecture, this may be done in order or out of order. Out of order issue makes better utilization of the issue slots of the functional units and is commonly used in dynamic scheduling techniques as well as Simultaneous Multi-Threading (SMT). The arithmetic operations that are necessary are performed in the execute stage. Load and store instructions use the execute stage of the pipeline to calculate the memory addresses. The results are placed in a re-order buffer(ROB) where data dependent operands can be updated in the issue slots of the functional units. The ROB is used in the case where hardware speculation is implemented. Hardware speculation is defined as predicting a conditional branch to enhance the performance of the processor. If the branch has been mis-predicted, the ROB is flushed and the instructions are re-executed with the correct branch value. Upon correct speculation, the values stored in the ROB are written to their respected destination registers. These actions are performed in the commit stage of the pipeline.

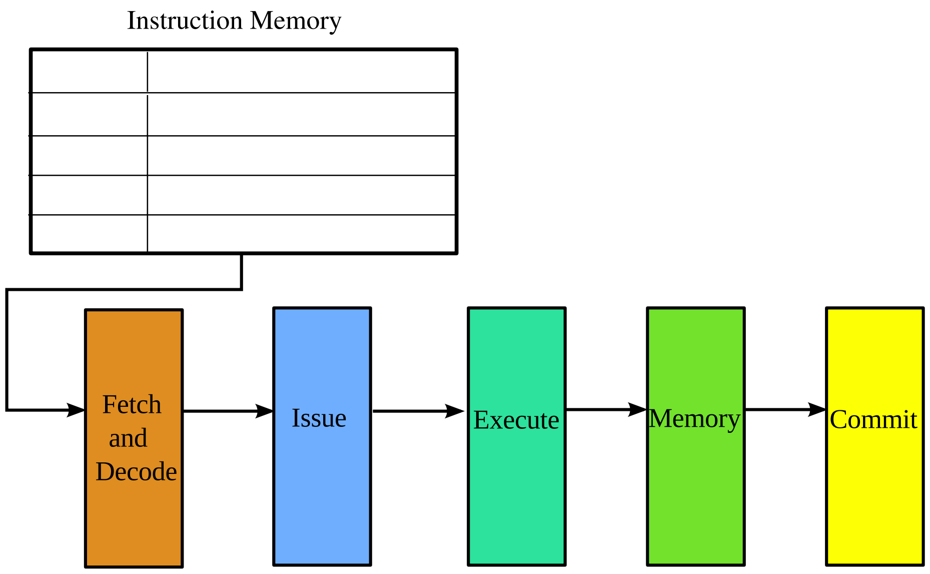


Figure 2.3. Pipeline Architecture Design

2.4 Processor Core Model

Earlier designs of the processor consisted of a large single core model. Within this model, there was a Central Processing Unit (CPU) and the Cache Memory as illustrated in Figure 2.4. As shown, These two components reside on the same die package. Performance optimizations for processors were implemented by increasing the clock of the processor so that operations could be executed faster. Architectural-level techniques such as instruction-level parallelism (ILP) and thread-level parallelism (TLP) were also incorporated to optimize performance from the software-level. Threads are processes that maintain their own instructions, data and stack such that multiple processes may execute in parallel. Threads allow for multiple processors to share the execution units of the processor so that multiple processes are able to progress and complete more efficiently. Processors are designed with the hardware resources to support a defined amount of threads simultaneously.

Increasing clock speed reached a limit when the power consumption of the chip generated large amounts of heat dissipation. Large cooling costs were incurred since the lifetime

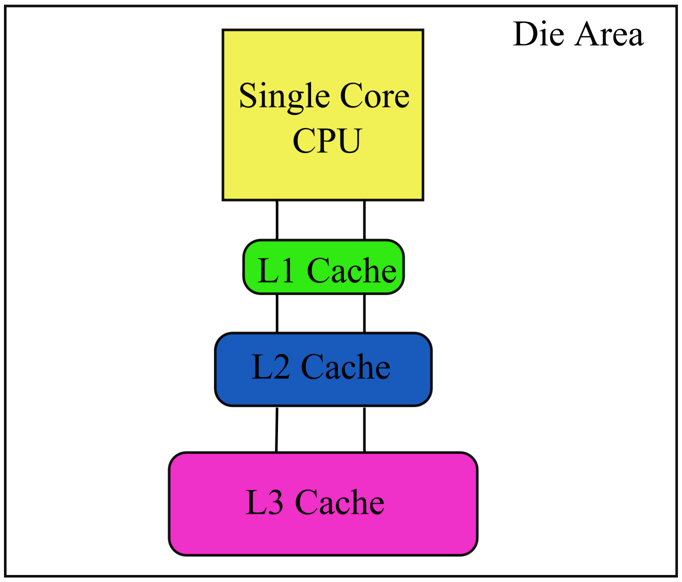


Figure 2.4. Single Core Architecture Model

and performance of the processor are affected by the temperature. This brought about a new era of processor designs called multi-core processors. Multi-core processors have multiple smaller CPU cores on a single die which are interconnected through some shared cache memory. With multiple CPU cores allowed to perform computations simultaneously, multi-core designs can achieve high performance at lower clock speeds which reduces the power consumption and heat dissipation. We illustrate a quad-core architecture design in Figure 2.5 where each processor core maintains private L1 and L2 cache and the L3 cache is shared amongst all of the cores. In multi-core designs today, a processor may have up to eight-cores and each core has the ability to execute at least two hardware threads.

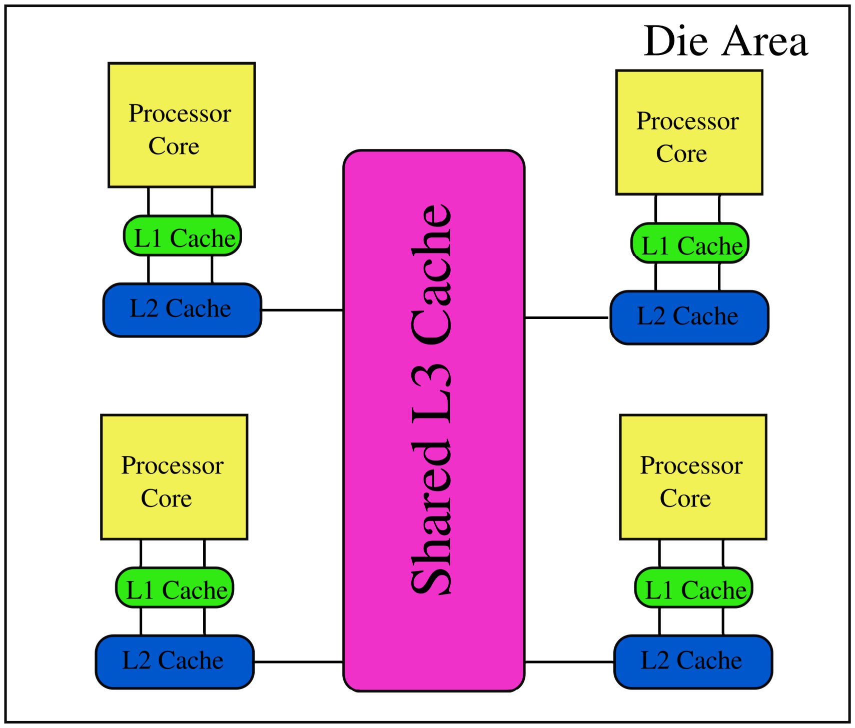


Figure 2.5. Quad-Core Architecture Model

2.5 Clocking in Digital Logic Circuits

Clocking is done in digital design to synchronize the combinational logic to assure that stable signals are read/written to sequential(memory) elements. Timing of a circuit is defined from the transistor level, where the delay of a node is defined by the time to charge/discharge a capacitance load. A clock is used to notate that at the end of its period, all outputs values should have been calculated and stable. Clocking is performed on sequential devices such as latches and flip-flops. A latch is defined to be a sequential structure that becomes transparent only when the level of the clock transitions. A flip-flop (also called a register) on the other hand reads values on the edges of the clock. Due to these characteristics, latches are commonly called level-sensitive devices and flip-flops are called edge-triggered devices. Once these structures are in their non-transparent mode, the data value is stored until a new data value has been read. To ensure correctness, sequential devices have a setup time and hold time constraint. The setup time notates the time the data signal must be stable before the clock signal transitions. The hold time defines the time the data signal must be stable after the clock transition has occurred. Violation of these constraints causes these sequential devices to store unpredictable values which in turn has an effect on the reliability of the design.

2.6 Power Definitions in CMOS Designs

Power consumption in CMOS designs contributes to both the speed and energy consumption of a circuit. Power dissipation is directly proportional to the power supply voltage and capacitive load of a particular node. Static power consumption correlates to the power dissipation when no circuit activity is being done. Static power consumption is contributed to second-order effects such as sub-threshold leakage and tunneling. Dynamic power is the largest factor of the total power consumption and directly corresponds to the charging and discharging of capacitors. When a capacitor is being charged, energy is transferred from the power supply to the capacitor. The dynamic power consumed is given by

$$P_{dynamic} = \alpha C_L V_{dd}^2 f \tag{2.1}$$

where α equals the activity factor, C_L equals the load capacitance and f equals the clock frequency. This short-circuit power comes from the direct path from the power supply (V_{dd}) terminal to ground (V_{ss}) terminal. Short circuit power is dependent on the input and output transition times of the gate. If the input transition time is larger than the output transition time, the short-circuit power is large due to the long period of time of a direct path from V_{dd} to V_{ss}. The Peak power is the maximum amount of power drawn given any time instance.

2.7 Power Supply Noise in VLSI Designs

Power is supplied to VLSI designs by the power supply unit. The power supply is connected to a chip design through bonding wires connected to the pads on the die. In synchronized designs, all the combinational begins on the edge transitions of the clock. This causes a large instantaneous current demand on the power supply. There are two sources of noise on the power supply: *IR noise* and *inductive noise*. IR noise is caused by the resistive parasitics within the interconnect wires causing a voltage drop over the signal. The farther a gate is away from the power source, the more voltage drop can be seen at its supply terminal. This lower voltage supply causes a delay in the charging of the capacitors which decreases the performance of the design. Inductive noise or $L \times \frac{di}{dt}$ noise is caused by the inductance within the bonding wires used to connect the power supply to the pins on the chip. When the chip requests a large instantaneous current demand on the power supply, this sharp change in current ($\frac{di}{dt}$) causes a drop in the supply voltage due to the opposing voltage created by the inductance. Given that there is also capacitance in the wire, this creates a resonator. If the transfer of voltage between the inductor and capacitor oscillates at the resonant frequency, this causes a resonance on the voltage supply. This resonant behavior can be seen in Figure 2.6 where the voltage supply resonates before returning to the nominal voltage level. These types of supply noise causes functionality problems due to the reduction in noise margins as well as performance issues from the delay incurred by the lower voltage supply. Also the large current draw creates a current spike causing large peak power values. High current demands not only effect timing, but large current densities can lead to permanent failure due to electro-migration, hot electron injection and electrostatic discharge(ESD).

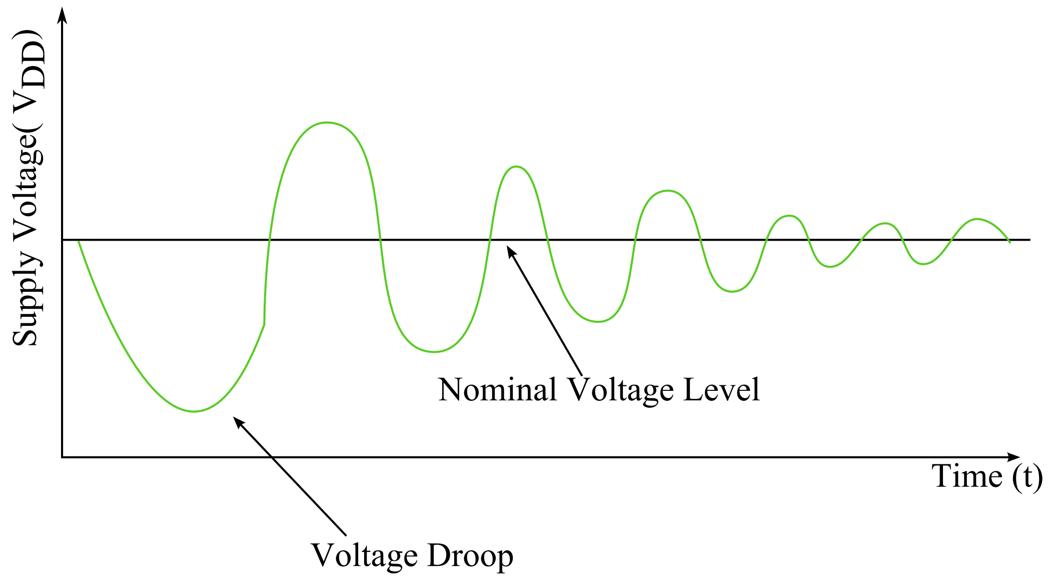


Figure 2.6. Power Supply Noise Due to Inductive Coupling

2.8 Variations

Variations within a design are caused by the factors imposed by the manufacturing process, temperature and voltage supply. The variations in transistor length, and doping concentrations generates variations in timing of the transistors. These variations also have an effect on the threshold voltage which determines when the transistor starts conducting current. These variations due to manufacturing causes issues in the timing the circuit. Designers account for this when designing the circuit, but too much conservativeness can cause for the design to be clocked slower than necessary. Temperature has an effect on the mobility of carriers which causes a delay in overall timing of the design. The voltage supply varies due to current demand which also has an effect on the timing of the design.

Two types of variations are Die-to-Die(D2D) and Within-Die (WID) variations. Die-to-Die, or inter-die variations, are the type of variations that have an effect on parameters across die differently, but do not have an effect on the parameters within the die. Within-Die, or intra-die variations, affect the parameters within the die differently. A random WID variation fluctuates randomly and independently from device to device. A systematic WID parameter variation results from a repeatable and governing principle where the device to device correlation is empirically determined as a function of the distance between devices.

Variations are modeled in a probabilistic manner, most often using the Gaussian distribution. Chips designed today are designed to meet 6σ standards such that 99.99966% of the yield meets the constraints set by the designers. Chips that do not meet the timing constraints have to be tossed which incurs a cost in overall yield. Designers perform strenuous testing in CAD modeling to hopefully account for any issues that may arise in manufacturing.

2.9 Related Work on Soft Errors

Several techniques have been proposed for reduction of soft error rates in caches. These include techniques like the use of error correction codes (ECC), parity bits, bit interleaving and small value duplication [1].

Soft errors can also occur in any internal node of a combinational logic and subsequently propagate to and be captured in a latch. Technology trends like smaller feature sizes, lower voltage levels, higher operating frequency and reduced logic depth, are projected to increase the soft-error rate (SER) in combinational logic [70]. Thus, the vulnerability of both combinational logic structures and latches have made modern processor pipelines significantly susceptible to soft errors. Soft errors may cause corruption in data which may lead to incorrect addressing, faulty instruction execution or generation of false exceptions. Various memory and latch elements within the processor pipelines, for example the issue queues in [19], have been individually protected against soft errors. In one of the earliest works, complete processor pipelines were duplicated [78] for detecting transient faults. In [88] ”valid but idle” instructions were exploited for soft error reduction. The

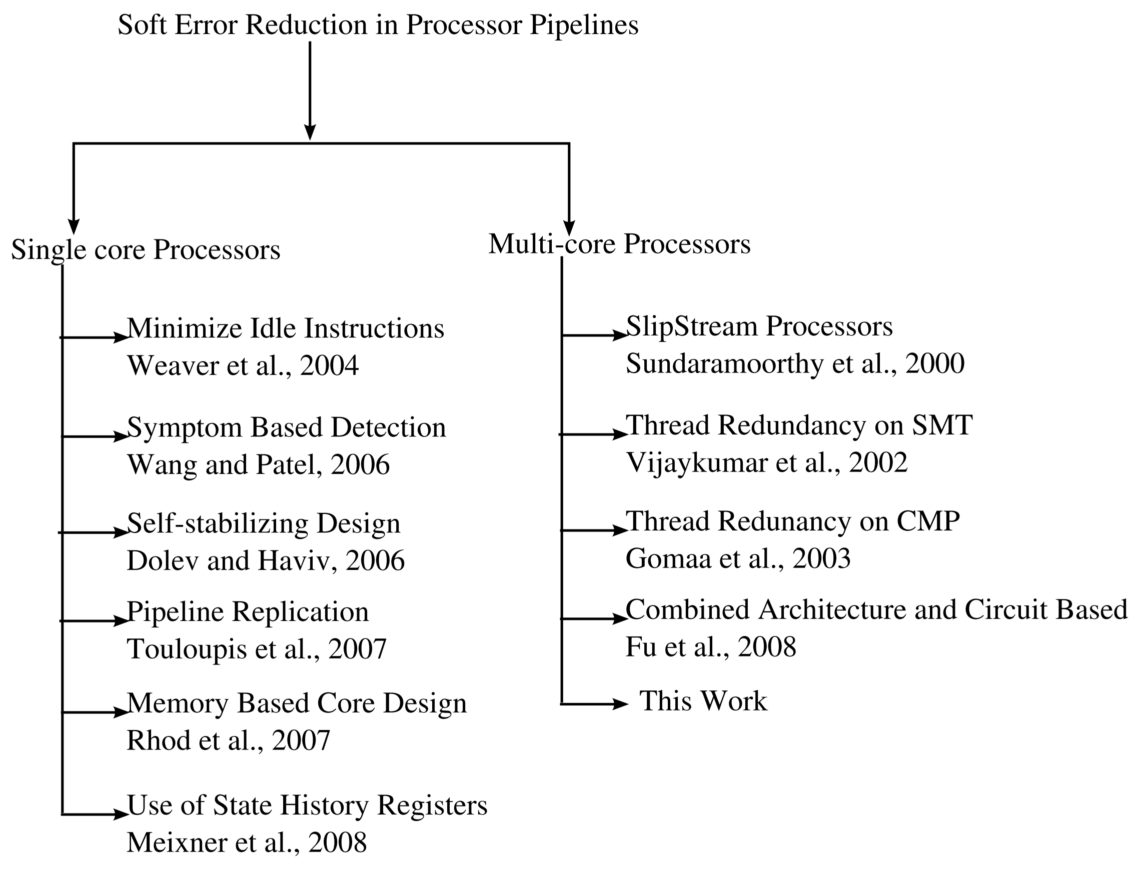


Figure 2.7. Taxonomy of Soft Error Research

authors in [14] propose the design of a self-stabilizing processor for improving soft error vulnerability. The core design in [41] uses *state history signatures*(SHS) on memory and functional units for error detection. In [58], the authors have introduced memory based core design that uses the FRAM technology for immunity to soft errors. However, this architecture has a high memory cost and has limited functionality. In [87], the authors have used exceptions and incorrect control flow as ”symptoms” for detection. However, since these symptoms are not known until the execution phase of the pipeline, it allows false positives to occur which can lead to huge performance costs. In [25], compiler directed instruction duplication has been proposed. The approach however leads to large increase in code size and inefficient resource utilization. The high transistor counts afforded by technology scaling are making chip multi-processors (CMPs)an attractive option to overcome the per-

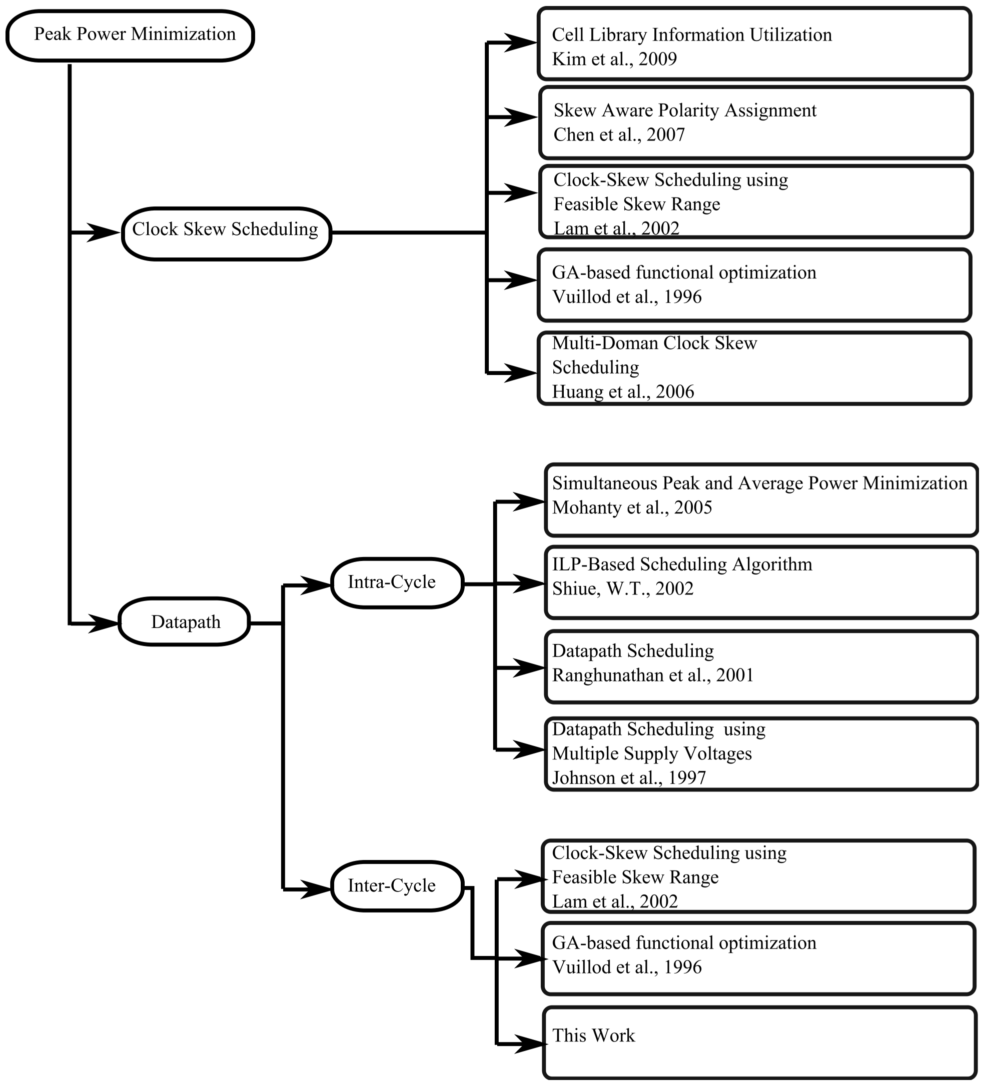

Peak Power Minimization
Clock Skew Scheduling
Cell Library Information Utilization Kim et al., 2009
Skew Aware Polarity Assignment Chen et al., 2007
Clock-Skew Scheduling using Feasible Skew Range Lam et al., 2002
GA-based functional optimization Vuillod et al., 1996
Multi-Doman Clock Skew Scheduling Huang et al., 2006
Datapath
Intra-Cycle
Simultaneous Peak and Average Power Minimization Mohanty et al., 2005
ILP-Based Scheduling Algorithm Shiue, W.T., 2002
Datapath Scheduling Ranghunathan et al., 2001
Datapath Scheduling using Multiple Supply Voltages Johnson et al., 1997
Inter-Cycle
Clock-Skew Scheduling using Feasible Skew Range Lam et al., 2002
GA-based functional optimization Vuillod et al., 1996
This Work

Figure 2.8. Taxonomy on Peak Power Research

formance and power wall provided by superscalar machines. CMPs are building blocks for server-class machines for which reliability is a key concern. The problem of soft errors is further exacerbated in a large multiprocessor server which requires even lower failure rates for the individual microprocessors. The authors in [18, 63] studies the tradeoff of the soft error rate, power and performance metrics for a reliable multi-processor. A two-way CMP enables on-chip fault detection using lockstepping in which the same computation is performed on cycle-by-cycle basis. Error detection and correction has been exploited using simultaneous redundant threads (SRT) of execution, on a SMT processor, by using a leading and trailing thread separated by a slack [59]. The trailing thread uses load memory values and branch outcomes of the leading thread to avoid memory latencies and mis-predicted computations. Memory corruption is avoided by only committing stores after comparison with the trailing threads. Register values may be committed before or after comparison of the trailing threads [84]. The motivation behind scheduling and checking independent threads as against lockstepping is that lockstepping uses hardware structures less efficiently than SRT. Both copies of a computation in lockstepping are forced to waste resources on mis-speculation and cache misses. Chip level redundancy (CRT) applies the SRT scheme to CMPs for error detection. Unlike SMT processors, reliable multi-core processors are required to execute redundant threads concurrently on *different* chips. Error detection is performed by checking by comparing the results from the two threads. The primary concern for CRT have been reducing the latency associated with checking of the results from the two threads which is performed using high-speed hardware structures for inter-chip communication. Although, techniques have been proposed like checking threads at the tail of dependence chains and judicious chaining of masking instructions, the performance overhead of such schemes is still significant.

