PRITEXT: PROCESSOR RELIABILITY IMPROVEMENT THROUGH EXERCISE TECHNIQUE

A Thesis

by

MADHUKARREDDY PAPPIREDDY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Paul V. Gratz
Committee Members,	Jiang Hu
	Rabi Mahapatra
Head of Department,	Miroslav M. Begovic

August 2016

Major Subject: Computer Engineering

Copyright 2016 Madhukarreddy Pappireddy

ABSTRACT

With continuous improvements in CMOS technology, transistor sizes are shrinking aggressively every year. Unfortunately, such deep submicron process technologies are severely degraded by several wearout mechanisms which lead to prolonged operational stress and failure. Negative Bias Temperature Instability (NBTI) is a prominent failure mechanism which degrades the reliability of current semiconductor devices. Improving reliability of processors is necessary for ensuring long operational lifetime which obviates the necessity of mitigating the physical wearout mechanisms. NBTI severely degrades the performance of PMOS transistors in a circuit, when negatively biased, by increasing the threshold voltage leading to critical timing failures over operational lifetime. A lack of activity among the PMOS transistors for long duration leads to a steady increase in threshold voltage Vth. Interestingly, NBTI stress can be recovered by removing the negative bias using appropriate input vectors. Exercising the dormant critical components in the Processor has been proved to reduce the NBTI stress. We use a novel methodology to generate a minimal set of deterministic input vectors which we show to be effective in reducing the NBTI wearout in a superscalar processor core. We then propose and evaluate a new technique PRITEXT, which uses these input vectors in exercise mode to effectively reduce the NBTI stress and improve the operational lifetime of superscalar processors. PRITEXT, which uses Input Vector Control, leads to a 4.5x lifetime improvement of superscalar processor on average with a maximum lifetime improvement of 12.7x.

DEDICATION

To my Parents, Teachers and the Almighty

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Paul V. Gratz for his continuous guidance and support throughout my career as a graduate student at Texas A&M University. I would like to thank the members on my committee, Dr. Rabi Mahapatra and Dr. Jiang Hu for their valuable feedback on my work.

I am grateful to Stavros Hadjitheophanous and Dr. Maria K. Michael from University of Cyprus, and Dr. Vassos Soteriou from Cyprus University of Technology for their valuable inputs and continuous assistance in completing my research work.

I am grateful to the Department of Electrical and Computer Engineering at Texas A&M University for providing me financial support in the form of one time graduate merit scholarship, Teaching Assistant-ship and Student Grader. I am also thankful to my academic advisors Tammy Carda and Melissa Sheldon for all their help throughout my master's degree.

Finally and most importantly, I am very thankful to my family and friends for their unconditional support during difficult times and their encouragement in all my endeavors.

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

TABLE OF CONTENTS v

LIST OF FIGURES vii

1. INTRODUCTION 1

2. BACKGROUND 6

   2.1 NBTI degradation 6
   2.2 NBTI recovery 7
   2.3 Transistor delay degradation model 8
   2.4 Path delay 10

3. NBTI SENSITIVITY TO INPUT VECTOR 12

   3.1 Input vector control 12
   3.2 Intuition behind NBTI mitigation 13
   3.3 Analysis of skewed duty cycles in a combinational circuit 14

4. DETERMINISTIC VECTOR GENERATION ALGORITHM 18

   4.1 Basics of ATPG 18
   4.2 Exercise vector compaction 21
   4.3 Algorithm for exercise vector generation 23
   4.4 Use of existing DFT infrastructure 25
      4.4.1 Scan chains 25

5. SUPERSCALAR MICROARCHITECTURE 27

   5.1 FabScalar 27
   5.2 Core microarchitecture 28
      5.2.1 Fetch 28

5.2.2 Decode 29
5.2.3 Rename 30
5.2.4 Dispatch, issue 31
5.2.5 Execute 31
5.2.6 Complete, retire 32

6. PRITEXT: PROCESSOR RELIABILITY IMPROVEMENT THROUGH EXERCISE TECHNIQUE 33

6.1 Timing critical paths in superscalar core 33
6.2 Workload characterization of critical paths 34
6.3 Analysis of load store unit critical path 38
6.4 PRITEXT 42
6.5 Exercise mode during processor standby 44
6.6 PRITEXT enhanced ASIC design flow 45