2.10 Related Work on Peak Power

Significant amount of research has been reported in the literature on peak current and peak power reduction as well as clock skew scheduling. In [26, 85], a multi-domain clock is used where a single clock source is used to create multiple phases of the clock within the original clock period. In [85], a 0-1 ILP is used to solve the multi-domain clock scheduling problem. In [26], They attempt to reduce the runtime of the ILP formulation by partitioning zones for the registers within a bounded constraint and solving each zone using the 0-1 ILP algorithm. In [57], a graph-based algorithm approach is taken such that for a user-specified amount of clock domains, an optimal phase shift for each clock domain is generated. In [9, 31, 32, 86], a single clock and edge-triggered flip flops are used to account for the worst-case scenario. In [86], a genetic algorithm was used to find the optimal clocking schedule to minimize peak current. In [32], peak current is minimized by shifting the arrival times of the flip-flops based on there respected clock skews between adjacent flip-flops. [31], extracts the current profile from the Cell Library and uses simulated annealing algorithm is used to find the optimal peak current minimization. In [9], the clock signal is inverted on half of all the flip-flop elements such that half of the flip-flops will become active the falling edge of the clock and the other half switches on the rising edge. In [89] he issues rising and falling edges to various IP cores in a SoC-designs.

Several researchers have studied the reduction of average and peak power at the behavioral and logic synthesis levels [44]. The use of multiple supply voltages for power reduction is well researched and several works have appeared in the literature [7,30,34,38,45,56]. In multiple supply voltage scheme, the functional units can be operated at different supply voltages. The energy savings in this scheme is often accompanied by degradation in performance because of the increase in the critical path delay. The degradation in performance can be compensated using dynamic frequency clocking (DFC) [42, 45], multi-cycling and chaining [52], and variable latency components [3, 4, 54]. In multi-cycling, an operation is scheduled for more than a single control step and in addition, each control step is of equal length. On the other hand, in the case of DFC, an operation is scheduled in one unique control step, but all the control steps of a schedule may not be of equal length and also, the clock frequency may be changed on-the-fly.

In the works reported in [39, 62], the peak power reduction is achieved through simultaneous assignment and scheduling. The authors demonstrate the use of power minimization at one level to achieve optimization at another level. Specifically, the simultaneous use of

SPICE and behavioral synthesis tools is demonstrated. The authors use genetic algorithms for optimization of average and peak power. In [67], ILP based scheduling and modified force directed scheduling have been proposed to minimize peak power under latency constraints. The ILP formulation considers multi-cycling and pipelining and single supply voltage. ILP based models to minimize peak power and peak area have been proposed in [68] for latency constraint scheduling. The authors also introduced resource binding to minimize the amount of switching at the input of functional units. The ILP based scheduler allows the minimization of multi-cost objectives using the user defined weighting factors. In [66, 69], a time constrained scheduling algorithm for real time systems using a modified ILP model that minimizes both peak power and number of resources, is described. The authors in [55] propose the use of data monitor operations for simultaneous peak power reduction and peak power differential. The authors advocate the need for the careful choice of the transient power metric for the minimization of area and performance overheads. In [46, 47], a heuristic based scheme is proposed that minimizes peak power, peak power differential, average power, and energy altogether. In [43, 48], the authors propose ILP based datapath scheduling schemes for peak power minimization under resource constraints. The scheduling algorithms handle multiple supply voltages, dynamic frequency clocking and multi-cycling.

2.11 Related Work on Variations

In this context, several researchers have proposed the use of statistical timing analysis and statistical optimization mechanism to meet timing in the presence of variations without significant overhead [10, 11, 23, 35, 36]. The variation aware optimization methodologies use stochastic or fuzzy methodology to minimize the impact of uncertainty due to process variations on performance, power and other design overheads. Statistical timing analysis

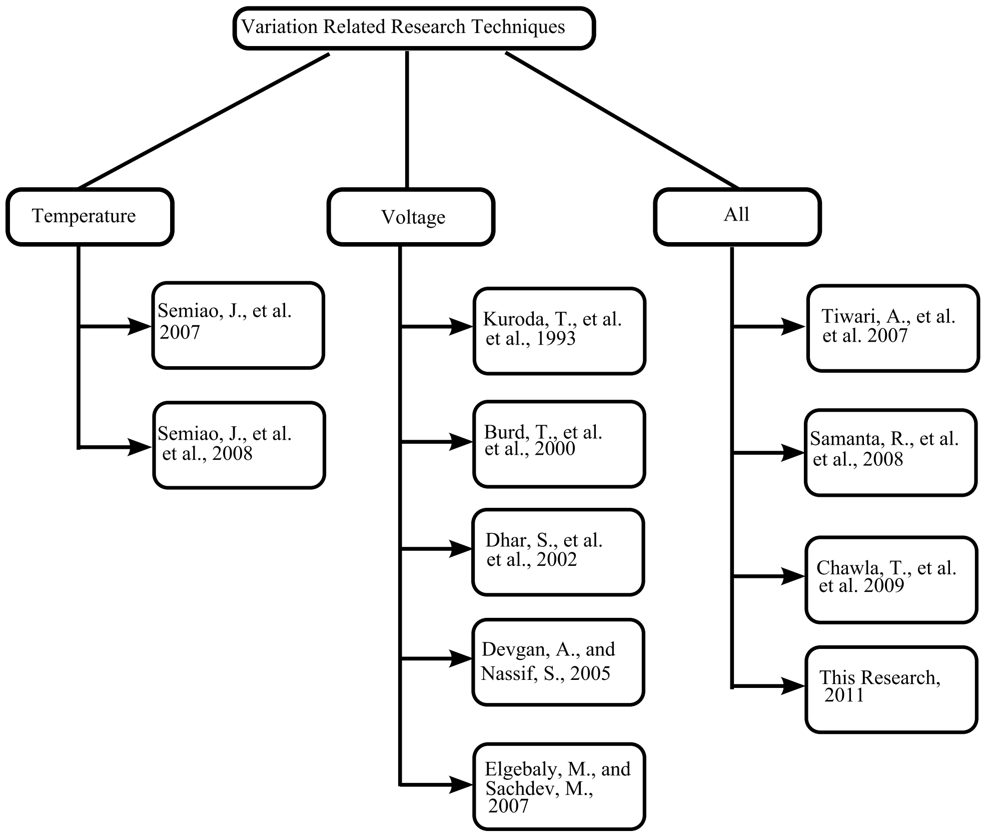

Variation Related Research Techniques
Temperature
Voltage
All
Semiao, J., et al. 2007
Semiao, J., et al. et al., 2008
Kuroda, T., et al. et al., 1993
Burd, T., et al. et al., 2000
Dhar, S., et al. et al., 2002
Devgan, A., and Nassif, S., 2005
Elgebaly, M., and Sachdev, M., 2007
Tiwari, A., et al. et al. 2007
Samanta, R., et al. et al., 2008
Chawla, T., et al. et al. 2009
This Research, 2011

Figure 2.9. Taxonomy of Research on Variations

(SSTA) was investigated in, [10,11], where continuous distributions are propagated instead of deterministic values to find closed form expressions for performance in presence of variations. Variation aware solutions have also been developed for circuit optimization problems like gate sizing, buffer insertion and incremental placement [23,35,36]. The main objective of these works has been to improve yield, without compromising on performance, power and area. The variation aware optimization techniques have shown to improve design overheads without loss in parametric yield. However, the statistical optimization methods still over consume resources irrespective of whether the circuit is affected by variations or not. Hence, to facilitate more aggressive power-performance-yield tradeoff improvement, dynamic schemes to detect and correct the uncertainty due to process variations are becoming necessary.

Further, the authors in [60], proposed a novel design paradigm which achieves robustness with respect to timing failure by using the concept of critical path isolation. The methodology isolates critical paths by making them predictable and rare under parametric variations. The top critical paths, which can fail in single cycle operation, are predicted ahead of time and are avoided by providing two cycle operations. The methodology which works well for special circuits with rare critical paths, however has severe timing penalty on benchmark designs.

One of the popular methods to dynamically combat process variation's impact on design has been to use adaptive voltage scaling (AVS) [13, 15, 16]. The voltage scaling systems track the actual silicon behavior with an on-chip detection circuit and scales voltage in small increments to meet performance without high overheads in the presence of process variations. In [6, 76], the critical path of the system was duplicated to form a ring oscillator and the actual performance requirement of the circuit is co-related to the speed of the oscillator and appropriate voltage scaling is performed. However, in the nanometer era, it is not feasible to use a single reference for a critical path and the variations' range can make the close to critical delay paths critical on actual implementation. Recently the authors in [15], proposed an AVS system which can emulate critical paths with different characteristics. With increasing amount of on-chip variations and spatial correlation the methodology can have severe discrepancies. In a bid to reduce such margin and remove the dependency of feedback mechanism on a single path, a novel on-chip timing checker was proposed in [16] to test a set of potential critical paths. The method uses a shadow latch with a delayed clock to capture data in all potential critical paths. An error signal is generated if the value in original and shadow latch is different due to a timing violation caused by process variations. The methodology however aims at correcting (not preventing) errors caused by aggressive dynamic voltage scaling. To guarantee high timing yield and low overheads in the presence of variations, the ultimate solution is to dynamically alter the clock signal frequency. The authors in [64, 65], proposed a technique to control and adjust clock phase dynamically in the presence of variations. The methodology focused on the design of a dynamic delay buffer cell that senses voltage and temperature variations and alters clock phase proportionately. However, it is not generic to all types of variations and does not include spatial correlation between the delay buffer and the gates in the critical path. Also, the methodology is not input data dependent and hence changes the clock capture trigger in more than required number of instances. [8] tries to mitigate variation effects by unbalancing the first stage of the combinational path to reduce the minimum clock pulse width variation . This technique was done from the transistor level and could be time consuming when considering Very-large scale integration. Both [61] and [77] use techniques from the micro-architecture level to reduce timing effects from process variations. [61] uses dynamic-speed boosting to speed up computation when necessary to reduce the timing effects of variations. This increase in speed caused by larger V_{dd} values can cause a large amount of average power overhead. In [77], clock skew between pipelines are computed and donor stages are inserted where slack is available so that slower stages and borrow time from faster stages. Since the definition of the clock period is determined by the average stage delay, the timing of the circuit may be too slow for certain designer's specifications.

CHAPTER 3

REDUNDANCY MINING TECHNIQUES FOR SOFT ERROR REDUCTION IN MULTI-CORE PROCESSORS

The trends in technology scaling have led to exponential growth in the number of on-chip transistors and significant reductions in the voltage levels of a chip. These trends for improving performance and power have made modern processors increasingly susceptible to transient faults. A majority of soft errors in modern processors are due to radiation induced transient faults. Radiation causes ’transient faults’ or ’single-event transients’ to occur in logic which, once propagated and latched, become full cycle errors or soft errors. If radiation hits memory elements, this is usually called ’single-event upset’ or ’soft error’ as it can further propagate as a full cycle error. Soft errors could occur when the energetic neutrons coming from space or the alpha particles arising out of packaging materials hit the transistors. With the shrinking of device geometries, the critical charge (Q_{crit}) required for the occurrence of soft errors, decreases. However, as the active silicon area of the cells also decrease due to scaling, the probability of radiation strike also decreases. Thus, the vulnerability of individual transistors due to cosmic ray strikes remains almost constant [21]. However, the decreasing voltage levels and the exponentially increasing transistor counts have caused the overall chip susceptibility to increase significantly.

In this chapter, it is shown that lock stepping can be used quite effectively if sufficient redundancy can be mined within each core’s boundary. Schemes are presented that utilize the property of temporal, data value, and information redundancies in programs for detection of soft errors. The use of latency slack cycles (LSC) for error detection using temporal redundancy and the mining of value based redundancy in a processor core by utilizing the small data value width of the operands are introduced in this chapter. Information redundancy is exploited by encoding the operand values with residue codes when possible. Furthermore, it is shown that the latency overhead associated with inter-processor communication can be significantly reduced when a clustered core micro-architecture is used. Experimental results indicate that the proposed schemes, on the average, can detect and correct soft errors in multi-core systems with negligible area and performance overheads.

The rest of the chapter is organized as follows: The architecture of a clustered core multi-processor for soft error detection is described in Section 3.1. In Section 3.2, it is discussed how temporal redundancy can be exploited for error detection. Static and dynamic techniques for exploiting temporal redundancy for soft error detection are presented. Section 3.3 describes how data value based redundancy can be employed using small value replication. Section 3.4 discusses error detection using residue code by utilizing information redundancy. The overall architecture for error detection using the proposed approach in a typical multi-core processor as well the algorithm implementation is shown in Section 3.5. In section 3.6, experimental results are given and finally section 3.7 gives simulation results and compares the experimented work with prior works.

3.1 Clustered Multi-Core Architecture for Error Detection

In multi-core processors, multiple threads can be mapped to multiple cores and executed concurrently to increase the system throughput. These threads can be separate independent applications or independent pieces of the same thread from a given application. Redundant execution of threads and comparing the results of their multiple executions can be effective towards reduction of soft errors. However, two problems seriously limit this approach.

First, the inter-processor communication latency can be significant. The wire delays due to inter-processor communication can impose significant latency overhead which cannot be hidden in the typical complete-to-commit times, if all register and memory accesses have to be checked. Secondly, due to the non-deterministic ordering of communication events, threaded applications can produce outputs in different orders and of different values when given the same inputs over consecutive runs [53]. This phenomenon can make error detection by dedicated execution of the original and the duplicate threads on different processors in a CMP ineffective or not beneficial. On the other hand, forcing deterministic execution for replication by lock-stepping will have a high performance cost.

A clustered multiprocessor architecture to reduce the communication latency due to inter-processor communication is used in this work. Further, it is shown that by mining of available temporal and data value redundancy and by exploiting the information redundancy, the inter-processor communication required in a fault-tolerant CMP can be brought down significantly. In this section, a discussion on the clustered core processor architecture for low overhead error detection in a CMP. The technique to mine available redundancy is described in the following sections.

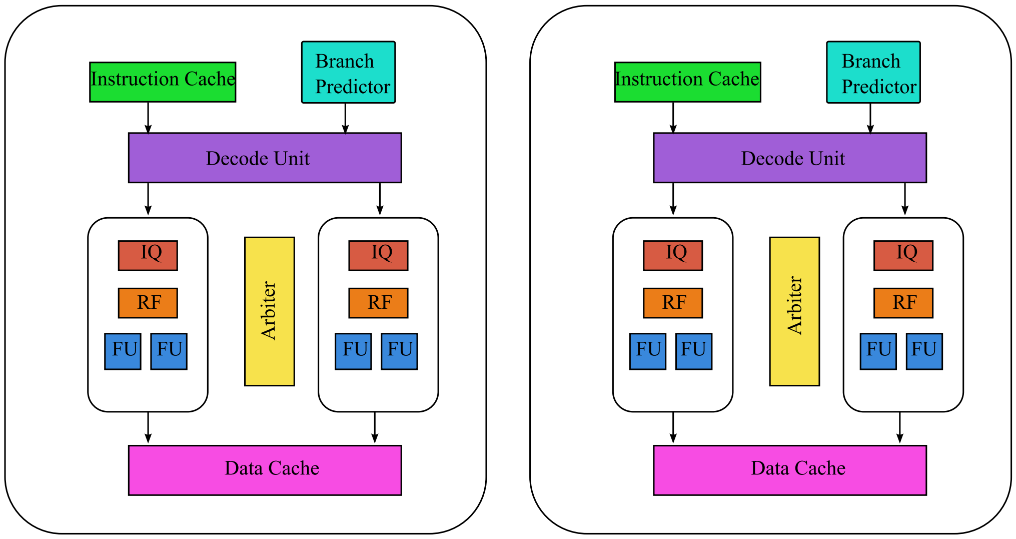


Figure 3.1. Hardware Architecture for Error Detection in a Clustered Core Processor

The clustered micro-architecture for a large multi-core processor core is divided into multiple clusters. Each cluster is small so that if the latency overhead for inter-processor communication can either be hidden completely by complete-to-commit times or is negligible in terms of impacting overall performance. Unlike a regular homogeneous multi-core processor, this architecture consists of multiple clustered cores rather than monolithic ones and each core is based on the clustered micro-architecture. Figure 3.1 illustrates the typical architecture of a multiple clustered core multiprocessor with two homogeneous clustered cores, each having two identical clusters. Each cluster consists of register files (RF), instruction scheduling queue (IQ), and functional units (FU), while instruction and data caches, branch predictor and decoder are shared by all clusters in a core. Each cluster shares an instruction arbiter for inter-processor communication. The overall architecture of our clustered core processor is shown in Figure 3.1. The detailed architecture of the instruction arbiter is described in the following section. A multiple clustered core multi-processor with two cores in each cluster can be effectively used for error detection. Since cores in a cluster are close in proximity and the latency overhead for inter-processor communication can be minimal. It is possible to exploit the clustered micro-architecture combined with multi-core architecture to trade-off between power and performance which is not considered in this work.

3.2 Mining Temporal Redundancy

It is generally not possible to detect a soft error in the processor pipeline before the instruction is executed at least once. Also, to achieve a low latency overhead, error detection must be done concurrently during execution of each instruction. Concurrent error detection can be achieved by complete duplication of the processor pipeline and issuing duplicate copies of each instruction. A soft error is detected when the result of the instruction and its duplicate instruction do not match. However, such an approach would be prohibitive in terms of area and power overheads. In this section, we propose a low overhead soft error detection technique by exploiting temporal redundancy. Temporal redundancy can be exploited by using compiler hints or by using a specialized predictor hardware. As discussed later, temporal redundancy in existing programs can be mined with a low overhead.

3.2.1 Compiler Directed Slack Computation

Temporal redundancy can be exploited by duplicating and re-executing instructions in a different time slot in the same processor. *Latency slack cycle* (LSC) value of an instruction is defined as the amount of cycles between the current instruction and the next true data dependent instruction. The LSC values can be determined statically during code generation in the compiling phase of a program. Given a fixed threshold (MIN_LSC), the compiler probes the nearest data dependency on any of the source operands and checks to see if the number of instructions between them is lower than the MIN_LSC. If this condition is met, then a special bit called the *slack bit* is set. This bit can be checked during the issue stage of the processor pipeline. The value MIN_LSC should be chosen in a manner such that the majority of the instructions should be able to make use of.

Given the slack bit, temporal redundancy is exploited for error detection by re-executing the instruction within MIN_LSC cycles. Once both instructions have been executed, comparison can be done to detect any transient faults that may have occurred. Slack information encoded in the instruction itself is used to determine if the result of the instruction are not immediately needed. The duplicate instruction can be executed following in time using a temporal redundancy approach.

The use of temporal redundancy for soft error detection is illustrated using Figure 3.2. A special *busy bit* is used which simply indicates if any of the issue slots are free in the current cycle. An instruction arbiter, which is a simple state-machine, and a special error

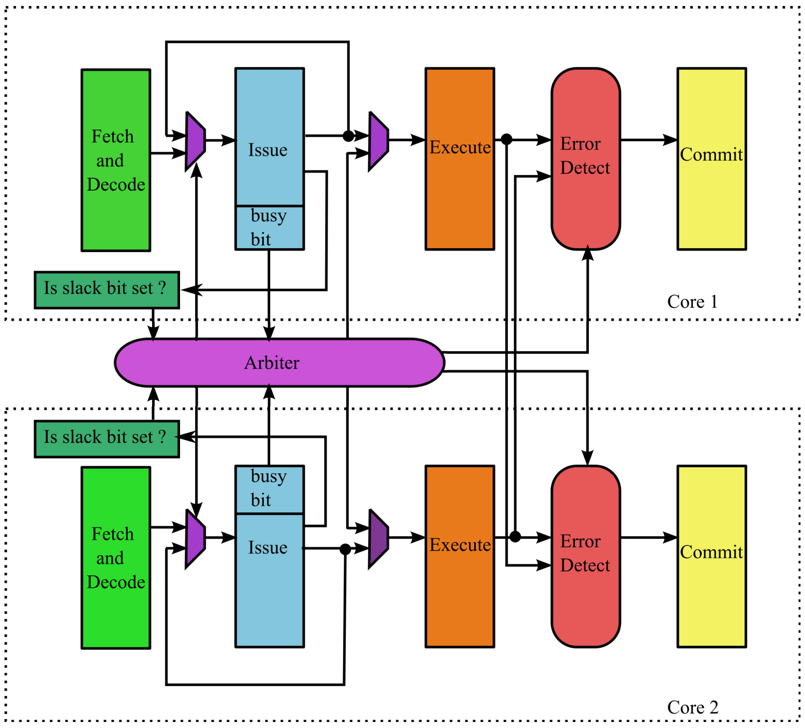


Figure 3.2. Instruction Arbiter for Temporal Redundancy Based Detection

detection stage in the processor pipeline between the execute stage and the commit stage, shown as the boxes with chamfered edges, to indicate the presence of internal buffers for storage. The sequence of steps for error detection is as follows. The slack bit is checked for each instruction that is issued. If it is set, the arbiter initializes itself for error detection using temporal redundancy and guides the multiplexor (MUX) at the input of the issue stage to issue the duplicate instruction within MIN_LSC cycles depending on the status of the busy bit in the issue stage. Since the arbiter knows *exactly* when the original and the duplicate instruction are issued and as the duplicate instruction does not wait in the issue queue due to data dependency, the arbiter can set the control signals at the appropriate clock cycle to latch the result from the execution of the original and duplicate instructions and compare them for error detection.

3.2.2 Dynamic Slack Implementation

In the previous subsection, a technique to statically estimate LSC using compile time analysis was described. The technique thus assumes that compiler performs additional tasks and encodes slack information in the form of the slack bit in the instruction. This technique thus can lead to increase in binary code size. Moreover, as the compiler directed technique only decides slack by analyzing instructions statically within a window, the slack estimate is not accurate. In this subsection, a technique is presented that exploits the temporal redundancy in a dynamic manner during execution.

A scheme for predicting instruction slack dynamically during execution has been developed. The technique is based on a *slack prediction* scheme and is illustrated in Figure 3.3. As shown in the figure, the slack predictor is a simple state based cache structure. Each entry in the array type structure stores a operation field, a destination register field, a slot to indicate a execution start time and a user defined number of voter slots. The slots indicate a vote on various binned predicted slack values. The number of slack slots for each cache entry depends on the granularity of binning of the predicted slack and its range. For example, if the instruction slack varies between 2 and 6, with a step size granularity of 2, there will be three slots for the corresponding cache entry. The three slots correspond to the slack values of 2, 4 and 6. The vote on each slack value slot indicates a confidence level on the amount of times the predicted slack was indeed correct.

The hardware architecture for dynamic slack prediction has been illustrated in Figure 3.3. The destination register of an executed instruction is noted in the destination register field of a free cache entry. It should be noted that this register number is the register address after register remapping which is available at the decode stage. When the corresponding instruction starts execution we also time stamp the execution start time slot. During this initialization, the votes associated with the cache entry are also set to zero. For each in-

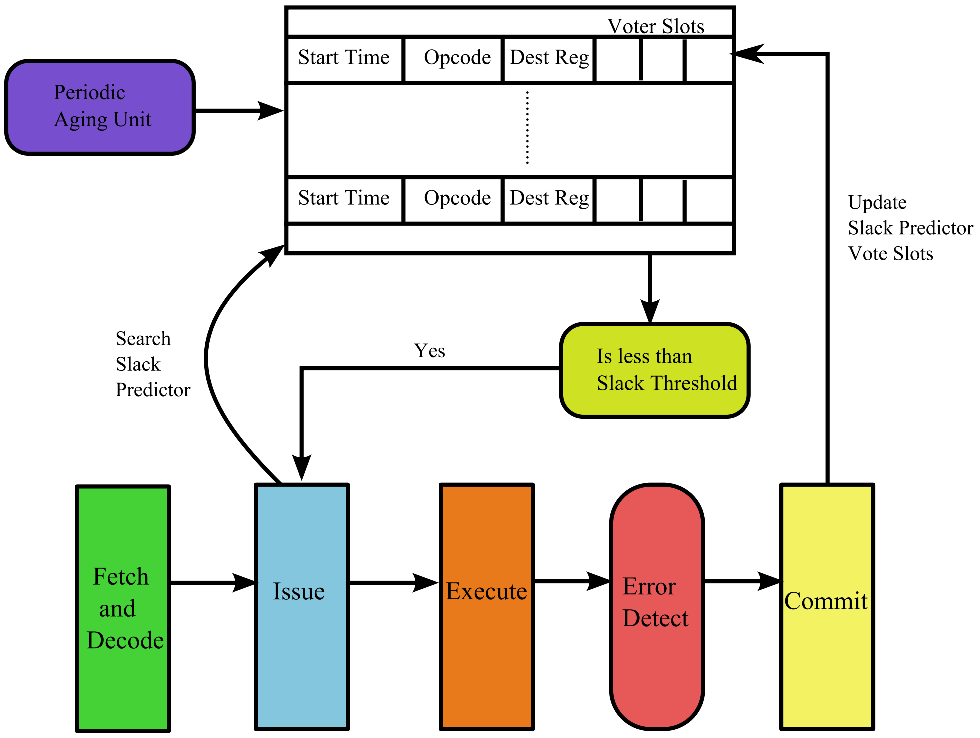


Figure 3.3. Slack Predictor Hardware for Dynamic Temporal Redundancy Based Detection

struction, the slack predictor cache is also checked to see if the source registers for that instruction matches any destination register in the slack predictor for any previous operation. If such a match occurs, the current time step is again noted and the difference between the start time stamp is binned and the vote for the corresponding predicted slack slot is increased by one. Thus, for each instruction, the destination register and the opcode and the execution start time stamp are noted in an available cache entry, as well as the source registers are checked to find if they match any previous destination register entries for any operation. In case of a match, the slack value with maximum votes is selected as the predicted slack for this instruction. The slack votes are gradually aged to a saturating zero at regular intervals of time. The LRU replacement policy is used for the individual entries of the slack predictor. The dynamic slack prediction scheme thus mimics a simple voting scheme. A correct prediction means that the actual slack will be truly higher than the predicted slack and the duplicated instruction can be executed without any performance penalty. In this case, the next data dependent instruction still executes at the same time as that without instruction duplication.

3.3 Mining Data Value Redundancy

It is commonly known that a large percentage of memory values are small [5], [24], [33]. Typically, small memory values use at the most half of the bit width of the registers. These small data values can be exploited to increase redundancy and improve the reliability of the processor pipeline. Small value replication is carried out for instructions with operand values that can be represented with half of the bit width size [24]. Small value based error detection is carried out differently on different operations. For example, if both source registers meet this requirement then their values are replicated and executed on the same functional unit. Note that before the execute stage of the pipeline, the data values of the source operands are already available. Thus, source registers with small data value width can be locally duplicated and both copies of the small data value can be executed on the same functional unit and compared for error detection. If the results are correct, the upper half words are filled with 0s or 1s to obtain the actual results.

Soft errors during execution of arithmetic operations like addition and subtraction can also be detected even if one operand is of small data value, by using value based redundancy. For example, consider the case when one operand is small and the arithmetic operation on the other operand does not generate any carries. In this case, the upper half word of the other operand can be stored in a register and the small value operand in the lower half word of the other operand can be locally duplicated as before. Again, both copies of the small data value operands can be executed on the same functional unit and compared for error detection. If the results from the upper and lower half words agree, the upper half word of the other operand is concatenated with the lower half word of the result to form the actual result. Thus, in the absence of a carry, local duplication can be applied to detect errors in the execution of arithmetic instructions even involving only one small operand. The same idea can also be extended for logical operations. For example, for a AND or OR operation if only one operand is small, local duplication can be applied to perform duplicate execution of the operation and detection of errors. The upper half word of the result of the logical operation can be recovered by filling them with all zeros, all ones, or with the upper half word of the other operand.

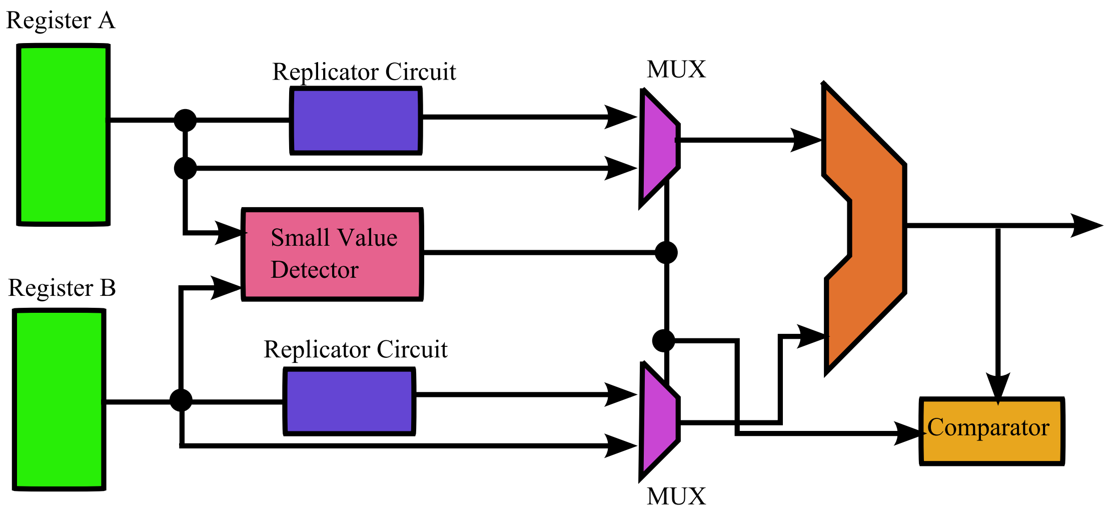


Figure 3.4. Error Detection Using Small Values

Figure 3.4 illustrates the hardware architecture for small value detection method. The source operand values stored in the registers are sent to the *small value detector* (SVD) to determine whether their data values are small. The SVD circuit is simply a zero detector for the MSB portion of the data bits. If a small value is detected, the SVD sends the signals to the multiplexors so that the replicated values are chosen. The two duplicate values are then sent to the ALU for execution. The results of the ALU operations are compared with a comparator that is controlled by the SVD circuit. The SVD is checked to see if it indicates a small value when the MSB and the LSB portion of the result should match. However, if a small value is indicated by the SVD and LSB and MSB portion of result does not match a soft error is detected and recovery by rollback from the previous checkpoint state can be initiated. Thus, as shown in Figure 3.4, error detection by exploiting small data value size can be implemented without much latency and area costs.

3.4 Information Redundancy Based Detection

In general, parity codes can be used to detect single bit soft errors in memory, however, parity check bits are not preserved across arithmetic operations. To maintain check bits across arithmetic operations, residue codes have been proposed in the literature. Residue codes have the desirable property that for arithmetic operations the check bits of the result can be determined directly from the check bits of the operands. Residue code is a systematic code, i.e., the check bits can be represented separately from the data bits. The residue check bits *C* are represented by $log_2 N$ bits , where N is the total number of bits used for representation, and can be computed as $C = (N) mod(m)$ where m is represents the residue of the code. It can be shown that for a pair of operands, the residue code for the sum (or product) of the operand is the sum (product) of the residue codes of the operands modulo m and that if the residue m is odd all single bit errors occurring during any arithmetic operations can be detected.

As shown in Figure 3.5, error detection using residue code requires only an extra adder unit. However, as the extra addition operation is off the critical path and since the delay due to the multiplexors can be neglected, no performance penalty will be incurred. If the residue check bits of the sum (product) do not match the sum (product) of the check bits of the operands, a single bit error is detected. Otherwise, no single bit error has occurred and the result is committed to update the processor state. The determination of the residue code is dominated by the computation of modulo addition. However, the operation can be

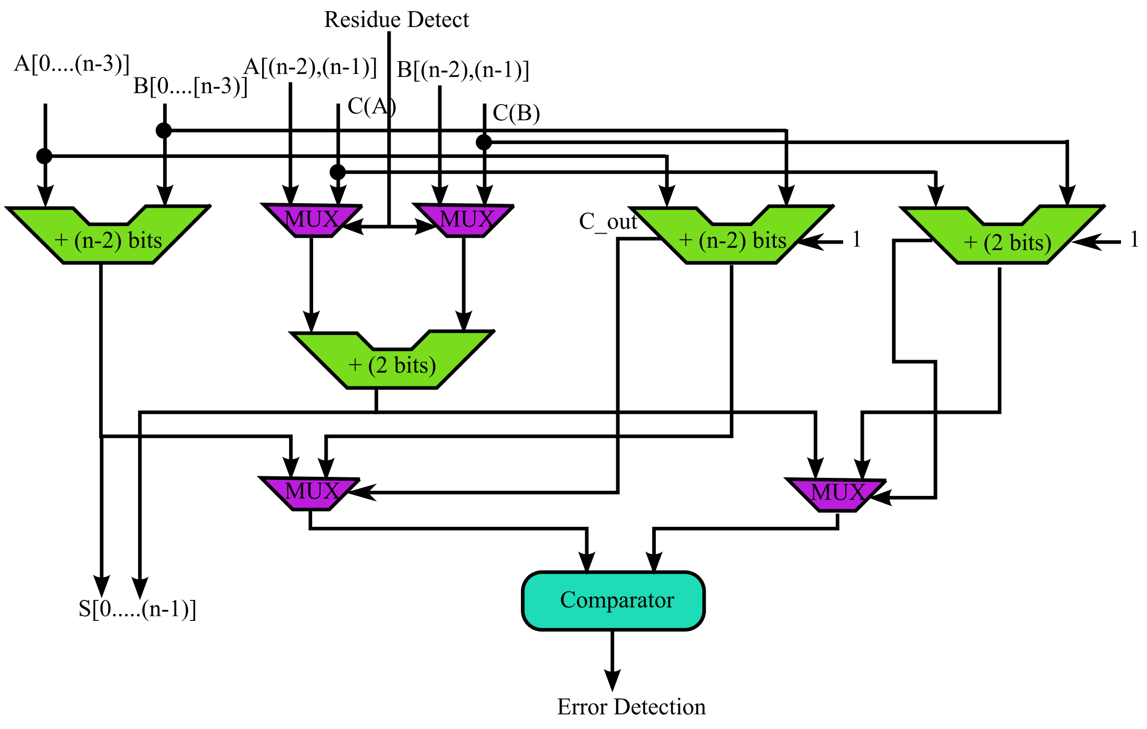


Figure 3.5. Error Detection Using Residue Code

simplified if the modulo is of the form $2^n - 1$, as follows,

$$\begin{aligned}(x+y)mod(2^n-1) &= (x+y+1)mod(2^n) \quad if(x+y+1) \geq 2^n \\ &= (x+y)mod(2^n) \quad otherwise \end{aligned} \tag{3.1}$$

where x and y represents the source operands and n represents the bit width of the registers used to store the source operands. In this work, this property can be exploited and the two most significant bits are used for representation of the residue parameter m.

3.5 Proposed Multi-Core Architecture

In the previous sections, several low overhead soft error detection techniques were described which exploit the various types of redundancy that can be mined within the boundary of the single core. Error detection is performed during the issue stage of each core's processor pipeline for each instruction. The error detection mechanism, in order to maintain low overheads, makes use of the available functional resources as well as the remaining latency slack cycles (LSC) which is the number of cycles before the computed result becomes the source operand of a subsequent instruction. For instructions with high LSC, the soft errors are detected using temporal redundancy wherein the duplicate instruction is executed at a later time step and compared with the result of the original instruction ensuring that there is no loss in performance. For instructions with small operand value size, soft errors are detected by duplicating the operands and executing on the same functional unit, thus using the available resources to maintain low overhead. For instructions with low LSC and not having small valued operands, detection is implemented by using the residue code. If none of these methods are applicable, the duplicate instruction is executed on a nearby idle processor core. The overhead associated with inter-processor communication is reduced using a multiple clustered core architecture. Finally, correction is done by recovery and rollback from the checkpoint states. It should be noted, however, that error correction is required only when a soft error has actually happened, which is rare, while error detection needs to be performed during execution of each instruction. Therefore, throughout this chapter, The focus is on providing techniques for low overhead error detection in multi-core processors.

Previously, it was discussed how the slack bit is used in an instruction arbiter for error detection using temporal redundancy. In the case when the slack bit is not set, the arbiter initializes itself for error detection by using spatial redundancy. The arbiter latches the instruction and its source operand values from the original core, keeps on polling the busy bit of the issue stage of the nearby core and once the busy bit is disabled, the duplicate instruction is issued to the adjacent core. The arbiter then sets the control signals at the appropriate clock cycles in the error detect stage to latch the results of the execution from the original core and the results of execution from the nearby core which are then compared for error detection. This is possible as the arbiter knows exactly when issue starts and execution finishes for the original and duplicate instructions. Many variants of this idea have been proposed [2], [73] in the literature. The experiments with redundancy based detection technique suggested that the scheme of spatial redundancy incurs a considerable latency overhead due to the interconnect delay in sending/receiving the instruction to the arbiter. In general, using the spatial redundancy based detection can lead to high performance overhead and about 200% increase in power. However, by mining of various redundancy schemes and using the clustered core micro-architecture, this scheme is seldom used for error detection. The overall hardware algorithm for error detection in a CMP processor using our proposed schemes is shown in Algorithm 1.

Algorithm 1 The algorithm for soft error detection

```
Compute LSC value for all ALU instructions statically or dynamically
for Each ALU instruction do
    if $slack > S_{Th}$ then
        Detect soft error using temporal redundancy
    end if
    if $slack < S_{Th}$ AND Both source operands are small then
        Duplicate data values and execute on the same functional unit
        Detect soft error if the LSB and the MSB of the result do not match
    end if
    if $slack < S_{Th}$ AND Both source operands are not small AND Two upper bits of the
    source operands are zero then
        Compute residue code for the operands
        Replace the top 2 bits of the operand with check bits
        Execute the operation and compute modulus m
        Compute the check bits of the data without the top two bits
        Compare this check code with the top two bit and detect soft error if they do not
        match
    else
        Duplicate the instruction into a nearby idle core
        Commit from ROB only when both original and duplicate instruction's result match
    end if
end for
```

The error recovery mechanism achieves error correction using the traditional rollback and recovery scheme. Thus, a checkpoint state of the processor after execution of each instruction is maintained. Such fine-level checkpointing can easily be achieved by using a structure similar to a re-order buffer. Instructions are committed from the re-order buffer only if no error has been detected. Otherwise, all instructions in the re-order buffer are flushed and instructions are re-executed from our last checkpoint. Thus, the hardware structures already present in current processors are leveraged for speculative execution, for error correction.

3.6 Experimental Setup

Table 3.1. Processor Configurations

Parameter	Configuration
Active list	64 instructions
fetch and commit rate	4 per cycle
Functional units	3 ALUs, 2 FPUs, 2 Address gen. units
Branch predictor	2-bit history predictor
Cache Line Size	64 Bytes
L1 Cache	32KB 8-way associative, Write Back
L1 Cache Latency	6 cycles
L2 Cache	64KB 8-way associative
L2 Cache Latency	16 cycles

In this section, the experimental setup used for simulations and the results of the various schemes proposed in the chapter for improving the reliability against soft errors for multi-core systems are described. The RSIM multi-multiprocessor simulator was modified [27] and used for this study. RSIM is an execution-driven simulator primarily designed to study shared memory multi-processors and can simulate applications compiled and linked for SPARC V9/Solaris. The base specifications are given in Table 3.1. Each core uses static scheduling, has a issue width of 4, with 3 integer and 2 FP ALUs, with a active list of 64 entries and used a 2 bit branch predictor. The local L1 cache is 32KB 8-way associative while the shared L2 cache is 64KB 8-way associative with cache access latencies selected as given in [29]. The soft-error detection architecture follows the procedure presented in Algorithm

1. Both redundancy based and information dependent methods can be determined during run time by checking the source operands during the decode stage of RSIM. The simulation results are stored in a well defined data structure so that they can be examined using Perl scripts after run-time. The simulations were performed on a subset of the SPLASH benchmark suite [72] that were ran on a Sun Blade 1500 uniprocessor with 4GB of RAM. Within a superscalar processor, the performance overhead of temporal redundancy due to redundant instructions delaying independent instructions is negligible. The performance overhead of our proposed schemes is calculated as follows

$$overhead_{perf} = I_{spatial} \times cost_{spatial} \tag{3.2}$$

where $I_{spatial}$ is the number of instructions that are forced to use the spatial redundancy technique, and $cost_{spatial}$ is the amount of additional cycles it takes to execute the replicated instruction in the neighboring core. Power calculations are given for the 16-core processor using the Nangate FreePDK 45 nm Open Cell Library. Power consumption values for the various techniques were calculated using the product between the percentage of usage and the power consumption of the functional units used.

3.7 Simulation Results

Figure 3.6 shows the average slack values in cycles for each benchmark for the implementation of static temporal redundancy. Since the static temporal redundancy technique is implemented on the compiler level, we assume single-issue width so that the technique is architecture independent. From the results given in column *avg*, it is clear that the appropriate LSC value to use is 5. Note that the MIN_LSC value may vary from different applications but the value of 5 suits best for the set of benchmarks. The results from the static slack implementation of *temporal redundancy* are given in Figures 3.7- 3.9. The

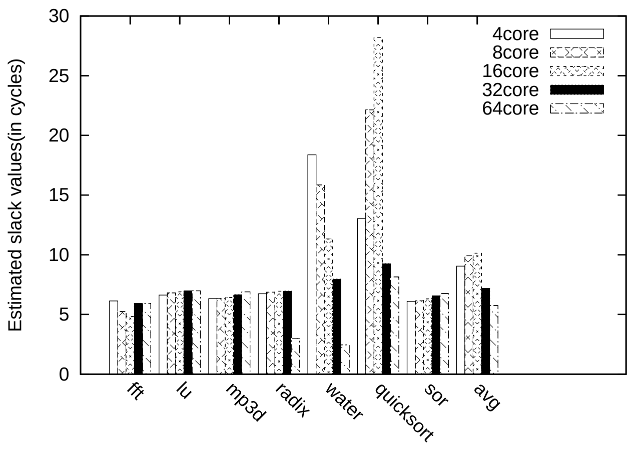
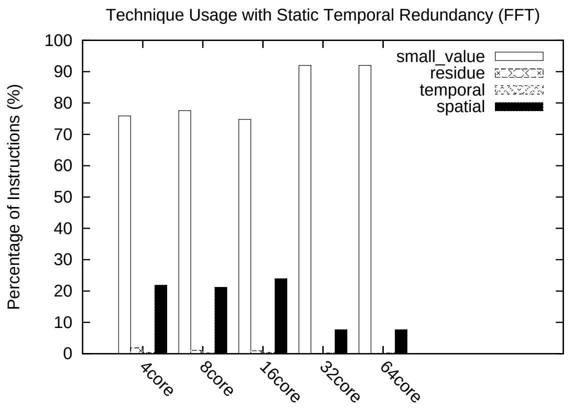
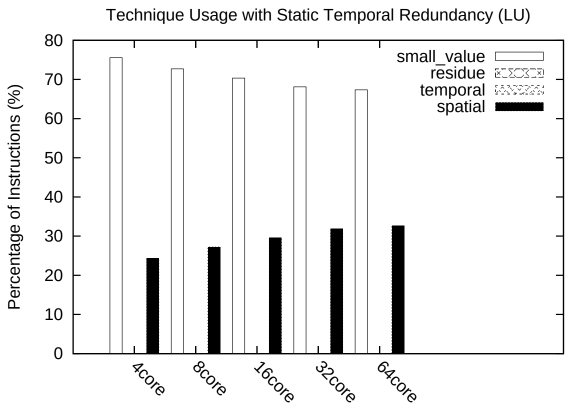


Figure 3.6. Static Slack Breakdown

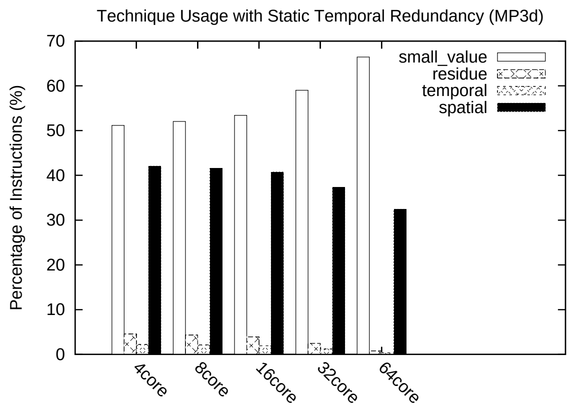


(a) Static temporal redundancy SPLASH-FFT

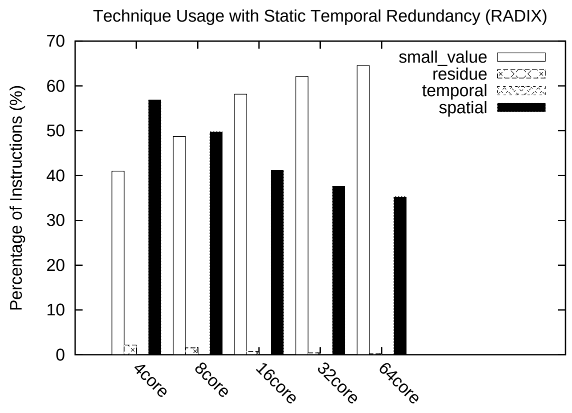


(b) Static temporal redundancy SPLASH-LU

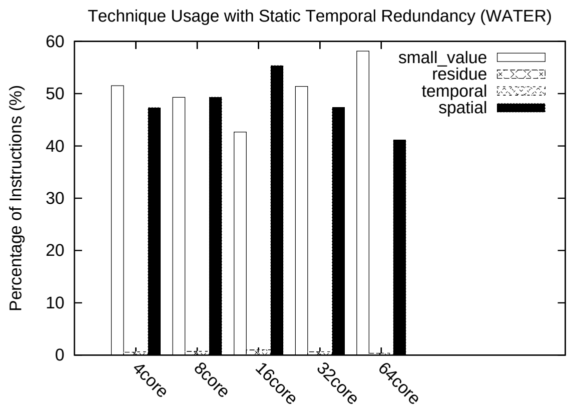
Figure 3.7. Results From Static Temporal Redundancy Technique for SPLASH-FFT and SPLASH-LU



(a) Static temporal redundancy SPLASH-Mp3d

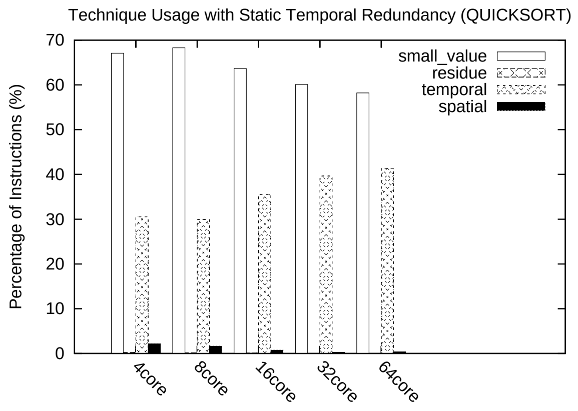


(b) Static temporal redundancy SPLASH-Radix

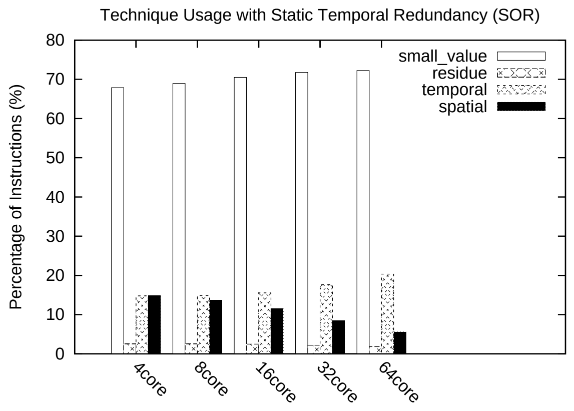


(c) Static temporal redundancy SPLASH-Water

Figure 3.8. Results From Static Temporal Redundancy Technique for SPLASH-Mp3d, SPLASH-Radix, and SPLASH-Water



(a) Static temporal redundancy SPLASH-Quicksort



(b) Static temporal redundancy SPLASH-SOR

Figure 3.9. Results From Static Temporal Redundancy Technique for SPLASH-Quicksort and SPLASH-SOR

majority of the benchmarks had less than 30% of their instructions using the spatial redundancy technique. The benchmarks, *quicksort* and *sor* were the most efficient with respect to the usage of static temporal redundancy technique. The other benchmarks either had high usage of data-dependent techniques or the MIN_LSC value may have been too large, thus giving us a low percentage usage for the static temporal redundancy method. To further lessen the amount of spatial redundant instructions, in the dynamic slack technique, 1, 2, and 4 cycles are used as the voting choices for the dynamic slack implementation. The results with the dynamic slack implementation are given in Figures 3.10-3.12. As illustrated in all benchmarks, more instructions are able to implement the temporal redundancy method, thus lowering spatial redundancy usage as well as the performance overhead.