7. EVALUATION 47

7.1 Experimental setup 47
7.2 Results 48
7.2.1 Vector generation results using deterministic algorithm 48
7.2.2 Balanced duty cycles using deterministic exercise vectors 51
7.2.3 Lifetime acceleration with RAS = 3:1 53
7.2.4 Lifetime acceleration with RAS = 1:1 53
7.2.5 Lifetime acceleration with unequal vector rotation periods 55
7.3 Power and area overheads of PRITEXT 56
7.3.1 Area overheard 56
7.3.2 Power analysis 56

8. RELATED WORK 58

9. CONCLUSION AND FUTURE WORK 63

REFERENCES 64

LIST OF FIGURES

FIGURE Page

1.1 Superscalar processor pipeline stages 2

2.1 PMOS transistor under NBTI stress when reverse biased 7

2.2 Increase in PMOS Vth during stress and recovery phases assuming 50% stress time 8

2.3 Increase in PMOS Vth during stress and recovery phases assuming 75% stress time 8

2.4 Delay increase in a multi-gate path between two latches 10

3.1 A synthetic combinational circuit with primary inputs $I[4:0]$, primary output Y and internal nets n_1, n_2, n_3 13

3.2 Block diagram of 4 input priority scheduler 15

3.3 Synthesized netlist of priority scheduler module with the critical timing path highlighted in red 16

4.1 (a) Activation and propagation cones for fault location f; (b) Stuck-at-0 fault location f; activated with B=1 and C=1, propagated by D=1 to output O2; effective test vectors ABCD of the form X111, (c) Assuming the critical net f to be stuck-at-0 fault, it can be activated by inputs B=1, C=1. 20

4.2 (a) On-Chip ROM with 3 vectors each with 10 bits; Size = 3 X 10, possible MUXs needed = 10, (b) After vector compaction: ROM Size = 3 X 4, MUXes needed = 4(unique values) + 4 (constants) = 8. . . 22

4.3 Algorithm for deterministic vector generation 23

5.1 FabScalar processor core with canonical template stages 28

5.2 Configuration of FabScalar processor core 29

5.3 Generic superscalar out-of-order pipeline stages 30

6.1 Duty cycle distribution of three critical path groups for bzip workload 35

6.2 Duty cycle distribution of three critical path groups for gap workload 36

6.3 Duty cycle distribution of three critical path groups for gzip workload 37

6.4 Duty cycle distribution of three critical path groups for mcf workload 38

6.5 Duty cycle distribution of three critical path groups for parser workload 39

6.6 Duty cycle distribution of three critical path groups for vortex workload 40

6.7 Increase in delay degradation for three critical path groups with time. The time at which the increase in delay exceeds the guardband (0.1T) is marked. 40

6.8 Critical path in load store unit: Memory violation check for load-store bypass 41

6.9 Proposed PRITEXT technique with On-Chip ROM , multiplexers and combinational logic cone of the critical path 43

6.10 Real-time statistics of processor utilization in a server 45

6.11 Automated ASIC design flow enabled with PRITEXT lifetime improvement technique 46

7.1 PRITEXT hardware with appropriate ROM vectors and multiplexers after vector compaction 48

7.2 Accessibility of internal nets with each input vector. 49

7.3 Coverage achieved by a set of input vectors. X-axis represents the number of vectors; Y-Axis represents the number of critical nets activated. Example: two of the nine vectors can activate 8 critical nets individually. 50

7.4 Duty cycle distribution of load store unit critical path of reference system. 51

7.5 Duty cycle distribution of load store unit critical path with RAS = 3:1 in a system enabled with PRITEXT. 52

7.6 Duty cycle distribution of load store unit critical path with RAS = 1:1 in a system enabled with PRITEXT. 53

7.7 Lifetime improvement with RAS = 3:1. 54

7.8 Lifetime improvement with RAS = 1:1. 54

7.9 Lifetime improvement with unequal rotation periods 55

1. INTRODUCTION

Moore's Law has been a primary contributor for performance improvement in microprocessors for several decades. The increased transistor count together with the smaller transistors helped to design processors with aggressive microarchitecture and higher clock frequency. The ever increasing demand for performance by emerging applications led to the design of Out-of-Order (OOO) Processors which effectively utilized the increased transistor count in the Chip.

However, as the transistor size of current CMOS technology approaches to nanoscale, transistor aging is a major concern in VLSI technology. Deep submicron process technologies are severely degraded by several physical wearout mechanisms which lead to prolonged operational stress and failure [6][20][26]. International Technology Roadmap for Semiconductors suggests that a ten-fold decrease in wearout of transistors is necessary to maintain the current design lifetimes without additional timing guardband [2]. In future CMOS process technologies, the large number of transistors are vulnerable to accelerated device degradation leading to poor reliability. Improving reliability of any semiconductor device is essential for achieving good lifespan in the deep submicron CMOS process technology.

The processor forms the crucial component of any computer system. With digitalization of crucial fields of human life such as Banking, Education, Health, Transportation, Entertainment, Manufacturing and Communications, modern applications demand superior performance and efficiency from processor. This inspired computer architects to design superscalar processors which leverage instruction level parallelism (ILP) in modern workloads [15]. In a modern Processor with superscalar execution capability, a large number of instructions are concurrently processed in the dynamic

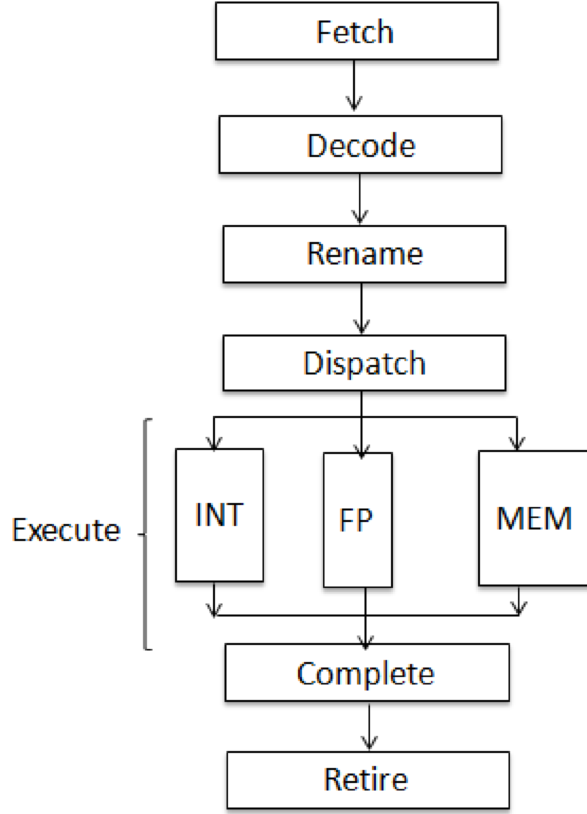

Figure 1.1: Superscalar processor pipeline stages

pipelines in an out-of-order fashion.

Figure 1.1 shows a block diagram of Superscalar processor. It consists of multiple pipeline stages including instruction fetch, decode, register rename, dispatch, issue, execute (Integer ALU, Floating Point ALU, Branch, Load Store Unit),complete, retire along with uncore components (not shown in figure). Even a single timing failure in any of the pipeline stages due to aging induced device degradation can render the processor and the entire system useless. Hence reliability is a major concern for semiconductor industry and has become a first-order constraint in processor design besides Power, Performance and Area.

Negative Bias Temperature Instability (NBTI) is the prominent failure mechanisms which degrades the reliability of current and future semiconductor devices. It stresses the transistor and increases threshold voltage Vth leading to switching delay and critical path degradation [3]. A transistor is said to be aged when its operational threshold voltage Vth is higher than its initial value. NBTI induced device aging is proportional to the device stress time (fraction of time PMOS is reverse biased i.e. its gate is at logic 0) respectively [16]. It does not result in circuit opens or shorts but leads to critical path timing violations over operational life-time. Chip designers have accounted for this aging induced circuit timing delays by adding a guard band (about 10%) to the system clock-period. The useful (operational) lifetime of a processor chip is limited to the time by which the transistors along the critical path wearout to cause the cumulative increase in the switching delay to exceed the available timing guard band i.e. the first timing failure in the circuit determines the lifetime of the processor.