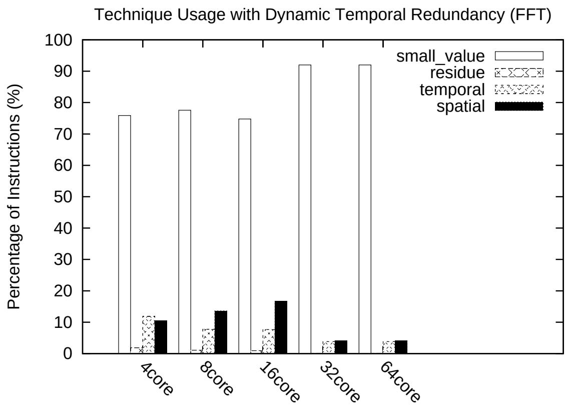
For the correct dynamic prediction rate and power overhead, only the 16-core configuration was analyzed. Figure 3.13 shows that most predictions were correctly predicted for our benchmarks. Figure 3.14 shows that the technique generates little power overhead. The instruction arbiter at the RTL level was designed using Verilog and synthesized it with a standard cell library of 45nm technology using Design Compiler. The small value based detection scheme only required small number of MUXes while residue code based detection required an extra n-bit adder. The calculations indicate that the overall area overhead for our combined error detection framework was less than 5%. In Figure 3.15, the percentage overhead in CPI compared to a multi-processor with no error detection capability for 8, 16, 64 and 128 cores was plotted. As shown, the performance overhead was only 8% for the 8 and 16 core multi-processors while it was 10.5% for the 64 core system and 13% for 128 core multi-processor system. It is noted that the large increase in average CPI is partly due to the large amount of spatial redundancy instructions in the mp3d benchmark, which if excluded will lead to a even lower performance overhead.

In Table 3.2, the work presented is compared with some recent works provided in literature, using the data provided in the papers and extrapolating according to the experi-

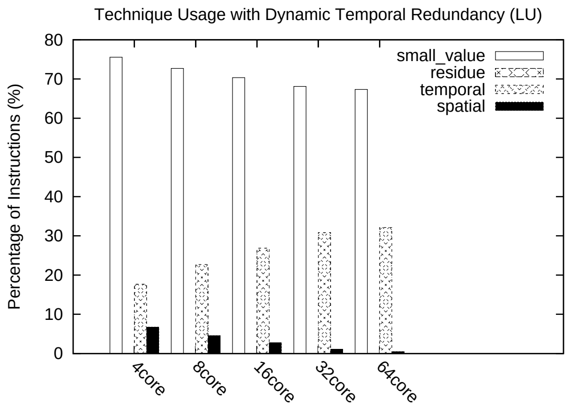
Table 3.2. Comparison with Related Works

Approach	Error Coverage	Latency Overhead
RMT [49]	100%	21%
CRT [20]	100%	15%
Slipstream [74]	74%	-7%
This work	100%	less than 10%

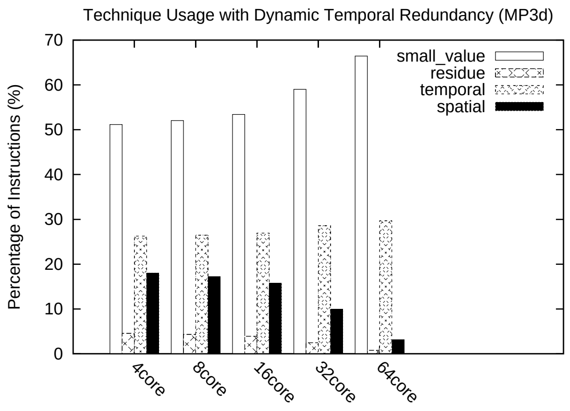
mental setup. SRT techniques [49] executes redundant threads on a SMT processor for error detection. The CRT technique extends the same idea on CMP processors. The CRT approaches [20] achieves considerably low latency overhead compared to instruction lock stepping in CMP processors. This is due to the fact that the trailing thread can effectively utilize speculative outcomes for a speedier execution. However, the trailing thread may read values which can be modified by other instructions in the leading thread and hence false positives may occur. The problem is circumvented by using additional hardware structures like the load value queue and by ensuring stores are committed only after the stores from the leading thread finishes execution, thus not changing memory state before checking. Further, as relative thread execution rates on different processors are non-deterministic, events among concurrent threads in a program cannot be replicated precisely and efficiently, leading to spurious divergences [53]. It is shown in this work that by mining and utilizing various redundancy mechanisms available within a core's boundary and using a clustered core architecture, inter-processor communication overhead can be made negligible. Using the various techniques proposed in this chapter, duplication on a per-instruction basis is more attractive than duplicating at the thread level. Slipstream processors proposed in [74] uses a reduced instruction stream in the leading thread to guide the trailing thread. As the leading thread is reduced in size and as it can provide key speculation outcomes to the trailing thread, this method can actually improve performance than executing the single thread with speculation. However, compared to this approach and the approaches described in CRT, the error detection coverage is not high and hence is not viable in server application where reliability can be a key concern. Thus, compared to other works proposed in literature, this combined framework can achieve complete error coverage at significantly lower latency overheads with negligible area cost.



(a) Dynamic temporal redundancy SPLASH-FFT

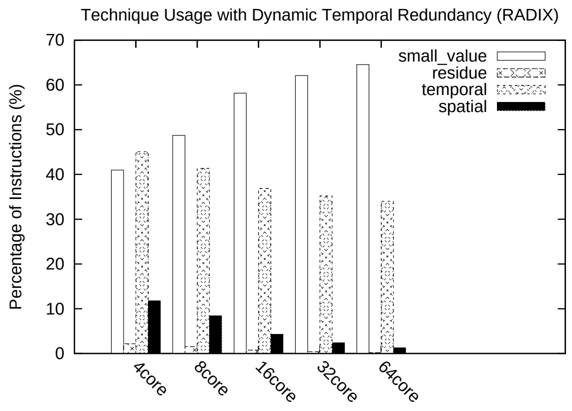


(b) Dynamic temporal redundancy SPLASH-LU

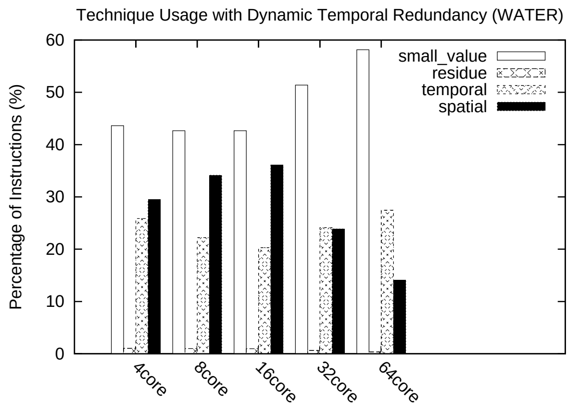


(c) Dynamic temporal redundancy SPLASH-Mp3d

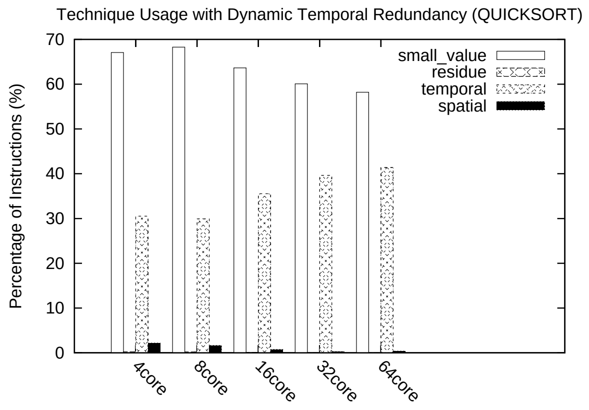
Figure 3.10. Results From Dynamic Temporal Redundancy Technique for SPLASH-FFT, SPLASH-LU, and SPLASH-Mp3d



(a) Dynamic temporal redundancy SPLASH-Radix

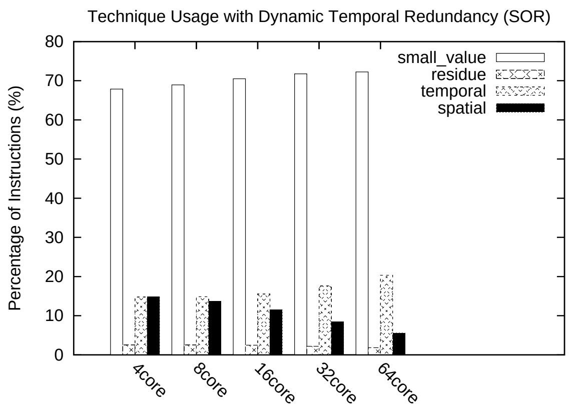


(b) Dynamic temporal redundancy SPLASH-Water



(c) Dynamic temporal redundancy SPLASH-Quicksort

Figure 3.11. Results From Dynamic Temporal Redundancy Technique for SPLASH-Radix, SPLASH-Water, and SPLASH-Quicksort



(a) Dynamic temporal redundancy SPLASH-SOR

Figure 3.12. Results From Dynamic Temporal Redundancy Technique for SPLASH-SOR

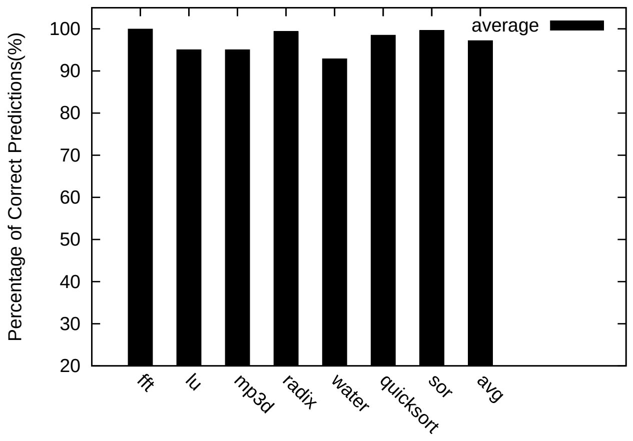


Figure 3.13. Dynamic Prediction Technique

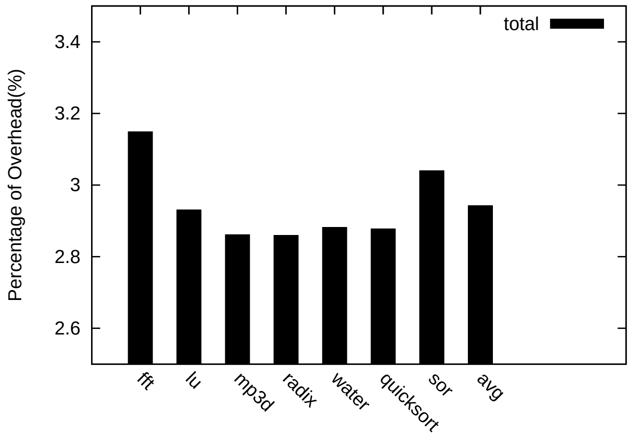


Figure 3.14. Power Results

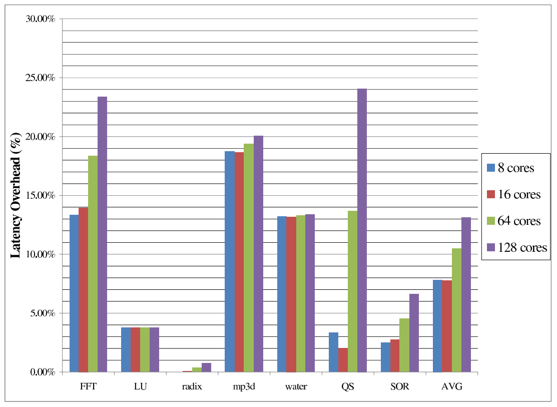


Figure 3.15. Performance Overhead for the Proposed Schemes

CHAPTER 4

PEAK POWER MINIMIZATION USING PATH CLUSTERING

4.1 Introduction

Due to technology scaling, with the increase in chip densities and clock frequencies, the demand for the design of low power integrated circuits has increased. This trend of increasing chip density and clock frequency has made reliability a major issue for the designers mainly because of the high on-chip electric fields [71, 75]. Several factors such as the demand of portable systems, thermal considerations and environmental concerns have further driven the area of low power design [71]. In low power design using deep submicron and nanometer technologies, both the peak power and the total power are equally critical design constraints. In this chapter, the focus is on the reduction of peak power through use of a novel clock management strategy.

Peak power is the maximum power consumption of the integrated circuit (IC) at any instance during its execution. Peak power can be defined as the maximum power consumption during any clock cycle. The reduction of peak power consumption is essential for the following reasons : (i) to maintain supply voltage levels and (ii) to increase reliability. High peak power can affect the supply voltage levels. The large current flow causes high IR drop in the power line, which leads to the reduction of the supply voltage levels at different parts of the circuit. High current flow can impact reliability because of hot electron effects and high current density. The hot electrons may lead to runaway current failures and electrostatic discharge failures. Moreover, high current density can cause electro-migration failure. It is observed that the mean time to failure (MTTF) of a CMOS circuit is inversely proportional to the current density (or power density).

The reduction of power consumption can be achieved at several levels, such as algorithmic, behavioral, register-transfer level, logic, gate and transistor [62] during the design flow. Several techniques have been proposed for peak-power reduction at different levels. At the algorithmic level, choosing a low power algorithm can be helpful in power reduction. Techniques, such as, shutting down unused operators and multiple supply voltage operators are used at behavioral level. At RTL level, proper state assignment is needed for power reduction. Low power design methods at logic level includes, the use of asynchronous circuits, the reduction of number of transitions and the reduction of hazards. The use of adiabatic systems and static CMOS can lead to low power consumption at the gate level. At the transistor level, smaller device geometry, smaller supply voltage and threshold voltage are the possible options for power reduction. During low power synthesis at the behavioral level, several low power subtasks, such as, scheduling, allocation and binding are performed with the goals of total and/or peak-power reduction.

As the IC processing technology continues to scale down, and the demand for high performance IC designs continue to increase, high peak current and peak power pose critical problems towards achieving higher performance and reliability. The peak current problem has been addressed in the industry by increasing the size of power and ground lines, and increasing the number of pin pads on the chip. The problems that are caused by high peak current values can be seen at all levels of processor designs. On the VLSI level, high peak current causes IR-drop which generates power supply noise and ground bounce. This power supply noise causes inconsistent supply voltage values to the transistors which can be translated to slower circuits and timing violations on the circuit level. Timing violations in turn translate to logic failures on the architecture level.

Synchronous-logic design methods rely on the master clock and/or the clock tree within clock domains that present drawbacks to the physical operation (as opposed to logic operation) of the digital-logic integrated circuit. These physical drawbacks are manifested as large peak (surge) power-bus current, increased RMS current, power and ground bus noise, ground bounce, simultaneous switching output (SSO) noise, input-threshold margin loss, timing jitter, and increased propagation delay (due to integrated-circuit power-bus compression and heating), and system electromagnetic interference. These deleterious effects can occur in the integrated core logic circuits and within the input/output ring. Thus, the focus is on a RTL-level method that takes advantage of the logic-path timing slack to re-schedule circuit activities at optimal intervals within the unaltered clock period. When switching activities are redistributed more evenly across the clock period, IC supply-current consumption is also spread across a wider range of time within the clock period. This has the beneficial effect of reducing peak-current draw in addition to reducing RMS power draw without having to change the operating frequency and without utilizing additional power supply voltages (as in dual or multi VT approaches). A major goal of this approach is that the proposed method can be utilized in conjunction with most power-reduction methods such as clock gating, multi- VT, power switching/power shut-off; back-end processes such as floor-plan and interconnect optimizations, and leakage reduction methods.

In this chapter, an innovative clock control strategy has been proposed and patented by [79]. A path clustering algorithm is introduced that uses the delay slack relative to the maximum delay to cluster the circuit paths together. Using the generated clusters, each cluster can be attached to a differently phase-shifted clock. This minimizes the number of simultaneous paths being executed at the same instant which implicitly reduces the peak power. A multiphase clock generator circuit is needed which will take the domain clock and produce the multiphase clocks using a decoder and counter architecture. Considering the size of the overall circuit, the overhead due to this clock circuitry in terms of area is minimal and it does not increase clock power due to the fact that the load on each phase clock is significantly smaller than the load on a single domain clock driving the entire circuit. Based on the experiments with the ISCAS 85 benchmark circuits, OpenCores circuits and LEON processor multiplier circuit, the proposed technique significantly reduces the peak power without significantly increasing clock-power or area overhead as presented in the experimental section. This work was supported by Florida CONNECT and East-West Innovation Corporation.

The rest of the chapter is organized as follows. In section 4.2, the problem formulation and the proposed clock scheduling strategy are discussed. In Section 4.3, a description of the proposed algorithm including the clustering step which together describe the clock scheduling algorithm to be incorporated in a design tool flow. The experimental set up and the design automation flow as well as the results are discussed in Sections 4.4 and 4.5.

4.2 Clock Control Strategy

The peak power of a circuit can be defined as

$$max \int V_{dd} i_{dd}(t) dt \tag{4.1}$$

where V_{dd} is the supply voltage and $i_{dd}(t)$ is the amount of current drawn by the circuit at time *t*. Given this equation, minimization of the peak power at a given time *t* is directly proportional to the amount of current drawn at time *t*. Since current is flowing ideally only when a circuit is active, by minimizing the number of simultaneously active elements, the spike in current drawn can be reduced from the power supply, thus reducing the IR-voltage drop. In clocked circuits, all register elements are active on the rising and falling edges of the clock causing a large draw in current and thus having the greatest effect on the peak power of the circuit.

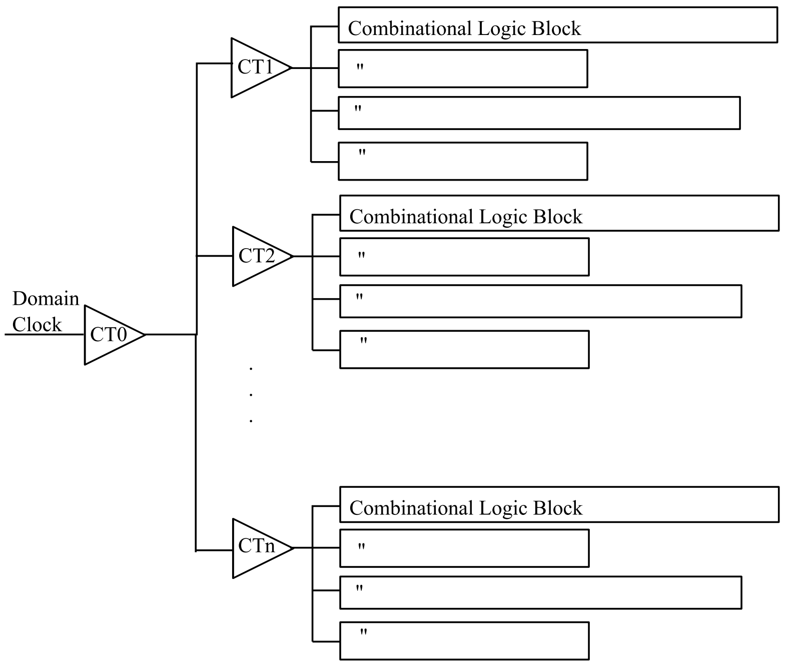


Figure 4.1. Traditional Clocking of Combinational Logic Blocks

An example of a traditional synchronous clock tree driving combinational logic blocks is shown in Figures 4.1 and 4.2. In Figure 4.1, the domain clock is distributed by a clock tree (CT) using CT buffers: CT1, CT2, through CTn. The individual clock tree drivers strobe numerous combinational logic blocks (CLB) of varying propagation time. The exact CLB propagation time depends on the logic block design, the combinational input to the CLB, and the inherent integrated-circuit speed. The focus herein is an CMOS integrated circuit, where inherent CMOS speed is a function of process, voltage, and temperature (PVT).

Figure 4.2 exemplifies the contributions of CLB currents to the total current in a CMOS IC. After a small CT delay, the strobe signals simultaneous CLB activity, with a commensurate sharp increase in the supply current. This creates an undesirable abrupt change in the supply current, di/dt. A large di/dt is difficult for power generation and distribution, which creates power-bus noise, and causes device voltage starvation (compression) due to bus inductance multiplied by di/dt ($V = L \times di/dt$). In addition to these problems with the high transient-current demand, large di/dt is a large current variance, invoking large RMS

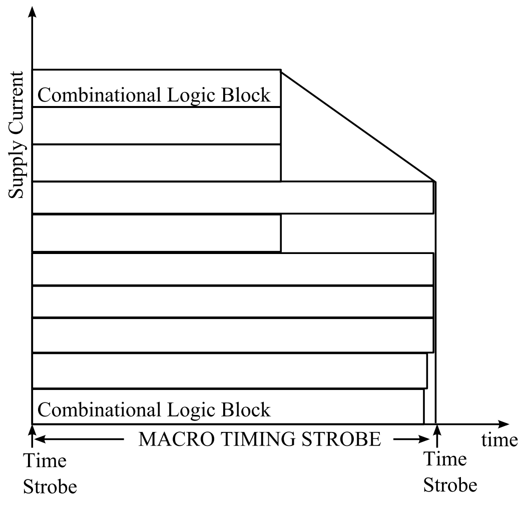


Figure 4.2. Current Profile for Traditional Clocking

current with increased power losses in the power grid. The idealized traditional current demand is shown in Figure 4.2, where CLB currents are depicted as ideal rectangles of various time durations and current amplitudes, all summed to form an aggregate power-bus current shown as a bold black line enveloping the CLBs. In the proposed method, peak power is reduced by taking advantage of timing slack of individual signal paths within a particular clock domain. Timing slack data is obtained from the tool-flow path-delay database, and its query time is only a linear function of IC complexity. It has been well established that timing slack is abundant in most IC designs which forms the main motivation behind our method.

The methodology consists of the following key steps listed as follows:

1. Combinatorial Logic Block (CLB) propagation time ranking algorithm is a routine that calculates the maximum propagation time through the CLB.

2. CLB Grouping/Clustering step in which the CLBs of similar maximum propagation time are grouped together to receive the optimal delayed clock to minimize di/dt.

3. The result of a di/dt minimization determines phase-shifted clock outputs to be assigned to various clustered CLBs.

By taking advantage of the timing slack and judiciously reallocating the activities of the various combinational logic groups into specified time intervals within a clock period (refer to Figure 4.3), the total current draw within the clock interval is moderated. The domain clock illustrated in Figure 4.3 can be viewed as the output driver of an buffer node apart of a H-tree clock distribution network. Through balancing the number of paths belonging to each cluster, it is expected that the interconnect delay is equal for the clock domains Φ_1, Φ_2 up to Φ_n. Also since the buffers are inserted at a particular node in the H-tree, the paths are considered to be in close proximity. Buffer design for generating the buffer delay can be implemented by adjusting the P/N transistor size ratio to skew the charging of the load driving the combinational paths. This skew varies from each circuit and is not considered in this work since our analysis is performed before placement and routing of the design. A conceptual traditional, zero-skew clocking scheme is depicted in Figure 4.4. On the left, four CLBs are clocked simultaneously. The additive current of these four CLBs is shown to the right; where the peak current is 4 A. However, using the new clock control strategy to schedule the operation of the CLBs as shown in Figure 4.5 (left), produces a summed current shown on the right side of Figure 4.5. The aggregate current variance is dramatically reduced with our clock control strategy, as exemplified in Figure 4.5 (the additive current of all the CLBs is represented as the bold black trace). Time T1, T2, T3, and T4 are the timing epochs for the respective blocks CLB1, CLB2, CLB3, and CBL4. Here, power is reduced via several mechanisms. First, by judiciously scheduling tasks in a synchronous system, peak current and peak-power consumption is reduced. Second, reducing peak current decreases the RMS current which is consumed in the power and ground grids. Third, reduced peak current and reduced transient current (di/dt) reduce the

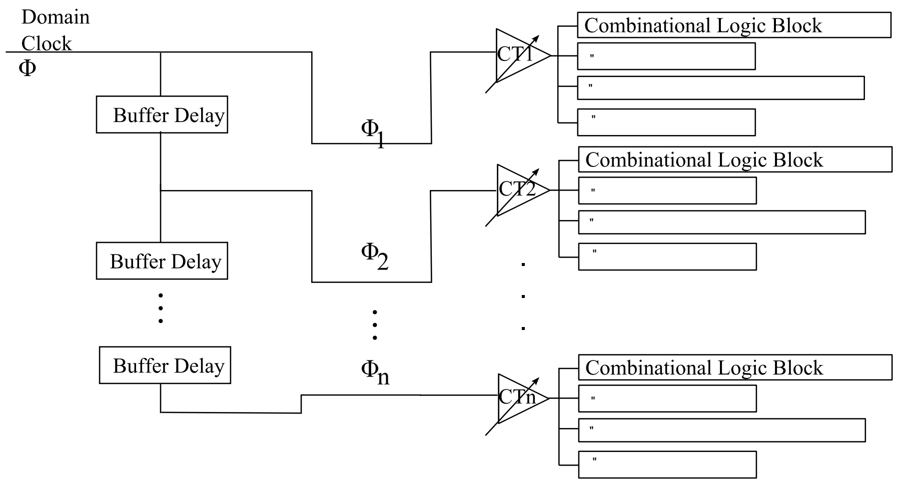


Figure 4.3. Phase Shifted Multi-Clocking of Combinational Logic Blocks

power-grid dynamic voltage drop, thereby allowing reduced decoupling capacitance, metal, and even reduced supply voltage.