The continuous increase in transistor density together with the aggressive microarchitecture of modern processors leads to unbalanced utilization of processor critical paths. This has resulted in higher device stress making NBTI degradation a threat to future processor designs [26][13][7][23]. As per IBM Reports, NBTI is the most severe degradation mechanism in current CMOS process technologies below 90-nm. The impact of NBTI stress on degradation of circuit delay is about 15 percent in 65-nm process technology [20].

In older process technologies, designers have dealt with the aging induced device degradation through additional timing guardband and ensured a reliable system performance over the expected lifetime of the Processor. However, in current and future CMOS process technologies, such guardband will be ineffective and new techniques at both microarchitectural and circuit level are necessary to mitigate the perfor-

mance degradation due to the accelerated physical device wearout mechanisms. Our work develops novel microarchitectural technique for NBTI mitigation and lifetime improvement of superscalar processors.

Several researchers have proposed various circuit level and microarchitectural techniques to manage device degradation due to NBTI induced transistor aging[27][13][7][16][14][25][17][5][12]. Several of these approaches use a well known technique called Input Vector Control (IVC)[27][13][7][12][29]. The idea is that since the NBTI stress depends on the logic level at the gate of the internal PMOS transistors, input vectors can be used to reduce the amount of time the PMOS transistors observe logic 0 at their gate inputs. Hence an input vector can be used to change the state of the combinational logic and reduce the NBTI induced stress.

Kim et al. have proposed one such technique to derive a set of input vectors to decelerate the aging induced by NBTI stress in a Network-on-chip (NoC) router of a Chip Multiprocessor (CMP)[16]. They developed critical path based timing model and characterized the timing delay degradation due to NBTI wearout across timing sensitive paths in NoC router. The study confirms the fact that NBTI induced aging is a major wearout mechanism and is accelerated because of the under utilization of various paths leading to biased duty cycles (fraction of time at logic 0) across the timing critical paths. We extend the work done by Kim et al.[16] to improve the lifetime of Superscalar processors and propose a novel microarchitectural technique, PRITEXT, which significantly prolongs the lifetime of processor by exercising the internal nodes along the timing critical paths using input vectors to achieve a balanced utilization (duty cycle). Our results are promising and PRITEXT achieves an average lifetime improvement of 4.5x compared to a reference processor with no NBTI mitigation.

The rest of the thesis is organized as follows. In chapter 2, we provide a brief

description of transistor aging and critical path delay degradation by NBTI stress as well as the interesting recovery phenomenon exhibited by NBTI degradation. We also explain the mathematical modeling of operational lifetime improvement factor originally proposed by Kim et al [16]. Chapter 3 provides a description of Input Vector Control technique and the intuition behind using input vectors to alleviate the NBTI stress. We also give a working example of NBTI acceleration factors in Priority Scheduler, a synthetic combinational logic which implements an arbitration module in several complex designs. In chapter 4, we introduce the algorithm developed by Kim et al. to generate a deterministic set of input vectors to achieve balanced utilization across PMOS transistors on the timing critical paths of the processor. Chapter 5 presents a brief description of superscalar processor microarchitecture. In Chapter 6, we present the workload characterization of all the timing critical paths which are sensitive to NBTI stress and describe our proposed lifetime extending micro-architectural technique, PRITEXT, which uses the exercise vectors generated from the novel algorithm described in Chapter 4. Chapter 7 evaluates the efficiency of our proposed methodology in mitigating the NBTI stress and provides the lifetime improvement results achieved. Chapter 8 discusses the prior work done by the research community. Finally Chapter 9 concludes our work and provides future work suggestions.

2. BACKGROUND*

In this chapter, we present the transistor aging model under NBTI stress followed by the modeling of degradation of critical path delay in a combinational circuit. We also describe the NBTI stress recovery phenomenon exhibited by the PMOS transistors when forward biased.

2.1 NBTI degradation

Negative Bias Temperature Instability, being one of the major wear out mechanisms in deep submicron process technologies, is responsible for gradual degradation of circuit performance over lifetime. Notably, NBTI affects PMOS transistor of a gate when reverse biased but does not significantly affect NMOS transistor. As shown in Figure 2.1, when the gate of the PMOS transistor is pulled to logic 0 by a corresponding internal net, the transistor is reverse biased (Vgs = -Vdd) and is continuously stressed[6][20][26]. This leads to generation of interface traps due to disassociation of Si-H bonds in Si/Sio2 interface which leads to an increase in the threshold of the transistor and a simultaneous reduction in the drive current due to charge carrier mobility degradation. A continuous increase in threshold voltage results in accelerated transistor aging leading to steady decrease in switching speeds. In long term, NBTI stress of PMOS transistor leads to circuit timing failures which undermine the operational lifetime of the device.

---

*Part of this chapter is reprinted with permission from Kim, H. "Use It or Lose It: Proactive, Deterministic Longevity in Future Chip Multiprocessors," Proceedings of ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 20, Issue 4, ©2015 ACM, Inc. http://doi.acm.org/10.1145/2770873

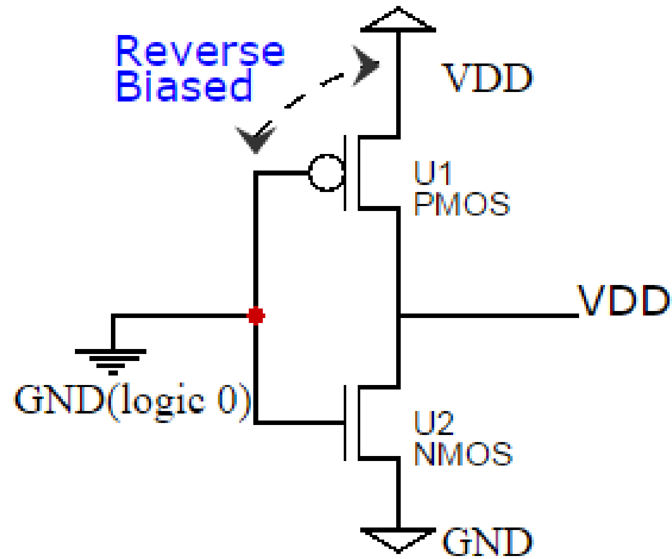

Figure 2.1: PMOS transistor under NBTI stress when reverse biased

2.2 NBTI recovery

NBTI has an interesting recovery phenomenon. Whenever the stress is removed i.e., the gate is not reverse biased(Vgs =0), most of the Hydrogen atoms diffuse back and bond with Silicon leading to recovery of the threshold voltage[6][20][26]. However, the recovery in the threshold voltage is only partial as shown in Figure 2.2. The increase in operating threshold voltage due to NBTI stress is sensitive to the fraction of the time the transistor is under negative bias(stress). Figure 2.2 illustrates the net increase in Vth assuming the PMOS transistor to be in stress for 50% of the time while figure 2.3 illustrates the net increase in Vth assuming the transistor is in stress mode for 75% of the time. The rate of increase in Vth is much higher when the fraction of stress mode is greater than recovery mode. This recovery phenomenon is exploited by several researchers to minimize the degradation of performance due to NBTI stress on PMOS transistors.

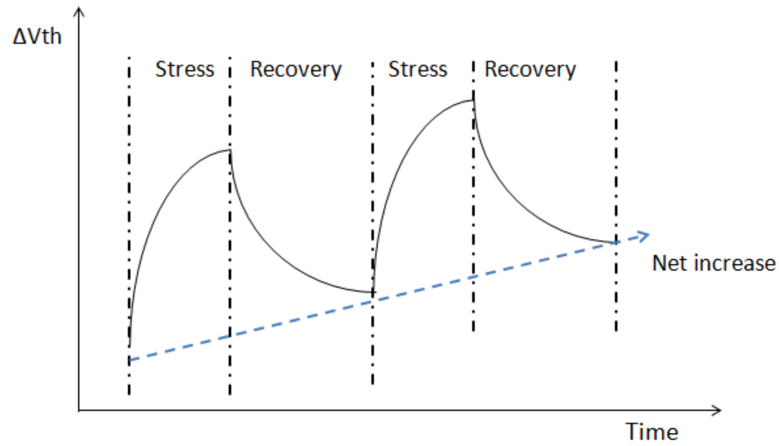

Figure 2.2: Increase in PMOS Vth during stress and recovery phases assuming 50% stress time