It will be seen in the results section that the method leads to significant peak-power reduction.

4.3 Proposed Peak Power Reduction Technique

In order to optimize the peak power of a circuit, the number of circuit elements that are simultaneously switching must be reduced. This can be accomplished by identifying which paths have delay slack among the various paths and then utilize the slack values to reduce the number of paths that are simultaneously switching. The circuit is modeled as a graph structure and timing analysis is performed to identify the delay slacks for each primary input among the combinational paths. With the obtained slack information, the clustering algorithm is ran to cluster and categorize the primary inputs (combinational paths) with similar delay slack values.

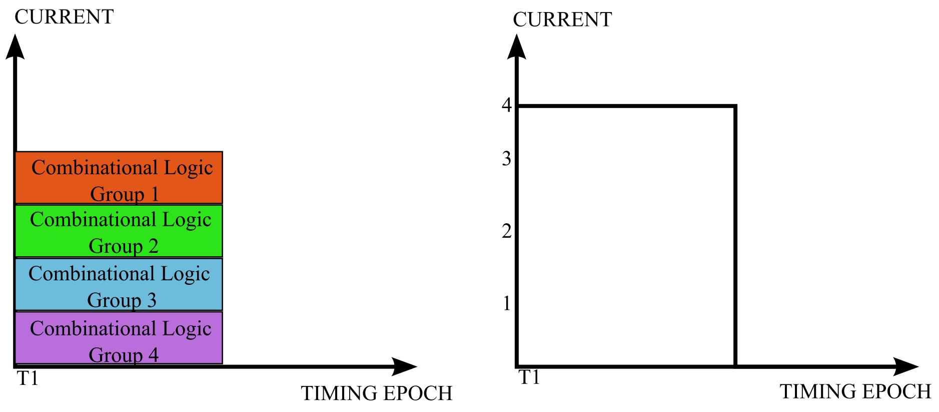

CURRENT
Combinational Logic Group 1
Combinational Logic Group 2
Combinational Logic Group 3
Combinational Logic Group 4
T1
TIMING EPOCH
CURRENT
4
3
2
1
T1
TIMING EPOCH

Figure 4.4. Traditional Clocking and Current Profile

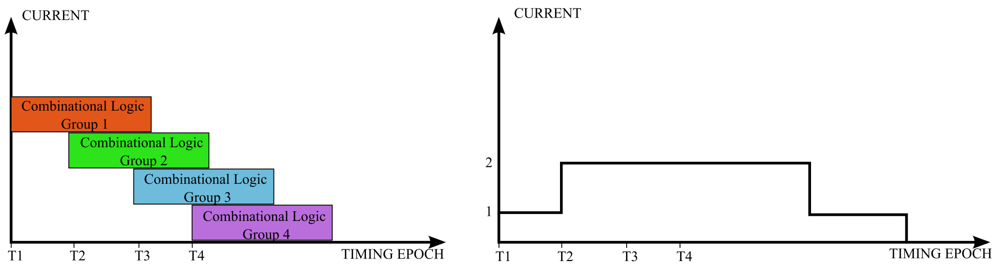

CURRENT
Combinational Logic Group 1
Combinational Logic Group 2
Combinational Logic Group 3
Combinational Logic Group 4
T1
T2
T3
T4
TIMING EPOCH
CURRENT
2
1
T1
T2
T3
T4
TIMING EPOCH

Figure 4.5. Proposed Multi-Phase Clocking and Current Profile

4.3.1 Graph Generation and Timing Analysis

The input behavioral circuit descriptions for the ISCAS 85 benchmarks are converted to the structural netlist by using standard cell libraries using the Nangate standard cell library based on the FreePDK 45nm technology for technology mapping. A graph is then generated from the technology mapped netlist. The graph is connected where the nodes are primary inputs, primary outputs, logic gates and nets as nodes and stems are the edges connecting the nodes. Since the experimentation is targeted towards combinational circuits and thus the logic is non-sequential, the circuits have no feedback logic thus making the graph acyclic. This technique can be expanded for sequential designs which will be investigated in future research. After all the nodes and edges that belong to the circuit graph have been created, the source and sink dummy nodes are added and connected to the primary inputs and outputs respectively.

An acyclic graph is generated so that we are able to traverse the graph in linear fashion with respect to the number of nodes for timing analysis. Timing analysis needs to be performed on the structural netlist generated after technology mapping. Timing analysis will identify slacks of various nodes in the circuit and the cumulative slacks on the various circuit paths. Timing analysis is done outside of the system tools so that their is more freedom to use a separate technique in the delay annotation of the circuit for further analysis.

Timing analysis is performed by executing the As Soon As Possible (ASAP) and As Late As Possible (ALAP) scheduling algorithms on the circuit's graph. Each node structure holds the instance name given, its node type, its strength, load capacitance, delay, and its inputs and outputs. The type of a graph node can be primary input, primary output, net, gate, source or sink. The strength is based on the type design cells used to synthesize the design while the load capacitance and delay values are all obtained by the technology library. These attributes make it easier for graph assembly. After generating the graph, each gate node is notated with its respective delay and load capacitive values. To expedite this annotation, the input capacitances, load capacitances and delay values of the combinational logic cells are extracted from the technology library to a file. The capacitive loads are annotated for each node and using this load capacitance value, the nearest delay value associated with the capacitance value is given to that node.

Now the graph has been fully annotated with its delay values, timing analysis can be performed on the circuit's graph. Timing analysis is done by performing the Depth-First-Search (DFS) algorithm from the source node to obtain the ASAP timing values, and from the sink node to obtain the ALAP timing values. By taking the difference between these values, the amount of slack delay that is on each path can be determined. Those paths with zero slack are those along the critical path.

4.3.2 Path Clustering

Algorithm 2 Cluster_Paths ()

```
Define number of nodes as n;
Define number of clusters as k;
balanced = $\frac{n}{k}$;
for each primary input $I$ do
    max_delay[$I$] = Delay of longest path connected to input $I$;
    while delay[i] $>=$ cluster_times[j] do
        Decrement j if j!= 0;
        while cluster_count $>$ balanced do
            if j == 0 then
                break out of while loop;
            else
                Go to previous cluster by decrementing j by 1;
            end if
        end while
        Increment cluster_count[j] by 1;
    end while
end for
```

A delay value is associated with each primary input so that our clustering algorithm only has to operate on the inputs to assign a timing value for all the paths connected to it. Since the timing for the primary input correlates with all the connected paths, the worst case timing value will be associated to some primary input. Therefore, all the paths connected to each primary input are traversed and the maximum delay of the paths connected to the primary input is calculated. This is accomplished by running a dynamic-programming scheme that calculates the maximum delay value connected to each graph node. Once the maximum delay value has been annotated for all the internal nodes, only the maximum value delay of all the internal nodes connected to the primary input needs to be associated with the primary input.

Algorithm 2 illustrates the proposed path clustering procedure. Since all the paths are connected to some primary input, the path clustering algorithm deals with clustering the primary inputs based on the maximum delay value associated with them. Each cluster's delay is mapped to a fraction of the clock period which equals the maximum delay of all of the paths. The primary input are placed into the cluster where the delay of a primary input is less than the delay of a cluster . For each primary input placed into the cluster, the size of the cluster is increased. With all of the paths associated with a primary input, the run time is significantly reduced since the number of primary inputs is usually significantly less than the number of combinational paths within a circuit. This is the main step of the clustering algorithm.

4.3.3 Clustering Algorithm

In performing path clustering, the delay values between each pair of graph nodes are initially calculated. These values are stored in an *n x n* matrix where *n* equals the total number of graph nodes. If there is no path between two nodes, their delay values are assigned -1. Now that we have the data efficiently stored, the maximum delay values connected to each primary input can be found using a memoized algorithm. After associating these max delay values with their corresponding primary inputs in a vector array, the clustering algorithm given in Algorithm 3 is executed.

The clustering algorithm takes the set of delay values associated with the primary inputs along with the primary inputs, the clock period and the user-defined cluster number value as inputs. The number of inputs are divided by the number clusters so that the ideal number of inputs that should be placed in each cluster to make each cluster balanced is defined. The clusters must be load balanced as much as possible so that the change in current over time ($\frac{di}{dt}$) is as close to zero as possible. Given the ideal case, where all of the primary inputs have equal input capacitances, equal logic paths connected to them, and equal switching activity, it is shown that all the primary inputs consume the same amount of power. From the worst case, where all the primary inputs (the paths) are switching simultaneously, it is observed that the peak-power value will be minimized given that the clusters are as balanced

Algorithm 3 Clustering_Algorithm (**k** clusters, **n** num of PI, **CP** clock period, **P** set of PIs, **D** set of max delays for each PI)

```
array[k] = 0;
// Ideally we would like to have a uniform distribution in each cluster
balanced = $\frac{n}{k}$
tc = critical delay value;
start time for $C_0$ is 0;
//Initial run
for i = 0 to k-1 do
    Set the start time for cluster i to (i $\times$ balanced)* tc;
end for
Cluster_paths();
array[i] = # of inputs placed in cluster i
OPT = n;
//Calculate Cluster factor
Cluster_factor for cluster i, $C_i$ = OPT - array[i];
//Calculate Total factor
Total_factor for circuit = $\sum_{j=0}^{k} C_j$
while Total_factor $>$ OPT OR loop has not ran for $\alpha$ iterations OR CF != PF do
    Store current total factor into PF
    Choose the largest imbalanced cluster j and change the start time $\frac{c_{j-1}+c_j}{2}$.
    Re-calculate Total and Cluster Factors;
    Store current total factor in CF;
    if PF $<$ CF then
        Previous configuration was better. Choose the next largest imbalanced cluster and
        repeat above.
    else
        Current configuration is better. Repeat While from here
    end if
end while
if array[i] == 0 for some i then
    Delete cluster i;
end if
```

as possible. The clock period is divided by the number of clusters so that initially, all clusters have equal sub-intervals for each cluster. The clock arrival times for each cluster have been divided into percentages of the clock period as well. For example, if there is a cluster_number = 4, then the clock signal of cluster 1 arrives the same time as the original clock period, the clock signal of cluster 2 arrives at time which is equal to 25% of the clock period after the original, the clock signal of cluster 3 arrives 50% after the original and the clock signal of cluster 4 arrives after 75% of the original clock period has past. From this example, one can see that the shorter delayed paths should be placed in the latter clusters, and the longer delayed paths should be placed in the beginning clusters.

The algorithm places each primary input in a cluster by comparing the delay widths of the cluster with the maximum delay associated with the primary input. The comparisons begin from the last cluster for two reasons: Firstly, in this optimized performance era, we expect that many of the delayed paths are close to the delay of the critical path; the second reason is that it is preferred to place as many primary inputs in the latter clusters as possible so that if any of the latter clusters become imbalanced, primary inputs can be shifted to an earlier cluster without generating any timing violations in the circuit. After placing each primary input into a cluster, we calculate the cluster factor for each cluster which is given by $||$*cluster_size - balanced_size*$||$, where *cluster_size* is the number of primary inputs placed in the cluster and *balanced_size* is the ideal number of evenly distributed primary inputs over the number of clusters. If each cluster has a factor of zero, then the configuration is balanced, and the algorithm exits with the current cluster configuration. If all the clusters have a nonzero cluster factor, then the cluster that has the largest cluster size chosen. If the cluster is not the first cluster, then we can always try to shift some of its inputs to an earlier cluster, as long as it does not cause the other clusters to become imbalanced. If this alternative is not possible, then the start time of the cluster is shifted to the midpoint of the previous adjacent cluster start time and its original start time. After adjusting the start time of the cluster, re-clustering of the primary inputs is performed, to see if a more balanced configuration can be obtained. This process is continued until a balanced configuration is generated, or a saturation point is reached where no improvement is occurring. To refrain from the algorithm running for some large finite amount of time, the algorithm stops after α number of iterations. This α constant can be configured to the user specifications.

The objective of the clustering algorithm is to cluster the primary inputs of the circuit such that the peak power is minimized for the circuit. The run-time of the algorithm is dominated by graph generation and the clustering algorithm. Given that *N* represents the total number of graph nodes and *E* represents the total number of edges in the graph, the graph generation is dominated by generating edges for all *N* nodes which takes O(*NE*) time. The clustering algorithm is dominated by Cluster_Paths() function which runs in O($N + E$) time. Thus the clustering algorithm runs in O($\alpha\dot{(}N + E)$). This gives a linear runtime as long as $\alpha < N + E$.

Peak power usually occurs during the transition period of the clock signal thus it is intuitive to reduce switching activity during this interval. During this transient period, many of the combinational module inputs are switching causing a large draw in the current from the power supply at once and then a decrease as the circuit begins to stabilize. The algorithm attacks this problem by balancing the number of inputs being executed, which in the ideal case would balance the current drawn by the power supply. By grouping the primary inputs in a balanced manner, the different paths of the combinational logic can be executed throughout the entire clock period rather than at the same time instance. By balancing the current drawn during the transition time of the clock, the instantaneous power consumed by the circuit is reduced which reduces the peak power of the circuit.

One benefit of this algorithm is that it gives a time efficient solution to the peak-power problem. Instead of addressing the problem at the transistor level where power accuracy is higher, the circuit is optimized at the synthesized-netlist level during the design automation process. Although accuracy may be not be as high at this level, solution time efficiency is better due to the lower level of complexity. Given that technology mapped netlists are optimized along with the Synopsys tools, power analysis values are closely relative to the actual power values of the individual circuits.

Table 4.1. Peak Power Reduction (PPR) Percentage in Benchmark Set

Benchmark	2 clusters	3 clusters	4 clusters	5 clusters	6 clusters
	PPR(%)	PPR(%)	PPR(%)	PPR(%)	PPR(%)
c432	26.15	47.55	22.78	46.48	49.85
c499	41.21	44.25	49.20	54.83	71.78
c880	6.55	30.25	42.66	30.70	N/A
c1355	40.18	38.40	45.93	49.97	68.99
c1908	14.22	34.54	35.21	46.28	N/A
c2670	-11.64	17.90	28.13	28.33	34.45
c3540	27.23	38.32	33.06	48.68	54.00
c5315	6.70	10.91	34.68	21.02	35.06
c6288	-24.78	-23.33	9.97	14.91	18.01
c7552	-9.02	2.37	-4.08	-1.90	31.72
divunit	2.63	2.10	4.24	18.13	21.31
fpadd	3.23	-8.62	10.08	13.81	4.84
mul32	7.75	-5.35	8.78	1.67	4.96
oc8051_alu	12.33	15.31	34.20	35.43	38.42
parallel_find	14.27	37.11	37.87	47.95	31.32
average	10.47	18.78	26.18	30.42	36.62

4.4 Experimental Setup

The simulation setup is centered around Synopsys DesignCompiler and Primetime-PX power extension software suites and simulations are performed on the *ISCAS 85* benchmarks and a few circuits from OpenCores [83]. The first step is to synthesize each benchmark using the technology standard cell library. Technology mapping is carried out using *Synopsys Design Compiler* and the 45nm Nangate Open Cell Technology Library. A small subset of the library's combinational cells are used to synthesize a simpler design. The technology mapped netlist is given as input to the proposed path clustering algorithm. The

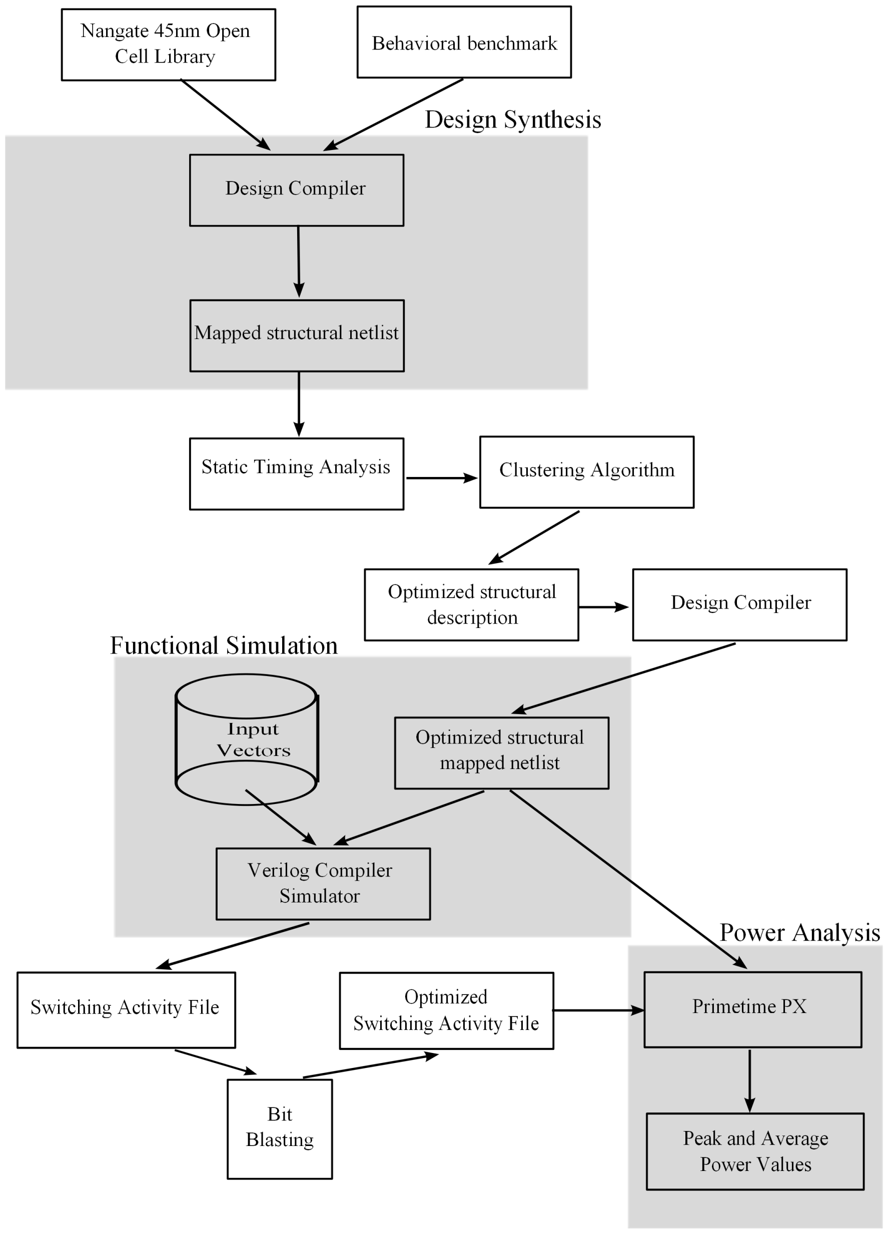

Nangate 45nm Open Cell Library
Behavioral benchmark
Design Synthesis
Design Compiler
Mapped structural netlist
Static Timing Analysis
Clustering Algorithm
Optimized structural description
Design Compiler
Functional Simulation
Input Vectors
Optimized structural mapped netlist
Verilog Compiler Simulator
Power Analysis
Switching Activity File
Optimized Switching Activity File
Primetime PX
Bit Blasting
Peak and Average Power Values

Figure 4.6. Process Flow for Power Analysis

clustering algorithm was coded in C/C++ programming language. The generated netlist is converted into an acyclic graph structure where each node represents a combinational cell module and each edge represents the nets. The graph is annotated with the load capacitances and cell delays for timing analysis. Path characterization and timing analysis are performed with the ASAP and ALAP algorithms. Path clustering is then executed to cluster the paths after which, Verilog source and script files necessary for re-synthesis, simulation and power analysis are generated.

Resynthesis is carried out using *Design Compiler*. In this stage, flip flop modules are connected to the primary inputs so that the execution of the test vectors can be synchronized. The phase-shifted multiple clocks with respect to the domain clock can be generated in different ways. The generation of the multiple phase-shifted clocks can be done by use of a counter and decoder logic module based on a clock divider strategy. The clock lines are buffered to drive the clock inputs on the flip flop modules to reduce the amount of internal short circuit power generated by the cell modules due to large input transition times. Given the resynthesized designs and the standard delay file (.sdf), the gate-level designs are simulated. Alternately, the phase-shifted clocks globally can be generated using delay buffers on the original clock which involves experimentation with the number of buffers, buffer sizes and the capacitive loads on each clock line. It is observed that in a few cases, when buffers are inserted at the HDL level, the DesignCompiler tool tends to remove some of the buffers during the optimization phase of logic synthesis affecting the amount of phase shifts needed in the clocks for our experiment. These needed to be checked in each case. The other observation was that the clock-power overhead was higher in the first case which shall be discussed later in this section. Thus, a second approach to implement the phase-shifted clock was chosen.

Gate-level simulation is performed for power analysis in the next stage. The gate-level designs are simulated using *Synopsys Verilog Compiler Simulator* using some input vectors as given in [80] and generating the other input vectors randomly. Each benchmark was simulated for a set of 10,000 input vectors. The simulator gives the switching activity in the form of a vcd file. Bit blasting was performed on the switching activity file so that each net on the wired bus has its individual switching activity annotation. The bit-blasting operation is performed for more accurate power analysis. Power analysis was performed on the designs using *Synopsys Primetime-PX*. The resynthesized gate-level netlist and the generated vcd files are passed into the software to perform the analysis. Power analysis is performed every 1ps to get maximum accuracy. The experiments were conducted on a 16-core processor server with 98GB of RAM. A 64-bit executable was necessary for some of our benchmarks due to the growth in size of their graphical representations. Through efficient coding optimizations, we were able to optimize our code to run most of the benchmarks on a 32-bit processor with 4GB RAM. The detailed design flow is shown in Figure 4.6. The specifications for the benchmarks tested are listed in Table 4.2. The clock period was calculated by adding the maximum delay to the maximum clock-to-q and setup time delay for the Flip-flop cell module in the technology library. The Phase shifting technique was initially performed using a counter-decoder logic block to perform the phase-shifted clock arrival times. This method requires the generation of a faster clock created by a clock multiplier. The number of clusters used has a direct relationship with the speed of the generated clock, thus a limit where the speed of our multiplied clock is higher than the flip-flop cell's propagation delay was reached, thus precluding counter logic operation. Due to this limitation, implementation of the phase-shifted clocks were carried out using buffer delays.

4.5 Simulation Results

In this section, the results obtained from the simulations of the ISCAS 85 combinational benchmark and OpenCores circuits are discussed. Also, a single-clock divider, single-

Table 4.2. Benchmarks' Functionality, Clock Period and Execution Time

Benchmark	Functionality	Clock Period(ns)	Exec. Time(μs)
c432	Controller	0.949	6.89E+05
c499	SEC circuit	0.969	5.04E+05
c880	8-bit ALU	0.879	1.11E+06
c1355	32-bitSEC circuit	0.969	1.12e+06
c1908	16-bitSEC/DED circuit	0.959	7.34e+05
c2670	ALU and controller	1.17	8.02E+05
c3540	8-bit ALU	1.12	9.18E+05
c5315	9-bit ALU	1.18	2.09E+06
c6288	16x16 multiplier	2.31	6.18E+06
c7552	32-bit adder/Comparator	1.13	3.65E+06
div_unit	Single-Clock Divider	2.07	1.06E+08
fp_add	Single-Precision FP Adder	1.52	4.49E+07
mul32	32-bit multiplier(LEON)	2.15	9.29E+08
oc8051_alu	Intel 8051 core ALU	0.435	5.84E+06
parallel_find	Max/Min Binary Tree finder	1.48	1.44E+08

precision floating-point adder, 32-bit multiplier from the LEON processor [81], Intel 8051 ALU core, and min/max binary tree finder were included to the set. The simulations were performed for two configurations in the case of each circuit: (i) traditional clock tree driven circuit (ii) the circuit after clustering being driven by multiple phase-shifted clock lines with additional buffering as needed. The maximum execution times for each benchmark is given in Table 4.2. The peak-power reduction due to the proposed clocking scheme are given in terms of percentage change in Table 4.1. The absolute peak power values for the traditional (sometimes referred to as original case) and the clustered clocking schemes along with the absolute values for the power overhead due to the clocking circuits are plotted for each benchmark circuit in the form of bar graphs in Figures 4.8-4.12. From our results, the majority of the circuits have a monotonic peak-power reduction with the largest peak-power reduction at 72%. In general, the amount of peak-power reduction (PPR) increases as the number of clusters increases. The circuits with the best PPR values are the error-correcting circuits (c1908, c1355, c499) followed by the ALU designs (c2670, c3540, c5315, and c880). The divider and multiplier circuits showed the least amount of peak power reduction which is most likely due to that these circuits are tightly interconnected (correlated) which reduces the amount of slack between the internal combinational paths. Note that some cluster configurations have negative PPR values which means that the peak-power value increased with respect to the original circuit configuration. There are also some instances where the benchmark circuits do not exhibit a monotonic or regular pattern in the decrease in peak power. It was observed that these two issues were caused by the following:

1. The internal circuit path delays were close to the critical path delay which gives very little slack for clustering

2. The cluster configurations are well balanced, but the switching activity across the clusters differed drastically causing a sharp change in current demand.