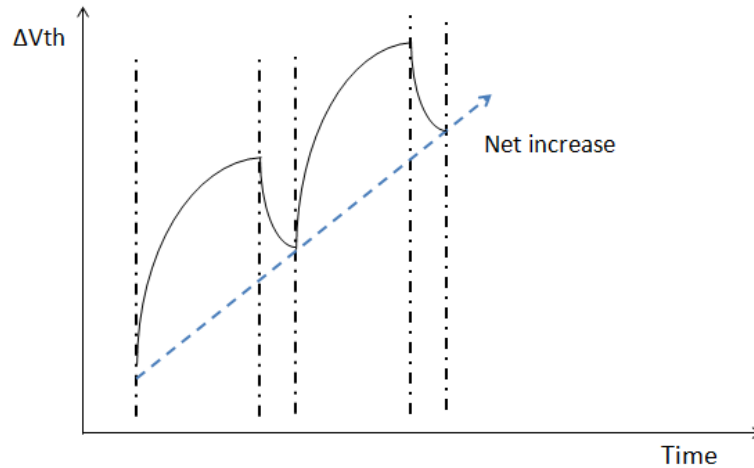

Figure 2.3: Increase in PMOS Vth during stress and recovery phases assuming 75% stress time

2.3 Transistor delay degradation model

As stated earlier, NBTI does not induce hard failures. It shifts the circuit parameters (threshold voltage, switching delays) over time under operational stress. We use Reaction-Diffusion (R-D) model that correlates Vth shift as an approximation

for NBTI stress [28]. Since the PMOS transistor is under NBTI stress only when it is reverse biased, the amount of stress is dependent on the fraction of time for which the gate terminal of the transistor is held at a low voltage level (logic 0) . Based on the AC stress model for NBTI degradation proposed by Lu et al. [19], the increase in threshold voltage Vth for PMOS transistor is estimated by the equation

$$\Delta V_{th_NBTI} = A\left(\frac{\beta}{1-\beta}\right)^{n} t^{n} e^{\left(-\frac{nE_{aNBTI}}{kT}\right)}, \tag{2.1}$$

where T is the temperature, E_{aNBTI} is the activation energy, t is the operating time, k is Boltzmann's constant, n is the time exponent(=1/6), and A is a fitting constant[19].

Alpha Power law[22] states that an increase in Vth leads to increase in transistor switching delay as shown here

$$d_g \propto \frac{V_{dd}}{\mu(V_{dd} - V_{th})^{\alpha}} \tag{2.2}$$

where d_g is the transition delay, $\mu \propto T^{-1.5}$ (T being Temperature) and $\alpha = 1.3$.

Based on the above two equations , the increase in transistor switching delay can be estimated as

$$\Delta d_{g_NBTI} = \hat{A}\left(\frac{\beta}{1-\beta}\right)^{n} t^{n} e^{\left(-\frac{nE_{aNBTI}}{kT}\right)}. \tag{2.3}$$

Lifetime of a device can be defined as the time until which an integral component degrades beyond the point which the device is no longer guaranteed to function for an intended application. Since designers consider a 10% guardband, an increase of 10% in transistor Vth is considered to be the end of lifetime since the transistor is considered to be over-aged [28]. Therefore the lifetime of a single transistor which is

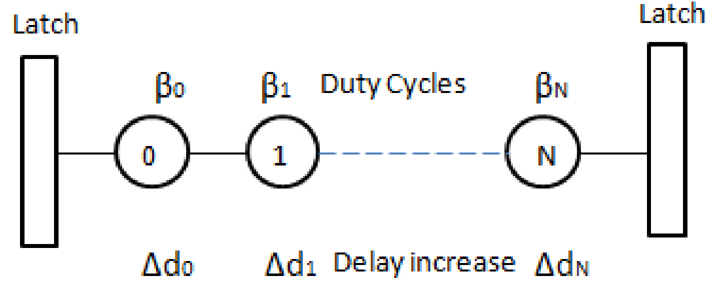

Figure 2.4: Delay increase in a multi-gate path between two latches

alternatively under stress and recovery phases can be estimated from above equations as

$$TTF_{NBTI} = \left[A_{NBTI} \left(\frac{1-\beta}{\beta} \right)^n e^{\left(\frac{nE_{aNBTI}}{kT} \right)} \right]^{1/n}. \tag{2.4}$$

2.4 Path delay

Section 2.3 gives a detailed analysis of a delay degradation of a transistor in a single gate. Any complex digital circuit has sequential and combinational logic. Such combinational logic has several timing paths between primary input and primary output with multiple gates in each path. In case of a multi-gate path having several transistors along the timing path, the cumulative transistor delay shift must be considered as the aggregate increase in timing delay rather than the worst-case delay degradation of a single gate [16]. Therefore , the lifetime of a device with combinational and sequential logic is limited by the operational time when the cumulative increase in the timing delay across any of the multi-gate paths exceed the timing guardband(typically 10% of clock period).

Figure 2.4 shows the multi-gate path between two latches. The cumulative delay increase in the path is dependent on the individual delays of every transistor along

the path. The increase in switching delay due to the PMOS transistor of the ith gate can be expressed as

$$\Delta d_i(\beta_i, t) = \psi \times t^n \times \left(\frac{\beta_i}{1-\beta_i}\right)^n \tag{2.5}$$

where β_i is the duty cycle fof the ith gate and ψ is a proportionality constant shown in equation earlier.

The cumulative increase in delay along a path with N gates at time t can be estimated as a sum of individual gate delay increase as shown below

$$\Delta d(t) = \sum_{i=0}^{N-1} \Delta d_i(\beta_i, t) = \psi \times t^n \times \sum_{i=0}^{N-1} \left(\frac{\beta_i}{1-\beta_i}\right)^n \tag{2.6}$$

A device is said to be functionally reliable as long as the increase in the switching delay of the critical timing path does not exceed the timing guard band. Hence, Lifetime, $T_{lifetime}$, is defined such that $\Delta d(T_{lifetime})$ is less than the guard band. The lifetime improvement is quantified by acceleration factor defined as follows:

$$AF(x) = \frac{T_{lifetime}(x)}{T_{lifetime}(ref)} = \left(\frac{\sum_{j=0}^{M-1} \left(\frac{\beta_j}{1-\beta_j}\right)^n}{\sum_{i=0}^{N-1} \left(\frac{\beta_i}{1-\beta_i}\right)^n}\right)^{1/n} \tag{2.7}$$

where $T_{lifetime}(x)$ is the operational lifetime of device enhanced with NBTI mitigation techniques using PRITEXT , $T_{lifetime}(ref)$ is the operational lifetime of the reference device with no reliability mitigation[16]. Also β_i and β_j represent the duty cycle of the ith gate on the critical path of the enhanced device and of the jth gate on the critical path of the reference device assuming the numbers of gates on the equivalent critical path are N and M respectively.

3. NBTI SENSITIVITY TO INPUT VECTOR

In this chapter, we present a brief description of Input Vector Control(IVC) technique and the intuition behind using input vectors to alleviate the NBTI stress. We also give a working example of NBTI acceleration factors in Priority Scheduler, a synthetic combinational logic which implements an arbitration module in several complex designs.

3.1 Input vector control

The state of any combinational circuit depends on the input vectors. Any internal net, which drives the gate of PMOS transistor, can be switched to logic 1 using an appropriate input vector as shown in figure 3.1. This technique of controlling the state of the internal combinational logic is termed Input Vector Control (IVC)[27][13][7][12][29]. It is a well known technique in minimizing the leakage current in CMOS devices [27][4][29].

NBTI strongly depends on the state of the internal nets. An internal net at logic 0 leads to NBTI stress on PMOS transistor as shown in figure 2.1. The NBTI degradation experienced by a transistor is directly dependent on the duration of stress i.e. the amount of time the transistor is reverse biased. IVC can be used to drive the internal nets to logic 1 thereby removing the negative bias on the PMOS transistor during standby mode when the device is not in active use. Hence by balancing the ratio of stress time to the total operational time i.e. fraction of the time the corresponding internal net is at logic 0, NBTI degradation can be greatly alleviated. This idea of controlling the state of the internal net using IVC and balancing its duty cycle is the basis of our approach in building the PRITEXT microarchitecture.