Results were generated for up to six clusters after which timing violations due to the incurred buffer delays between the cluster clock time periods that extend past the next rising clock edge of the original clock were observed. In the six cluster configuration, c880 and c1908 had timing violations.

Given the dynamic power consumption

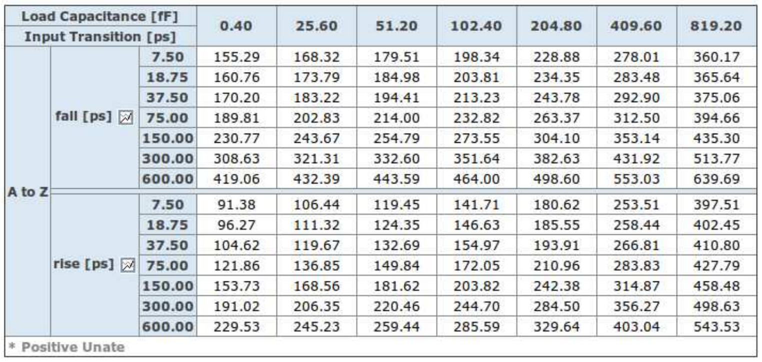
$$P_{dyn} = C_L V_{dd}^2 P_{0\to1} f$$

where C_L is the load capacitance V_{dd} is the supply voltage, $P_{0\to1}$ is the probability that the clock event changes the output of a gate and f is the frequency, the only factor that varies in the clustered configuration and the original configuration is $P_{0\to1}$. Contributors to our power overhead are the buffers inserted as well as the clock power for driving the clusters. The clock-power overhead depends on the method used for generating the phase-shifted clocks and is somewhat simulation dependent. Clock-power overhead is considered as the additional power overhead incurred due to generating the phase-shifted clocks and this was much less for the second approach discussed above with buffers than the decoder-counter approach.

Table 4.3. Average Power Overhead Analysis

Benchmark	# of Buffers	Total Power(mW)	Power Overhead(%)
c432	3	1.41	2.8
c499	3	1.24	4.7
c880	3	2.74	2.1
c1355	3	1.42	4.1
c1908	3	3.47	1.7
c2670	4	5.05	1.5
c3540	4	6.24	1.2
c5315	4	8.07	1.0
c6288	7	23.5	0.57
c7552	4	13.1	0.6
div_unit	7	35.2	0.4
fp_add	5	14.8	0.7
mul32	7	118.2	0.11
oc8051_alu	2	3.54	1.1
parallel_find	7	28.1	0.3
MINIMUM	2	1.24	0.1
AVERAGE	4.4	17.7	1.5
MAXIMUM	7	118.2	4.7

Initially, while performing the simulations, it was observed that the estimated power values were increasing significantly with the multiphase clock network due to internal power consumption of the cell modules and the internal power is basically the short circuit power as described in the manuals for Primetime-PX. This internal power increase was due to the large input transition time caused by charging the net capacitance of the clock signal inputs connected to the flip flop modules. Since the clock signals are generated by the phase-shifting logic, buffers must be inserted on the clock drivers which in turn will minimize this input transition time. To obtain more accurate power estimation, it was noticed that the gate-level simulation and switching-activity estimation give significantly more accurate results than the RTL process for power estimation. In Primetime-PX, power estimation is



Load Capacitance [fF]			0.40	25.60	51.20	102.40	204.80	409.60	819.20
Input Transition [ps]									
A to Z	fall [ps]	7.50	155.29	168.32	179.51	198.34	228.88	278.01	360.17
		18.75	160.76	173.79	184.98	203.81	234.35	283.48	365.64
		37.50	170.20	183.22	194.41	213.23	243.78	292.90	375.06
		75.00	189.81	202.83	214.00	232.82	263.37	312.50	394.66
		150.00	230.77	243.67	254.79	273.55	304.10	353.14	435.30
		300.00	308.63	321.31	332.60	351.64	382.63	431.92	513.77
		600.00	419.06	432.39	443.59	464.00	498.60	553.03	639.69
	rise [ps]	7.50	91.38	106.44	119.45	141.71	180.62	253.51	397.51
		18.75	96.27	111.32	124.35	146.63	185.55	258.44	402.45
		37.50	104.62	119.67	132.69	154.97	193.91	266.81	410.80
		75.00	121.86	136.85	149.84	172.05	210.96	283.83	427.79
		150.00	153.73	168.56	181.62	203.82	242.38	314.87	458.48
		300.00	191.02	206.35	220.46	244.70	284.50	356.27	498.63
		600.00	229.53	245.23	259.44	285.59	329.64	403.04	543.53
* Positive Unate									

Figure 4.7. Snapshot of Propagation Delay Characteristics of the Nangate Buffer Cell [82]

computed by taking the average of the power consumption within a user-defined sampling interval, but in the RTL process, this sampling interval can only be defined relative to the clock period. By performing gate-level simulation, a much smaller sampling interval was possible (every 1 ps) and it was possible to get significantly more accurate power-estimation values. It is also useful to perform bit-blasting on the switching activity file so that each net within a bus is given its own switching-activity profile. It was also observed that buffer insertion resulted in a decrease in peak power with a small average-power overhead.

Next, the power overhead in generating the phase-shifted clock using delay buffers was analyzed. Given the largest clock period from Table 4.2, one can note that the worst case where the largest amount of delay needs to be inserted from the chain of buffers. Given the propagation delay characteristics shown in Figure 4.7, if the buffer was used with the least amount of propagation delay, it is only necessary to insert seven buffers maximum to generate the delays necessary for the phase-shifted clocks. The power overhead percentage is shown in Table 4.3. The total power column represents the amount of power consumption for each benchmark in its original configuration. From our simulation results, it is worthing noting that the phase-shifted clock average-power is small, averaging only 1.5% over the entire suite of benchmark circuits. It is important to note that this is an extreme upper bound for our smaller benchmark and one should expect a much smaller percentage in overhead in the actual(physical) implementation.

The benchmark-circuit area overhead attributable to the addition of the phase-shifted clock(buffer chain) is shown in Table 4.4. Area overhead is negligible, averaging only 1.6%, since the area of the BUF_X32 buffer in our 45-nm technology library is only 2.394 μm^2. Variance in the area overhead is possible due to the technology library, cell selection, and buffer sizing necessary to generate the delay values, but since the calculated area was done using the largest cells possible, the area overhead should not increase within this technology size(45nm).

Table 4.4. Area Overhead Analysis

Benchmark	# of Buffers	Total Area(μm^2)	Area Overhead(%)
c432	3	129	3.7
c499	3	289	2.5
c880	3	286	2.5
c1355	3	289	2.5
c1908	3	316	2.3
c2670	4	488	2.0
c3540	4	795	1.2
c5315	4	1160	0.8
c6288	7	1946	0.9
c7552	4	1347	0.7
div_unit	7	2359	0.7
fp_add	5	416	2.9
mul32	7	5347	0.3
oc8051_alu	2	703	0.7
parallel_find	7	5001	0.2
MINIMUM	2	129	0.2
AVERAGE	4.4	1391	1.6
MAXIMUM	7	5347	3.7

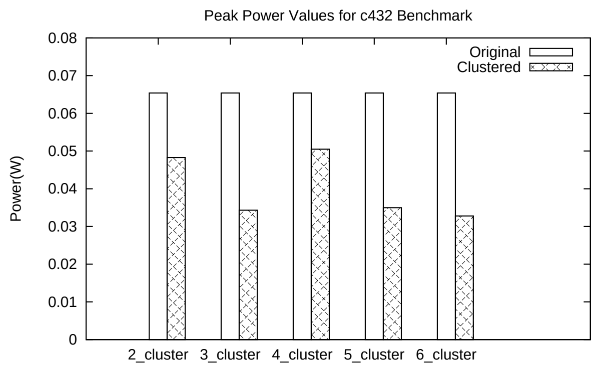
How the proposed peak power scheme affects the RMS current of the benchmark set was also investigated. To calculate the RMS current for the given power numbers, results from the fsdb file generated by Synopsys Primetime-Power Extension were extracted and converted into an text file. The generated text file is then passed into a C++ program designed to compute the RMS current values. For computing the RMS current we use the following formula:

$$I_{rms} = \sqrt{\frac{\int_0^T I(t)^2 dt}{T}} \tag{4.2}$$

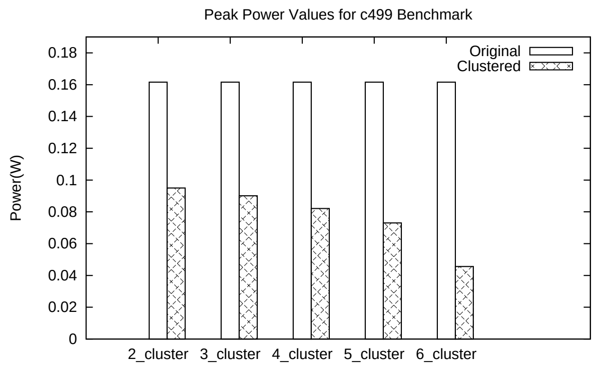
To obtain the current values we divided the calculated power values by the nominal supply voltage which in our case was 1.2V. To calculate the integral function we use the Riemann Sum equation to get fairly accurate calculations. As you may recall, the integral of an function can be calculated using the following equation:

$$\sum_{\Delta t \to 0} f(t)\Delta t \tag{4.3}$$

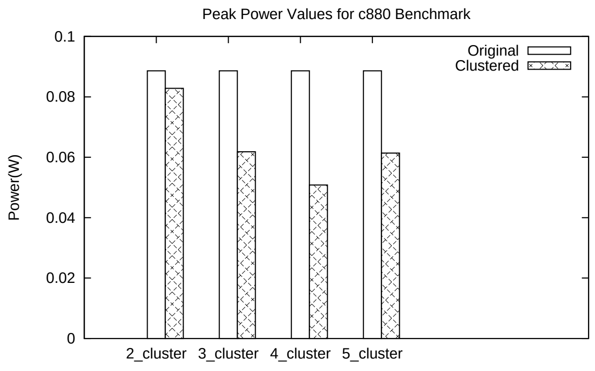
Since the simulation annotates values every 1ns, it is expected to get fairly accurate numbers. Execution times were calculated using the gettimeofday() function which is part of the sys.time.h header library. Using this function, the simulation time with accuracy down to the microsecond is able to be calculated. RMS percentage reduction is shown in Table 4.5. Results show a monotonous increase in percentage reduction of RMS current with an average reduction of 49% across the defined cluster configurations. Thus, it is shown that benchmark circuit paths that have been clustered using our technique generates a lower RMS current demand, which gives a reduction in the peak power draw from the power supply.



(a) c432 Power Results

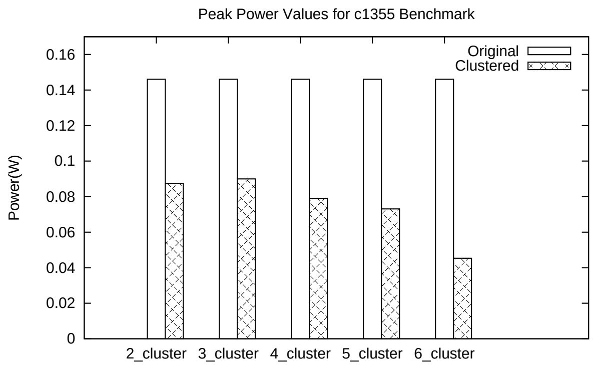


(b) c499 Power Results

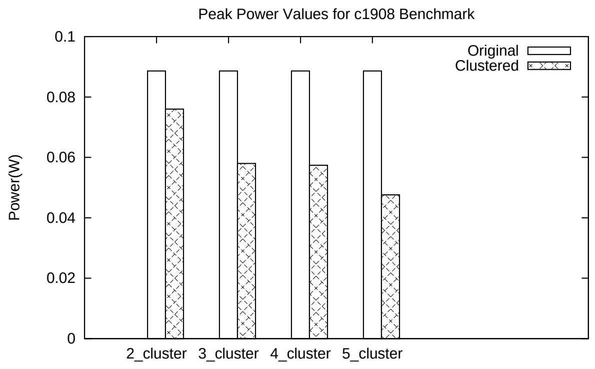


(c) c880 Power Results

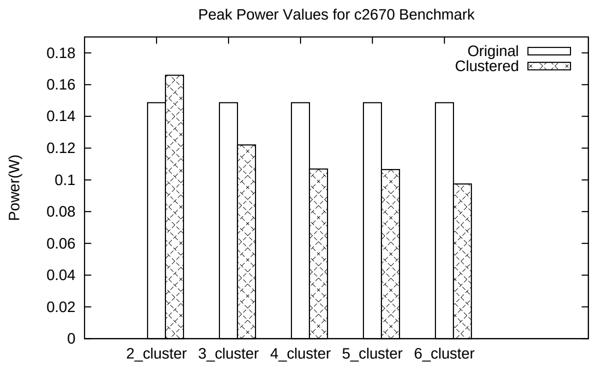
Figure 4.8. Peak Power and Average Power Values for ISCAS 85 Benchmarks for c432, c499, and c880



(a) c1355 Power Results

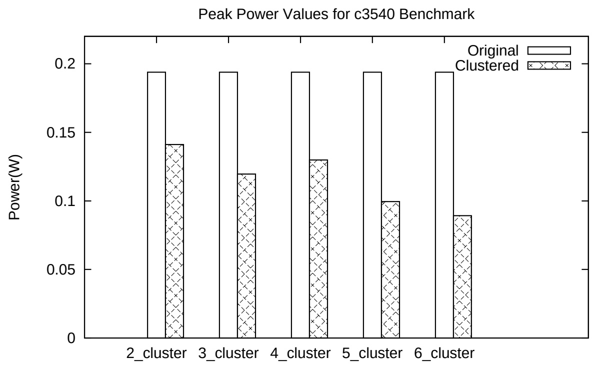


(b) c1908 Power Results

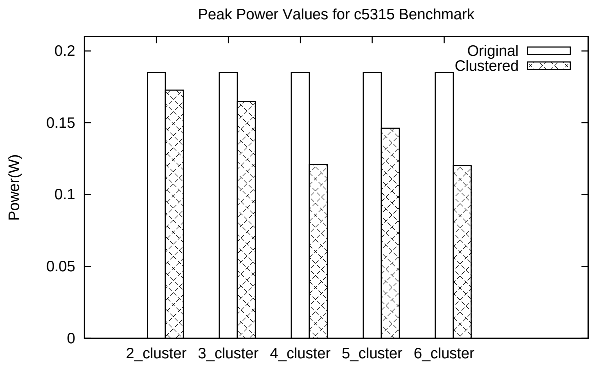


(c) c2670 Power Results

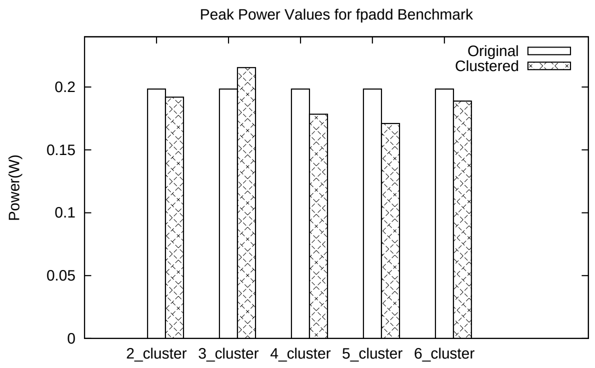
Figure 4.9. Peak Power and Average Power Values for ISCAS 85 Benchmarks for c1355, c1908, and c2670



(a) c3540 Power Results

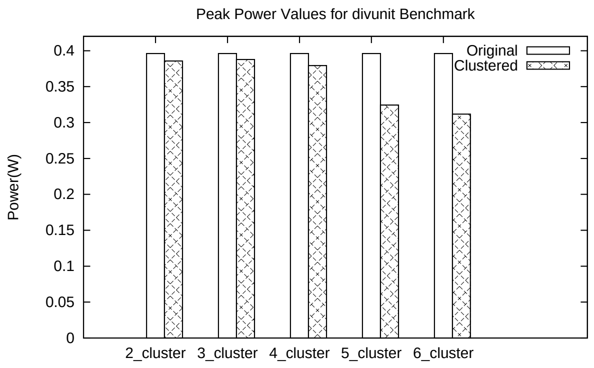


(b) c5315 Power Results

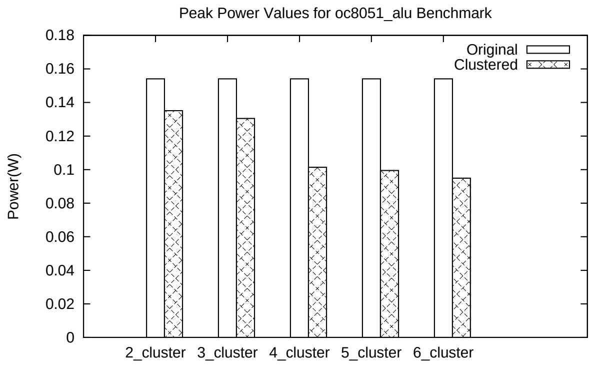


(c) fpadd Power Results

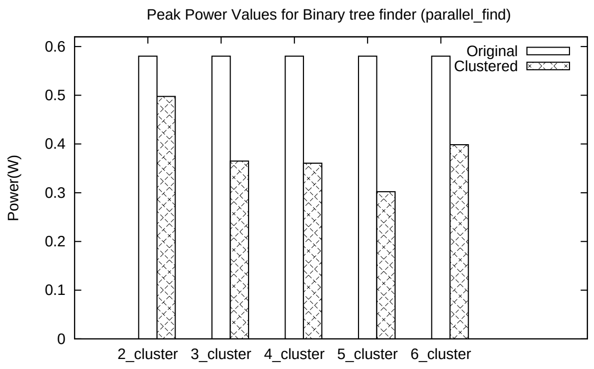
Figure 4.10. Peak Power and Average Power Values for ISCAS 85 Benchmarks for c3540, c5315, and fpadd



(a) divunit Power Results

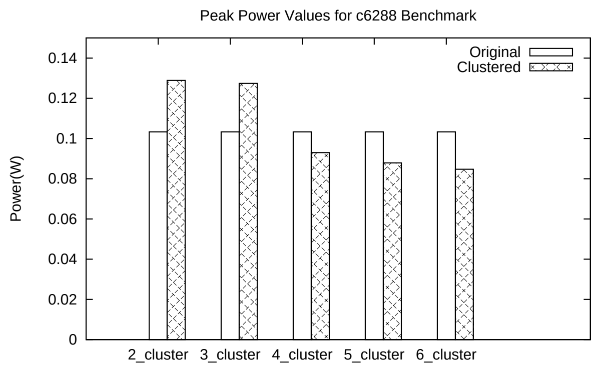


(b) oc8051_alu Power Results

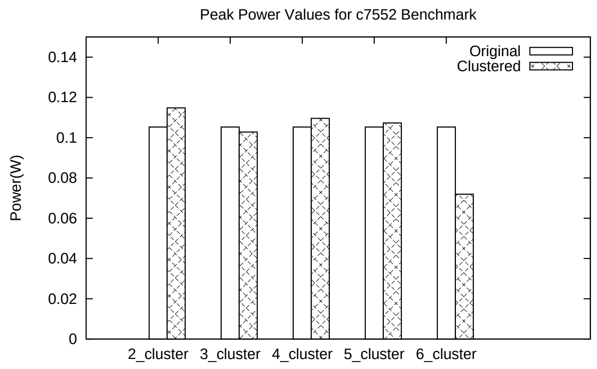


(c) parallel_find Power Results

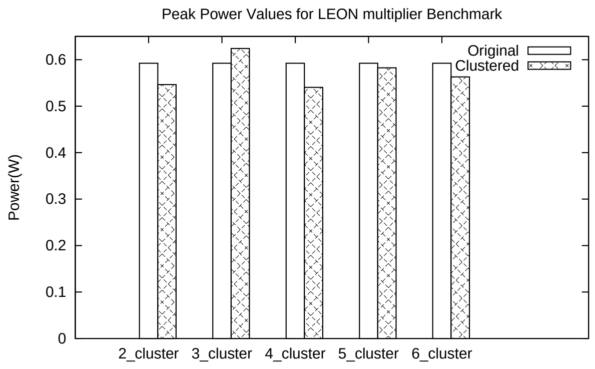
Figure 4.11. Peak Power and Average Power Values for ISCAS 85 Benchmarks for divunit, oc8051_alu, and parallel_find



(a) c6288 Power Results



(b) c7552 Power Results



(c) Mul32 Power Results

Figure 4.12. Peak Power and Average Power Values for ISCAS 85 Benchmarks for c6288, c7552, and mul32

Table 4.5. Percentage of RMS Current Reduction in Benchmark Set

Benchmark	2clusters	3clusters	4clusters	5clusters	6clusters
c432	36.12	52.41	65.53	71.44	80.77
c499	32.32	45.07	59.52	69.48	65.56
c880	28.58	42.16	56.48	68.04	N/A
c1355	28.58	42.16	56.48	68.04	64.04
c1908	19.92	46.70	58.42	68.56	N/A
c2670	39.80	47.73	54.68	60.23	65.26
c3540	16.94	23.46	31.84	41.45	45.82
c5315	36.73	49.54	53.05	59.54	63.89
c6288	41.64	51.51	64.79	68.37	77.19
c7552	21.95	25.36	43.50	38.90	59.56
divunit	0.00	-3.34	10.32	13.51	31.64
fpadd	63.51	67.89	73.84	79.33	84.35
mul32	43.89	52.02	56.00	57.77	67.03
oc8051_alu	33.16	51.08	56.66	59.03	60.92
parallel_find	24.36	21.24	35.23	49.37	53.48
average	31.31	41.19	51.41	57.55	64.06

CHAPTER 5

A VARIATION-TOLERANT DESIGN USING DYNAMIC CLOCK STRETCHING

5.1 Introduction

Technology scaling into the nanometer dimensions has made the design of high performance and versatile computing systems feasible. Smaller feature sizes in devices has enabled more integration and performance within the same area. A critical concern however, has been to provide the desired performance with ever diminishing power budgets. The power-performance trade-off has only been exacerbated with the inception of parameter variations in nanometer technology. Parameter variations defines the amount of process deviation due to doping concentration, temperature fluctuations, power supply voltage variations and noise due to coupling. Variations within a design causes variances in power dissipation and clock frequency from the specified target and hence can result in parametric yield loss. Parametric yield is used to define the design's sensitivity to variations. This variation sensitivity factor is expected to cause 60-70% of all yield losses in the impending technology generations [17]. To ensure that designs are tolerable to all possible variations (process, voltage and temperature), circuits are often designed with a conservative margin. These margins are created by increasing the supply voltage and/or by varying the transistor sizes of device structures to account for the uncertainty due to worst-case combination of variations. This method of over-design takes into account a situation that rarely occurs thus the design is not allowed to achieve its most optimal performance.

In this chapter, an approach for dynamic clock stretching by dynamically detecting delay incurred by process variations is discussed. The clock edge is delayed to critical path register cells to accommodate the increased signal propagation delay due to variations. The clock stretching logic captures the signal transition halfway along the critical path into a positive level-sensitive latch. If the signal transition on the critical path is delayed due to process variation, the latch in the detection circuit holds a different value compared to the signal line and a delay-flag is set. Given that T is the clock cycle time, the signal transition is expected to occur before time T/2. The delay-flag, if set, dynamically stretches the clock at the destination register to accommodate the additional variation-induced propagation delay. Thus the clock stretching methodology avoids a mismatch in the data being captured and hence prevents a timing error which allows for the designer to clock the circuit at a faster clocking rate. The detection circuitry needs to be added to the top critical paths and an error signal from any of these paths can stretch the clock in the appropriate destination register. The clock is stretched (the capture edge trigger is delayed) considering both spatial correlations between closely spaced critical path gates and an average variation range as reported, in [12, 50]. Experimental results based on Monte-Carlo simulations on ITC'99 benchmark circuits indicate efficient improvement in timing yield with negligible area overhead. The rest of the paper is organized as follows. In Section 5.2, the construction of the delay detection circuit for efficient clock stretching is explained. Experimental evaluation and results for an example circuit and benchmark circuits are presented in Sections 5.3 and 5.4 respectively.

5.2 Proposed Methodology

The objective is to add the clock stretching logic to the top critical paths in the design. These paths that are chosen are most likely to be affected by the delay effects of variation.

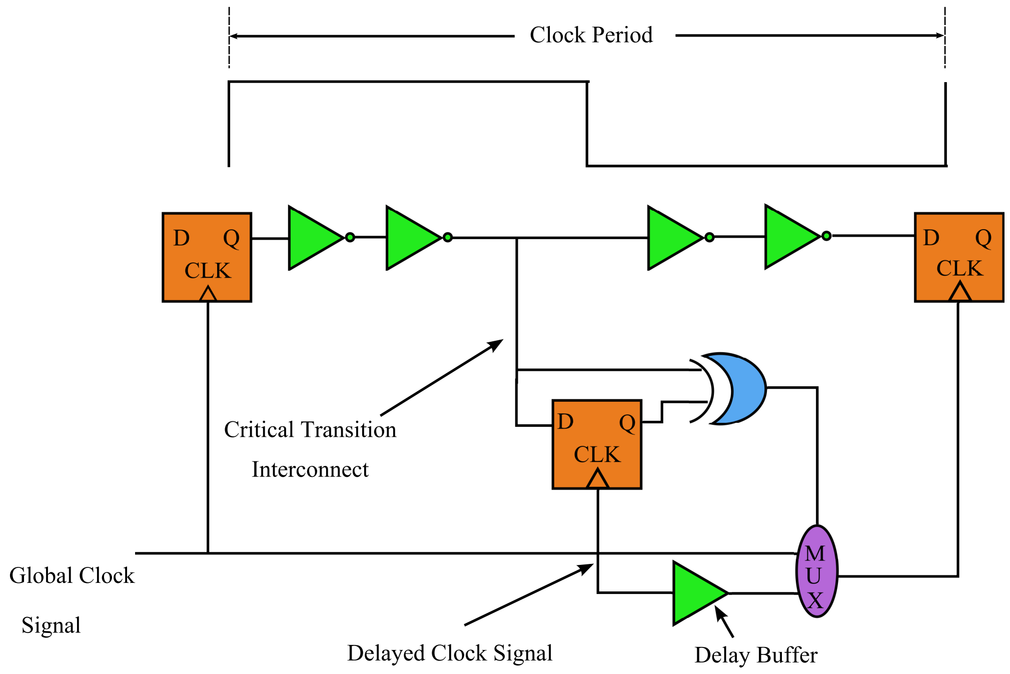


Figure 5.1. Dynamic Clock Stretching for Variation Tolerance

The paths with a delay within 15% of the most critical path are selected as the candidates for dynamic clock stretching [12, 16]. Variations have shown to incur up to 10% delay at of the maximum delay [22]. For the selected critical paths, there must be a critical interconnect transition before the T/2 time instance. At this transition, the signal's value is captured into the latch of our clock stretching logic(CSL) module. To ensure signal integrity, the signal should remain stable for the setup time of the latch. Thus this gives us

$$T_{critical\_trans} \leq T_{\frac{CLK}{2}} - T_{Latch\_setup\_time} \tag{5.1}$$

Simple circuit sizing, buffer insertion, or other incremental changes can be done to create a critical interconnect transition, if one does not exist automatically in any of the top critical paths. Once the transition signal has been captured, the CSL module monitors the transition point for any change in the signal value. If the signal value is different then the value stored into the latch, this is a notification that a delay has been incurred due to variation effects. Thus the output of the XOR selects the stretched clocked input which delays the arrival

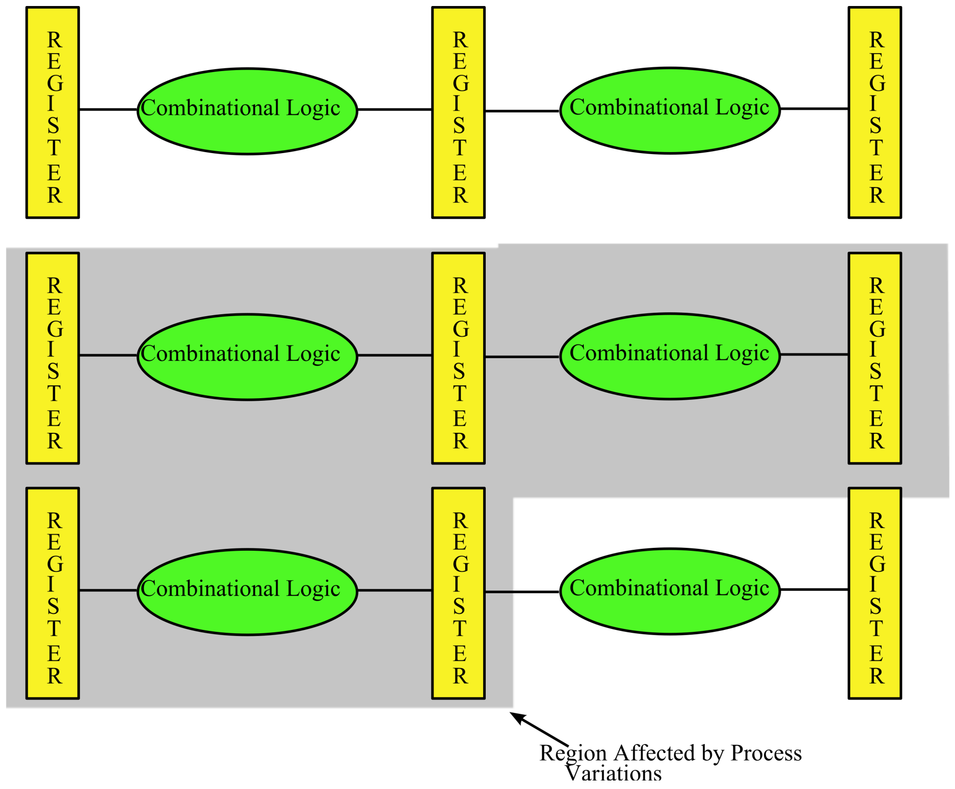


Figure 5.2. Illustration of Spatial Correlation

time of the clock signal to the destination register to accommodate for the delay incurred by variations. If no delay due to variations has occurred, then the path is clocked at the normal clock frequency. With the use of dynamic clock stretching, it eliminates the need for over conservative timing margins for rare cases. This can be most useful for high performance designs.

In the case of short paths following critical paths, the effects of dynamic clock-stretching can be eliminated by gate sizing since the clock is stretched by at most 10%. With multiple design objectives such as power and performance, the existence of short paths are rare in current circuit designs. When two consecutive critical paths are connected, the clock stretching signal should be propagated. This can be taken care of by connecting the previous output signal from the MUX and the current XOR output to an OR gate as illustrated in Figure 5.3 The dynamic clock stretching technique incorporates the spatial correlation

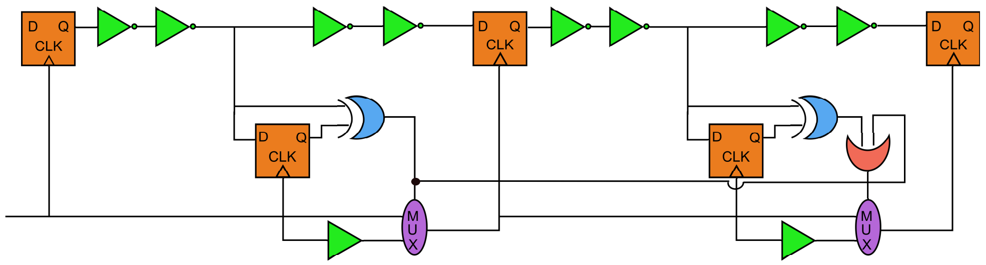


Figure 5.3. Consecutive Critical Paths Implementation

property which states that if an component is affected by variations, then there is a high probability that nearby components are affected as well. Thus we assume that if the combinational logic after the falling edge of the clock are affected variations, then the logic before the falling edge is affected as well. The magnitude of these variations, will be hard to predict, and can be different based on their location in the chip layout. However, the presence or absence of variations can be safely assumed with the property of spatial correlations. In the case where variation delay is less than the setup time of the latch, there could be unstable behavior in the CSL module or there could be a small increase in delay on the path causing incorrect data to get stored in the destination register. With variation delay effects around 5-10% [22], this should be larger than the setup time of the latch within the CSL module. In an effort to confirm, an example was constructed for a critical path with 20 levels of logic at the 45nm technology node level. Delay information was used from the 45 nm technology library for this experiment. The setup clearly confirms the feasibility of the proposed transition capture methodology.

5.3 Experimental Evaluation

To evaluate the proposed methodology, an experimental circuit was simulated using Synopsys Verilog Compiler simulator. The purpose of this simulation is to validate the functionality of the methodology in the presence of variations within the circuit. The efficiency of the methodology is computed using Monte-Carlo based timing yield simulations.

Table 5.1. Description of Symbols in Simulation Snapshot

Clock signals	
glb_clk delayed_clk MUX/Z	Circuit Clock Delayed clock Output clock from multiplexor
Data signals	
in11 eco_net_14 out1	Input data value Critical interconnect transition value Output data value
Clock stretch signals	
d1/q_reg/Q MUX/S	Latch output Select line of multiplexor

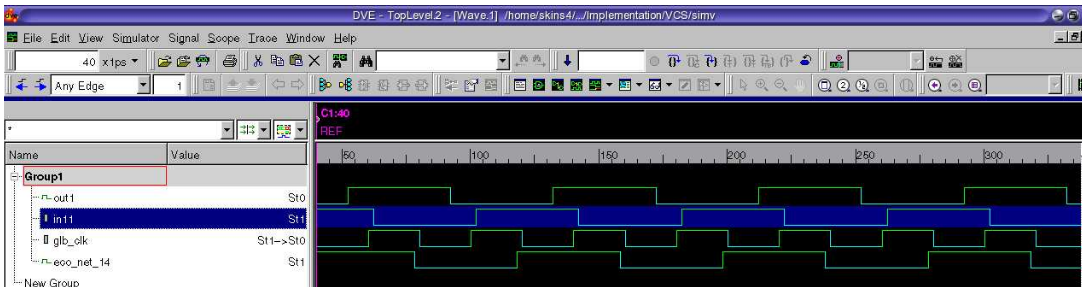

DVE - TopLevel.2 - [Wave.1] /home/skins4/.../Implementation/VCS/simv
File Edit View Simulator Signal Scope Trace Window Help
40 x1ps
Any Edge
C1:40
REF
Name
Value
Group1
out1 St0
in11 St1
glb_clk St1->St0
eco_net_14 St1
New Group
50
100
150
200
250
300

Figure 5.4. Simulation Snapshot of Example Circuit: No Variations; No Clock Stretching

A chain of inverters in between two registers are chosen as the experimental circuit. The chain of inverters are chosen since every interconnect makes a transition and hence the interconnect halfway in the path easily becomes the necessary critical interconnect transition. The clock cycle time is chosen to be the critical path delay of the inverter chain. In addition to the input, output and clock signals, the critical interconnect that transitions from 0→1 (or 1→0) just before the negative edge of the clock is also displayed in the simulation snapshots (Figures 5.4 and 5.5). A simulation snapshot of the example circuit with no variations is shown in Figure 5.4. A brief description of the symbols used in the simulation is shown in Table 5.1. The signal in11 is the primary input (output from the source flip-flop). The signal eco_net_14 is the critical interconnect halfway in the path and out1 is the output signal connected to the destination flip-flop. It can be seen from Figure 5.4, that a 0→1

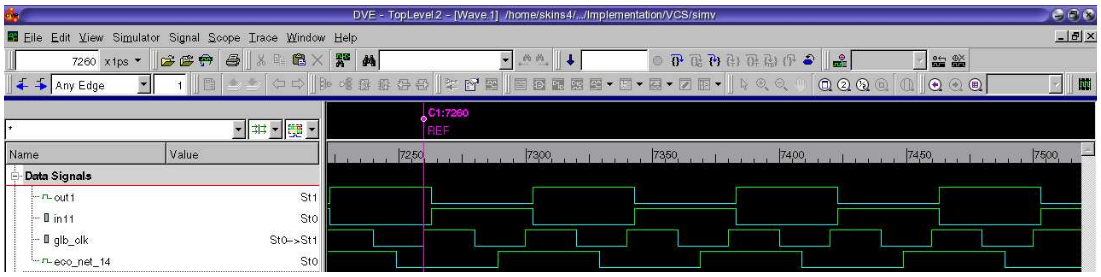


Figure 5.5. Simulation Snapshot of Example Circuit: With Variations; No Clock Stretching

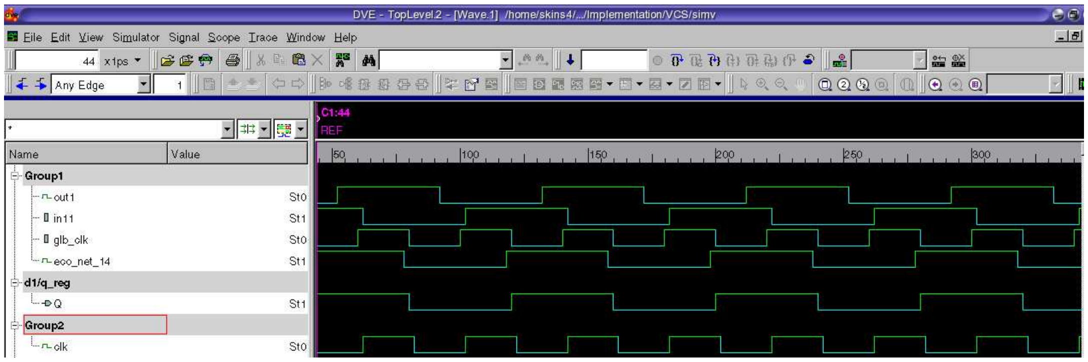


Figure 5.6. Simulation Snapshot of Example Circuit: No Variations; With Clock Stretching

transition on the input signal (in11), initiates a transition on a5 and is captured on the next clock cycle in the destination flip-flop (out1). The simulation snapshot for the same circuit in the presence of delay uncertainty due to process variations is shown in Figure 5.5. In the presence of variations, the same 1→0 transition on the critical net eco_net_14, happens after the negative edge of the clock. The delay due to process variations also caused a timing violation on the output flip-flop (out1), as the (0→1) transition is captured on the subsequent clock cycle.

The simulation snapshot for the experimental circuit with the clock stretching logic is shown in Figures 5.6 and 5.7. The delayed clock (delayed_clk) and the multiplexor output to the destination flop (muxoutclk) is added to the list of clock signals. In addition to the input and clock signals, the CSL snapshots also show clock-stretched signal d1/q_reg/Q and MUX/S(select signal). It can be clearly seen in Figure 5.7, that the delayed clock is sent

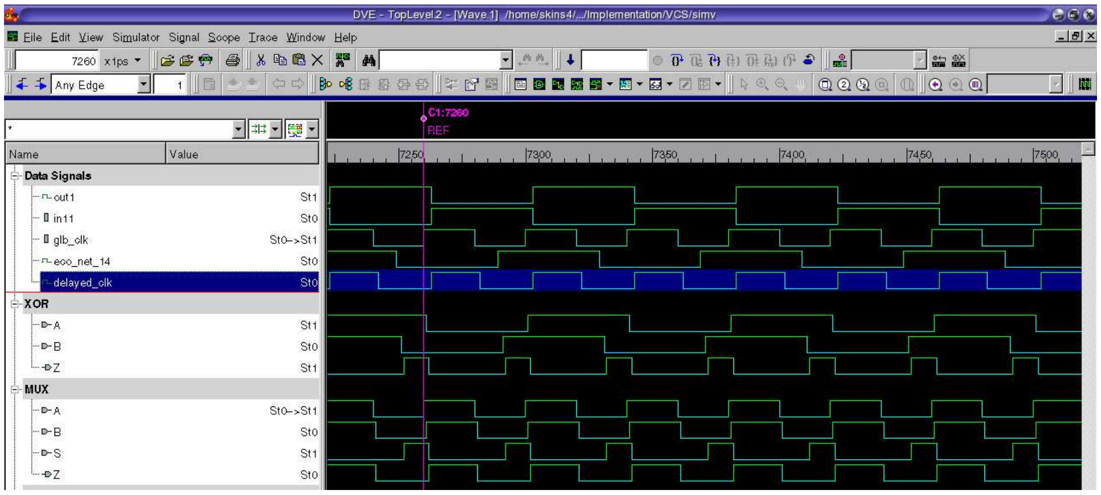


Figure 5.7. Simulation Snapshot of Example Circuit: With Variations; With Clock Stretching

to the MUX/Z pin whenever the transition on the critical interconnect transition happens after the negative edge of the clock. Further in Figure 5.6, we show that in the absence of variations the circuit operates normally and clock (glb_clk) is used at the source and the destination flip-flops.

5.4 Simulation Results

In this section, results for the ITC'99 benchmarks using 65nm technology from [37] are given. The improvements in timing yield for the circuits were estimated using Monte-Carlo simulations. The simulation flow for the timing yield estimation is shown in Figure 5.8. The gate and net delay of the circuit elements were assumed to have a variation range of around 20% from the nominal value. In the absence of real statistical data, it has been pointed out in [50], that it is reasonable to assume a variation parameter value of around 20-25% on the delay due to process variations. The RTL level VHDL netlists were synthesized using the Synopsys Design Compiler. The gate-level Verilog netlist is then placed and routed using Cadence Encounter tool. A timing analysis report (TARPT) file is then generated

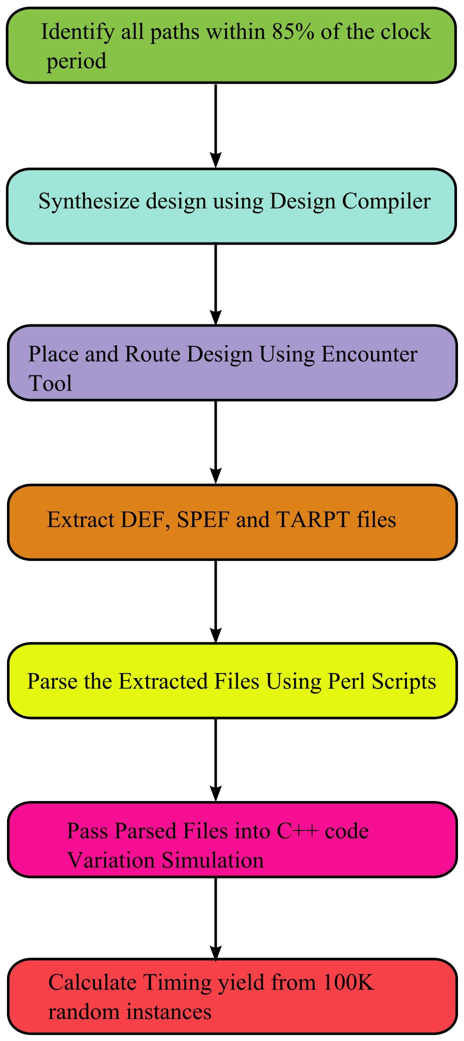

Identify all paths within 85% of the clock period
Synthesize design using Design Compiler
Place and Route Design Using Encounter Tool
Extract DEF, SPEF and TARPT files
Parse the Extracted Files Using Perl Scripts
Pass Parsed Files into C++ code Variation Simulation
Calculate Timing yield from 100K random instances

Figure 5.8. Simulation Flow for Timing Yield Estimation

to identify the critical paths whose delay value is within 15% of the most critical path. A Monte-Carlo simulation framework is created in a C-program environment for the ITC'99 benchmarks with the placed and routed file(DEF), the parasitics file (SPEF), the timing analysis report (TARPT) and the standard cell delay libraries as input. The Monte-Carlo simulation creates 20000 instances of the benchmark with varied delay between nominal and the maximum range to estimate the timing yield. Timing yield in this context, is defined as the percentage of the circuit instances meeting the timing specification. The circuits are tested in two configurations, namely (i) original circuit with variations and (ii) the original

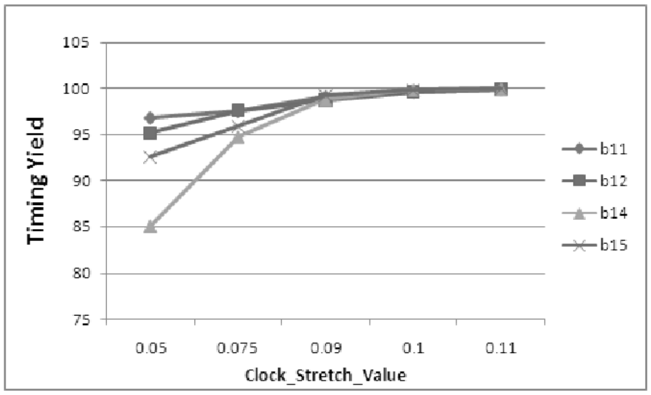


Figure 5.9. Clock Stretch Range Vs Timing Yield [37]

Table 5.2. Timing Yield Results on Benchmark Circuits at 65nm

ITC' 99 Benchmark	No. of Gates	No. of Nets	Near Critical Paths	CSL overhead	Timing Yield without CSL	Timing Yield with CSL
b11	385	322	9	9%	96.5%	99.64%
b12	834	847	16	7.6%	82%	99.65%
b14	4232	4544	65	6.1%	66.2%	99.97%
b15	4585	4716	80	6.9%	48.6%	99.99%
b20	8900	9538	110	4.9%	62.1%	99.95%
b22	12128	13093	118	3.8%	37.0%	99.92%
b17	15524	15911	150	3.8%	56.2%	99.97%
b18	42435	44554	152	1.5%	61.2%	99.99%
Average Percent				5.4%	63.7%	99.9%
Legend: CSL- Clock Stretching Logic Legend: CSL overhead: Percentage of CSL logic area compared to total circuit area Legend: Near Critical Paths: Paths that can violate timing yield with variations						

circuit with the CSL module with variations. The timing specification for the original and the CSL inserted circuit, is assume to be 100% in the absence of variations.

The timing yield results that were presented in [37] are shown in Table 5.2. It can be seen that the average timing yield of the original circuit with the nominal timing specification is approximately 60%. This low timing yield forces circuit designers to add an extra margin in order to improve timing yield. The extra timing margin increases the overhead and/or decrease the performance at which the circuit can operate. The proposed methodology dynamically detects the delay due to variations and adds the extra timing margin only

Table 5.3. Timing Yield Results on Benchmark Circuits under 45nm

ITC’ 99 Benchmark	No. of Gates	No. of Nets	CSL overhead	Timing Yield without CSL	Timing Yield with CSL
b11	1484	767	4.5%	44.409%	96.444%
b12	2190	1252	15.7%	33.260%	91.189%
b14	34158	14654	$<$1%	68.124%	99.999%
b15	20880	8501	2.6%	42.608%	99.680%
b20	70418	30521	$<$1%	73.468%	99.999%
b22	82371	40791	$<$1%	87.915%	99.998%
b17	58284	25433	$<$1%	49.992%	99.999%
b18	200772	85717	1.5%	61.871%	99.334%
Average Percent			3.86%	57.7%	98.3%

Legend: CSL- Clock Stretching Logic
Legend: CSL overhead: Percentage of CSL logic area compared to total circuit area
Legend: Near Critical Paths: Paths that can violate timing yield with variations

when required. The proposed CSL methodology has increased the average timing yield to around 99.9%. The clock was stretched to create an extra timing slack of 10% only if the delay due to process variations are activated in the worst-case critical paths. In the context of timing failures due to short paths, it is crucial to keep the clock stretching range as short as possible. Hence, a simple analysis on selected benchmark circuits was performed to see the impact of clock stretch range on timing yield (Figure 5.9). A smaller value for clock stretch range, for example 5% is shown to impact the timing yield significantly. Thus, the clock stretch range was chosen to be 10% of the clock period. In addition to the timing yield improvement results, the benchmark characteristics (number of gates and interconnects), the number of near critical paths and the area overhead due to CSL logic have been specified in Table 5.2. The proposed CSL methodology also incurs an average area overhead of 5%. The area overhead can be further reduced, if the critical paths are isolated using techniques similar to the previous works on dynamic clock stretching [60, 65].

5.4.1 Simulation Using 45nm Technology Library

The methodology was simulated for the ITC ’99 benchmarks using the Nangate 45nm Open Cell Library. The circuits were synthesized with Synopsys Design Compiler using negative unate logic to make it simpler to extract a critical interconnect transition. The Cadence First Encounter tool was used for placement and routing. Encounter generates the DEF, SPEF and TARPT files needed for further processing. The necessary information is extracted from the DEF, TARPT, and SPEF using Perl scripts and is passed into a C++ simulation program. For each gate, the names, types and locations of the gates are extracted from the DEF file. The SPEF file is used to calculate the lumped RC delays for the interconnect nets. For simplicity and better accuracy, the total lumped RC delay value is divided by the number of fanouts on the net so that the RC delays along single paths can be closer to their actual values. The paths were created by the timing analysis report (TARPT) which gives the top worst delay paths in our circuit. The gate information in the TARPT file is used to create our nodes in the paths and to connect the wires associated with them from the SPEF file to give us our edges. After path generation, the path delays and maximum path delay are calculated. Using a correlation matrix, the critical paths that are spatially correlated with each other are defined. Paths are defined to be spatially correlated if any of their gates are within 10 units of distance between each other. Distance was calculated using the place and route information in the DEF file.

A near critical path and a gate along this path are randomly chosen for each instance. Given a random amount of variation, the delay of all the gates before the chosen gate on the path are increased by some variable amount. The cell delay values are varied between 0-20% of the critical path delay to simulate the effects of variations. The input slews and interconnect delays are held constant to avoid re-synthesizing the circuit for each instance. After varying the cell delay, re-calculation of the path delays is done for the chosen path and all the paths that were spatially correlated to that path. The calculated path delays are compared with the maximum delay value for our timing yield. In the original case, the timing yield is defined by those path delays that have met the maximum timing delay constraint under variations with the implementation of the CSL module. For the 45nm case, the CSL module stretches the clock signal by 5% beyond the maximum delay. 100K random instances were performed and the timing yield was calculated which is shown in Table 5.3. From the analysis a 40% percent increase on average in timing yield with an average of 3.8% area overhead is shown. The variation in gate count, net count, and CSL overhead from the 65nm analysis is due to both the technology size as well as the cells used to synthesize the circuit. The benchmark b12 has the largest increase in timing yield but also the largest increase in area overhead which is due to the circuit area and the large number of near critical paths. The circuits which were impacted the most by the CSL insertion were b11, b15, and b17. These benchmarks incurred less than 5% area overhead and increased the timing yield of the circuits at least 50%. From the analysis, the benefit of CSL insertion can be realized and even increased as the scale of the circuits decrease.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this dissertation, multiple ways to enhance the reliability of VLSI circuits has been illustrated. In Chapter 3, several ways to detect and correct soft errors from the architectural level were investigated. Through the use of redundancy techniques like temporal, data value and information redundancy, soft error detection with negligible overheads in latency, area and power were given. Temporal redundancy was exploited statically using compiler directed slack computation and dynamically using the proposed slack predictor hardware. Small data value widths were mined to exploit data value redundancy. Information redundancy was supported by using an efficient implementation of residue codes. A cluster core architecture was proposed to reduce the latency overhead for inter-processor communication for our spatial redundancy technique where the redundant instruction is sent to the nearby core for execution. The results indicate that the combined framework can archive complete error coverage with significantly less overheads than other works existing in literatures. By incorporating the techniques for soft error detection with other low overhead methods designed to protect the effects of soft errors in sequential designs, a reliable fault tolerant design can be generated. To further this research, one could possibly experiment with a different simulator (such as the M5 simulator) to test the redundancy techniques as well as experiment with different architectures and benchmark suites.

In Chapter 4, a new clocking strategy for peak-power reduction was presented as well as a detailed set of algorithms to implement it at the gate level. A given circuit is considered in terms of its primary inputs and after the available slack values are determined through timing analysis, the slacks were used to sort the primary inputs into clusters. The clustered circuit paths can be clocked by phase-shifted clocks within the assigned clock periods where the phase shift is dependent on the slack values. The clustering algorithm determines the number of clusters and the allocation of paths into clusters in a fashion as to ensure that the clusters are balanced as much as possible in order to evenly distribute the load on the phase-shifted clocks. Our path clustering algorithm was computed to have a run-time analysis of O($\alpha(N + E)$. Experimental results were carried out on the ISCAS '85 benchmarks, along with test circuits from OpenCores and the LEON processor. The results were quite positive in terms of peak-power reduction. An average peak reduction around 25% was seen across the defined set cluster configurations. While this work is mainly focused on combinational circuits to show the proof of concept, further investigation is required to explore sequential circuits as well as complete processor architectures. The proposed method is extremely significant, since peak-power reduction is a critical challenge in VLSI circuits. High power density is recognized by the ITRS, Semiconductor Research Council (SRC) and by the VLSI research community as a key issue impeding advances in VLSI CMOS technology. This problem becomes more critical with the advent and proliferation of high-speed high-reliability processors and low-power computing devices happening today. The approach identified in this chapter was intended to reduce power density, peak-current demand and RMS current of high-speed devices. To extend this research, one could combine the proposed path clustering technique with other clock-skew scheduling techniques to determine whether further peak power minimization is possible. Another direction could be to utilize some of the industrial tools such as Synopsys ICC Compiler to perform IR noise analysis from the physical level of the design.

In Chapter 5, a dynamic clock stretching technique was presented to improve the timing yield of circuits in the presence of uncertainty due to process variations. Statistical optimization based techniques due to their conservative design, consume extra resources (performance and/or power) even in the absence of variations. The proposed methodology, on the other hand, adds a timing slack/margin (clock stretching) only in the presence of variations. Through identification of the critical transition interconnect, the CSL module was placed on near critical paths which stores the signal value before the falling edge of the clock arrives. After the falling edge of the clock, if a different value is seen in the CSL module, this signals the logic that delay due variation has occurred. Thus the CSL module stretches the clock signal to accommodate for this delay. The dynamic delay detection circuitry improves yield by controlling the instance of data captured in the critical path registers. Experimental results based on Monte-Carlo simulations ran on ITC ’99 benchmarks indicate a significant improvement in average timing yield with a negligible area overhead. Experimental results show that reducing the scale can also possibly increase the timing yield of the circuit with variations. To further this research, one could experiment using the PrimeTime-VX (Variation Extension) in order to extend the coverage of variation analysis. One would need to use a complete technology library which includes the FRAM view files as well as the technology files needed for placement and routing.

LIST OF REFERENCES

[1] Hossein Asadi, Vilas Sridharan, M.B. Tahoori, and D. Kaeli. Vulnerability analysis of l2 cache elements to single event upsets. In *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, volume 1, pages 1 –6, march 2006.

[2] Gordon B. Bell and Mikko H. Lipasti. Skewed redundancy. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 62–71, New York, NY, USA, 2008. ACM.

[3] L. Benini, G. De Micheli, A. Lioy, E. Macii, G. Odasso, and M. Poncino. Automatic synthesis of large telescopic units based on near-minimum timed supersetting. *Computers, IEEE Transactions on*, 48(8):769 –779, aug 1999.

[4] L. Benini, E. Macii, M. Poncino, and G. De Micheli. Telescopic units: a new paradigm for performance optimization of vlsi designs. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(3):220 –232, mar 1998.

[5] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *High-Performance Computer Architecture, 1999. Proceedings. Fifth International Symposium On*, pages 13 –22, jan 1999.

[6] T. Burd, A. Stratakos T. Pering, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *IEEE Solid-State Circuits Conference (ISSCC)* , pages 294–295, 2000.

[7] Jui-Ming Chang and Massoud Pedram. Energy minimization using multiple supply voltages. *IEEE Trans. Very Large Scale Integr. Syst.*, 5:436–443, December 1997.

[8] T. Chawla, S. Marchal, A. Amara, and A. Vladimirescu. Pulse width variation tolerant clock tree using unbalanced cells for low power design. In *Circuits and Systems, 2009. MWSCAS '09. 52nd IEEE International Midwest Symposium on*, pages 443 –446, aug. 2009.

[9] Po-Yuan Chen, Kuan-Hsien Ho, and TingTing Hwang. Skew aware polarity assignment in clock tree. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 376 –379, nov. 2007.

[10] S. Devadas, H. F. Jyu, K. Keutzer, and S. Malik. Statistical Timing Analysis of Combinational Circuits. In *International Conference on Computer Design*, pages 38–43, 1992.

[11] A. Devgan and C. Kashyap. Block-based Static Timing Analysis with Uncertainty. In *IEEE trans on CAD*, pages 607–614, 2003.

[12] A. Devgan and S. Nassif. Power Variability and its Impact on Design. In *Proc. of Intl. Conf. on VLSI Design*, pages 679–682, 2005.

[13] S. Dhar, D. Maksirnovi, and B. Kranzen. Closed-loop adaptive voltage scaling controller for standard-cell ASICs. In *IEEE symposium on Low Power Electronic Design*, pages 103–107, Aug 2002.

[14] S. Dolev and Y.A. Haviv. Self-stabilizing microprocessor: analyzing and overcoming soft errors. *Computers, IEEE Transactions on*, 55(4):385 –399, april 2006.

[15] M. Elgebaly and M. Sachdev. Variation-Aware Adaptive Voltage Scaling System. In *IEEE Trans. on VLSI* , pages 15(5) 560–571, May 2007.

[16] D. Ernst et al. A low power pipeline based on circuit level timing speculation. In *IEEE/ACM International Symposium on Microarchitecture*, pages 7–18, Dec 2003.

[17] S. Borkar et al. Parameter Variations and Impact on Circuit and Microarchitecture. In *DAC*, pages 338–342, 2003.

[18] Xin Fu, Tao Li, and J. Fortes. Combined circuit and microarchitecture techniques for effective soft error robustness in smt processors. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 137 –146, june 2008.

[19] Xin Fu, Wangyuan Zhang, Tao Li, and J. Fortes. Optimizing issue queue reliability to soft errors on simultaneous multithreaded architectures. In *Parallel Processing, 2008. ICPP '08. 37th International Conference on*, pages 190 –197, sept. 2008.

[20] M.A. Gomaa, C. Scarbrough, T.N. Vijaykumar, and I. Pomeranz. Transient-fault recovery for chip multiprocessors. *Micro, IEEE*, 23(6):76 – 83, nov.-dec. 2003.

[21] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walsta, and Changhong Dai. Impact of cmos process scaling and soi on the soft error rates of logic processes. In *VLSI Technology, 2001. Digest of Technical Papers. 2001 Symposium on*, pages 73 –74, 2001.

[22] B. P. Harish, Navakanta Bhat, and Mahesh B. Patil. On a generalized framework for modeling the effects of process variations on circuit delay performance using response surface methodology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(3):606 –614, march 2007.

[23] M. Hashimoto and H. Onodera. A Performance Optimization Method by Gate Sizing using Statistical Static Timing Analysis. In *International Symposium on Physical Design*, pages 111–116, 2000.

[24] J. Hu, S. Wang, and S.G. Ziavras. In-register duplication: Exploiting narrow-width value for improving register file reliability. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 281 –290, june 2006.

[25] J.S. Hu, F. Li, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin. Compiler-directed instruction duplication for soft error detection. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1056 – 1057 Vol. 2, march 2005.

[26] Shih-Hsu Huang, Chia-Ming Chang, and Yow-Tyng Nieh. Fast multi-domain clock skew scheduling for peak current reduction. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 254–259, Piscataway, NJ, USA, 2006. IEEE Press.

[27] C.J. Hughes, V.S. Pai, P. Ranganathan, and S.V. Adve. Rsim: simulating shared-memory multiprocessors with ilp processors. *Computer*, 35(2):40 –49, feb 2002.

[28] H. Iwai, K. Kakushima, and H. Wong. Challenges for future semiconductor manufacturing. *Int'l J.High-Speed Electronics and Systems*, 16:43–81, 2006.

[29] J. Jaehyuk Huh, C. Changkyu Kim, H. Shafi, L. Lixin Zhang, D. Burger, and S.W. Keckler. A nuca substrate for flexible cmp cache sharing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(8):1028 –1040, aug. 2007.

[30] Mark C. Johnson and Kaushik Roy. Datapath scheduling with multiple supply voltages and level converters. *ACM Trans. Des. Autom. Electron. Syst.*, 2:227–248, July 1997.

[31] Yooseong Kim, Sangwoo Han, and Juho Kim. Clock scheduling and cell library information utilization for power supply noise reduction. In *Semiconductor Technology and Science, 2009. JSTS 2009. Journal of*, volume 9, pages 29 –36, 2009.

[32] W.-C.D. Lam, Cheng-Kok Koh, and C.-W.A. Tsao. Power supply noise suppression via clock skew scheduling. In *Quality Electronic Design, 2002. Proceedings. International Symposium on*, pages 355 – 360, 2002.

[33] H.H.S. Lee, G.S. Tyson, and M.K. Farrens. Eager writeback-a technique for improving bandwidth utilization. In *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on*, pages 11 –21, 2000.

[34] Yann-Rue Lin, Cheng-Tsung Hwang, and Allen C.-H. Wu. Scheduling techniques for variable voltage low power designs. *ACM Trans. Des. Autom. Electron. Syst.*, 2:81–97, April 1997.

[35] V. Mahalingam and N. Ranganathan. Variation Aware Timing Based Placement Using Fuzzy Programming. In *Intl. Symposium on Quality Electronic Design*, pages 327–332, 2007.

[36] V. Mahalingam, N. Ranganathan, and J. E. Harlow. A Novel Approach for Variation Aware Power Minimization during Gate sizing. In *Intl. Symposium on Low Power Electronic Design*, pages 174–179, 2006.

[37] V. Mahalingam, N. Ranganathan, and R.Hyman Jr. A variation tolerant circuit design technique using dynamic clock stretching. *Emergining Technologies, ACM Journal of*, Sept 2011.

[38] Ali Manzak and Chaitali Chakrabarti. A low power scheduling scheme with resources operating at multiple voltages. *IEEE Trans. Very Large Scale Integr. Syst.*, 10:6–14, February 2002.

[39] R.S. Martin and J.P. Knight. Using spice and behavioral synthesis tools to optimize asics' peak power consumption. In *Circuits and Systems, 1995., Proceedings., Proceedings of the 38th Midwest Symposium on*, volume 2, pages 1209 –1212 vol.2, aug 1995.

[40] T.C. May and M.H. Woods. Alpha-particle-induced soft errors in dynamic memories. *Electron Devices, IEEE Transactions on*, 26(1):2 – 9, jan 1979.

[41] A. Meixner, M.E. Bauer, and D.J. Sorin. Argus: Low-cost, comprehensive error detection in simple cores. *Micro, IEEE*, 28(1):52 –59, jan.-feb. 2008.

[42] S. P. Mohanty, N. Ranganathan, and V. Krishna. Datapath scheduling using dynamic frequency clocking. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 65–, Washington, DC, USA, 2002. IEEE Computer Society.

[43] Saraju P. Mohanty, N. Ranganathan, and Sunil K. Chappidi. Simultaneous peak and average power minimization during datapath scheduling for dsp processors. In *ACM Great Lakes Symposium on VLSI*, pages 215–220, 2003.

[44] Saraju P. Mohanty, Nagarajan Ranganathan, Elias Kougianos, and Priyardarsan Patra. *Low-Power High-Level Synthesis for Nanoscale CMOS Circuits*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[45] S.P. Mohanty and N. Ranganathan. Energy efficient scheduling for datapath synthesis. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 446 – 451, jan. 2003.

[46] S.P. Mohanty and N. Ranganathan. A framework for energy and transient power reduction during behavioral synthesis. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 539 – 545, jan. 2003.

[47] S.P. Mohanty and N. Ranganathan. A framework for energy and transient power reduction during behavioral synthesis. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 12(6):562 –572, june 2004.

[48] S.P. Mohanty and N. Ranganathan. Simultaneous peak and average power minimization during datapath scheduling. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(6):1157 – 1165, june 2005.

[49] S.S. Mukherjee, M. Kontz, and S.K. Reinhardt. Detailed design and evaluation of redundant multi-threading alternatives. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 99 –110, 2002.

[50] S. R. Nassif. The impact of variability on power. In *Intl. Symposium on Low Power Electronic Design*, page 350, 2004.

[51] Fusun Ozguner, Duane Marhefka, Joanne DeGroat, Bruce Wile, Jennifer Stofer, and Lyle Hanrahan. Teaching future verification engineers: the forgotten side of logic design. In *Proceedings of the 38th annual Design Automation Conference*, DAC '01, pages 253–255, New York, NY, USA, 2001. ACM.

[52] Sanghun Park and Kiyoung Choi. Performance-driven high-level synthesis with bit-level chaining and clock selection. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(2):199 –212, feb 2001.

[53] Jesse Pool, Ian Sin Kwok Wong, and David Lie. Relaxed determinism: making redundant execution on multiprocessors practical. In *Proceedings of the 11th USENIX workshop on Hot topics in operating systems*, pages 5:1–5:6, Berkeley, CA, USA, 2007. USENIX Association.

[54] V. Raghunathan, S. Ravi, and G. Lakshminarayana. High-level synthesis with variable-latency components. In *VLSI Design, 2000. Thirteenth International Conference on*, pages 220 –227, 2000.

[55] V. Raghunathan, S. Ravi, A. Raghunathan, and G. Lakshminarayana. Transient power management through high level synthesis. In *Computer Aided Design, 2001. ICCAD 2001. IEEE/ACM International Conference on*, pages 545 –552, 2001.

[56] Salil Raje and Majid Sarrafzadeh. Variable voltage scheduling. In *Proceedings of the 1995 international symposium on Low power design*, ISLPED '95, pages 9–14, New York, NY, USA, 1995. ACM.

[57] Kaushik Ravindran, Andreas Kuehlmann, and Ellen Sentovich. Multi-domain clock skew scheduling. In *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, ICCAD '03, pages 801–, Washington, DC, USA, 2003. IEEE Computer Society.

[58] E.L. Rhod, C.A. Lisboa, and L. Carro. A low-ser efficient core processor architecture for future technologies. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1 –6, april 2007.

[59] E. Rotenberg. Ar-smt: a microarchitectural approach to fault tolerance in microprocessors. In *Fault-Tolerant Computing, 1999. Digest of Papers. Twenty-Ninth Annual International Symposium on*, pages 84 –91, 1999.

[60] S. Bhunia S. Ghosh and K. Roy. CRISTA: A New Paradigm for Low-Power, Variation-Tolerant, and Adaptive Circuit Synthesis Using Critical Path Isolation. In *IEEE Trans. on CAD*, pages 1947–1956, 26(11), 2007.

[61] R. Samanta, G. Venkataraman, N. Shah, and Jiang Hu. Elastic timing scheme for energy-efficient and robust performance. In *Quality Electronic Design, 2008. ISQED 2008. 9th International Symposium on*, pages 537 –542, march 2008.

[62] R. San Martin and J.P. Knight. Optimizing power in asic behavioral synthesis. *Design Test of Computers, IEEE*, 13(2):58 –70, summer 1996.

[63] T. Sato and T. Funaki. Power-performance trade-off of a dependable multicore processor. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pages 268 –273, dec. 2007.

[64] J. Semiao, J.J. Rodriguez-Andina, M. Santos F. Vargas, I. Teixeira, and P. Teixeira. Improving the Tolerance of Pipeline Based Circuits to Power Supply or Temperature Variations. In *Intl. Symposium on Defect and Fault-Tolerance in VLSI Systems*, pages 303–311, Sep 2007.

[65] J. Semiao, J.J Rodriguez-Andina, F. Vargas, M. Santos, I. Teixeira, and P. Teixeira. Process Tolerant Design Using Thermal and Power-Supply Tolerance in Pipeline Based Circuits. In *IEEE workshop on DDECS* , pages 1–4, Apr 2008.

[66] W. T. Shiue. Low power vlsi design: Peak power minimization using novel scheduling algorithim based on an ilp model. In *VLSI Design, 2002., Proceedings of the 10th NASA Symposium on*, Mar 2002.

[67] Wen-Tsong Shiue. High level synthesis for peak power minimization using ilp. In *Application-Specific Systems, Architectures, and Processors, 2000. Proceedings. IEEE International Conference on*, pages 103 –112, 2000.

[68] Wen-Tsong Shiue and C. Chakrabarti. Ilp-based scheme for low power scheduling and resource binding. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, volume 3, pages 279 –282 vol.3, 2000.

[69] Wen-Tsong Shiue, J. Denison, and A. Horak. A novel scheduler for low power real time systems. In *Circuits and Systems, 2000. Proceedings of the 43rd IEEE Midwest Symposium on*, volume 1, pages 312 –315 vol.1, 2000.

[70] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Dependable Systems and Networks, 2002. DSN 2002. Proceedings. International Conference on*, pages 389 – 398, 2002.

[71] D. Singh, J.M. Rabaey, M. Pedram, F. Catthoor, S. Rajgopal, N. Sehgal, and T.J. Mozdzen. Power conscious cad tools and methodologies: a perspective. *Proceedings of the IEEE*, 83(4):570 –594, apr 1995.

[72] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. Splash: Stanford parallel applications for shared-memory. *SIGARCH Comput. Archit. News*, 20:5–44, March 1992.

[73] Jared C. Smolens, Brian T. Gold, Babak Falsafi, and James C. Hoe. Reunion: Complexity-effective multicore redundancy. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 223 –234, dec. 2006.

[74] Karthik Sundaramoorthy, Zach Purser, and Eric Rotenburg. Slipstream processors: improving both performance and fault tolerance. *SIGOPS Oper. Syst. Rev.*, 34:257–268, November 2000.

[75] D. Sylvester and H. Kaul. Power-driven challenges in nanometer design. *Design Test of Computers, IEEE*, 18(6):12 –21, nov/dec 2001.

[76] T. Kuroda et al. Variable supply-voltage scheme for low-power high-speed CMOS digital design. In *IEEE Journal of Solid-State Circuits* , pages 33(3) 454–462, Mar 1993.

[77] Abhishek Tiwari, Smruti R. Sarangi, and Josep Torrellas. Recycle:: pipeline adaptation to tolerate process variation. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 323–334, New York, NY, USA, 2007. ACM.

[78] E. Touloupis, J.A. Flint, V.A. Chouliaras, and D.D. Ward. Study of the effects of seu-induced faults on a pipeline protected microprocessor. *Computers, IEEE Transactions on*, 56(12):1585 –1596, dec. 2007.

[79] D. Tran Vo and T. Bingel, October 2009. Systems and methods of integrated circuit clocking, Patent, US 8040155.

[80] http://www.eecs.umich.edu/~jhayes/iscas. Iscas '85 benchmarks.

[81] http://www.gaisler.com. Leon 2 processor.

[82] http://www.nangate.com. Nangate 45nm freepdk opencelllibrary.

[83] http://www.opencores.org. Opencores.

[84] T.N. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-fault recovery using simultaneous multithreading. In *Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on*, pages 87 –98, 2002.

[85] A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska. Clock skew optimization for ground bounce control. In *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on*, pages 395 –399, nov 1996.

[86] P. Vuillod, L. Benini, A. Bogliolo, and G. De Micheli. Clock-skew optimization for peak current reduction. In *Low Power Electronics and Design, 1996., International Symposium on*, pages 265 –270, aug 1996.

[87] N.J. Wang and S.J. Patel. Restore: Symptom-based soft error detection in microprocessors. *Dependable and Secure Computing, IEEE Transactions on*, 3(3):188 –201, july-sept. 2006.

[88] C. Weaver, J. Emer, S.S. Mukherjee, and S.K. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, pages 264 – 275, june 2004.

[89] Tsung-Yi Wu, Tzi-Wei Kao, Shi-Yi Huang, Tai-Lun Li, and How-Rern Lin. Combined use of rising and falling edge triggered clocks for peak current reduction in ip-based soc designs. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, ASPDAC ’10, pages 444–449, Piscataway, NJ, USA, 2010. IEEE Press.

ABOUT THE AUTHOR

Ransford Hyman Jr. received his Bachelor of Science degree in Mathematics in 2006 from Bethune-Cookman in Daytona Beach, Florida. He received his Master of Science in Computer Engineering Degree in 2010 from the University of South Florida in Tampa, FL. He is currently pursuing his Doctoral Degree in Computer and Science and Engineering at the University of South Florida in Tampa, Florida. He has accepted a Software Engineering position in the Design and Technology Solutions group at Intel in Folsom, California. His research interests are Reliability, Design Automation, Computer Architecture and multi-core processors. He is a National Science Foundation Bridge to Doctorate fellow as well as a Florida Education Foundation McKnight Doctoral fellow. He has mentored many students and has assisted with the recruitment of students in the College of Engineering. He was the 2009-2010 president of the IEEE-Computer Society USF Chapter and was awarded Best Student Organization at the 2009 USF Engineering Expo. He was awarded the Best Poster Award at the 2011 Richard Tapia Conference in San Francisco, California. He is a student member of the IEEE and the IEEE Computer Society.