Note that a single input vector cannot activate (i.e. pull up) all the internal nets

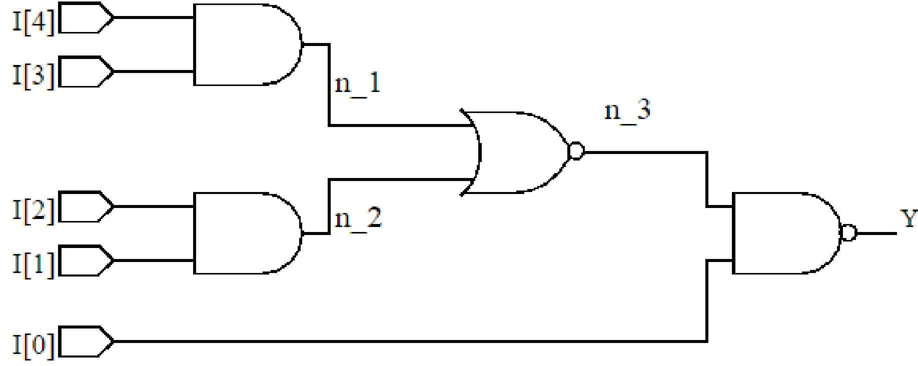

Figure 3.1: A synthetic combinational circuit with primary inputs $I[4:0]$, primary output Y and internal nets n_1, n_2, n_3

on any single timing path. Consider the path $I[4]->n_1->n_3->Y$ in the synthetic circuit shown in figure 3.1 where I is the Primary input, n_1, n_3 are the internal nets and Y is the primary output. Input vector I = 11111 switches net n_1 to logic 1 while simultaneously it switches net n_3 to logic 0. Hence a set of input vectors are required to activate all the internal nets and balance the duty cycles. Our aim is to achieve a minimum set of deterministic input vectors to achieve balanced duty cycles across the internal nets. Switching internal nets to logic 1 during standby time using IVC is often known as exercise mode[16]. Since the input vectors are used to exercise (i.e. switch) the internal nets, the set of input vectors are termed as exercise vectors.

3.2 Intuition behind NBTI mitigation

As explained in earlier section, NBTI is highly sensitive to duty cycle due to the $\frac{1}{(1-\beta)}$ factor which is evident from the mathematical expression in equation 2.3. Using IVC, the highly skewed duty cycles (closer to 1.0) of internal nets can be balanced (bring down to 0.0) in exercise mode. In order to demonstrate the sensitivity of NBTI to duty cycle, let us assume an internal net having a duty cycle of 0.99. The

delay degradation is proportional to

$$\left(\frac{0.99}{1-0.99}\right)^n = (99)^n \tag{3.1}$$

where n is the time exponent and is equal to 1/6. Considering a worst case scenario in which the device is in standby mode for 10% of the operational time, the average duty cycle can be brought down by 10% to 0.9 by using exercise vectors and driving the net to logic 1. The delay degradation is now proportional to 9^n which is 11^n times better than a reference device with no exercise mode. The lifetime improvement (AF) can be computed from equation 2.7 as:

$$\left(\frac{99^n}{9^n}\right)^{1/n} = 11 \tag{3.2}$$

3.3 Analysis of skewed duty cycles in a combinational circuit

As evident from the equation 2.6, NBTI sensitivity of a multi-gate timing path is highly dependent on the total switching delay of the path and the duty cycle (probability of signal to be at logic 0) of the transistors on the path [17][11]. The total switching delay of the path depends on the microarchitecture of the design and increases exponentially with the design complexity. The duty cycle of the gates depend on the utilization of the critical paths in the circuit and the design choices of the various components of the processor [14][5]. Some of the observations based on the common design choices of processors are listed below:

- Most data-path components in processors exhibit a substantial non-uniformity in their utilization for real world workloads. This contributes to skewed duty cycle of the gates along the under-utilized critical paths of the circuit [5].
- Most designs of various components of a processor are highly biased to logic 0.

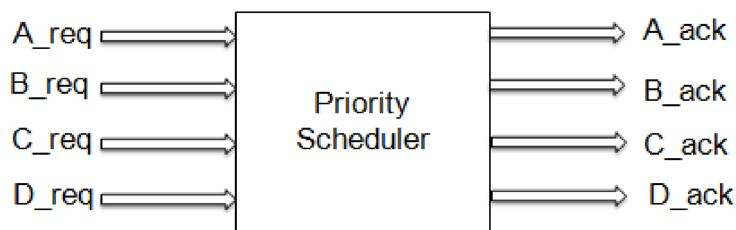

Figure 3.2: Block diagram of 4 input priority scheduler

PMOS transistors of Adder module observe logic 0 at their gate for significant fraction of time. The probability of logic 0 for various patterns at the inputs of register files, data caches, schedulers and various other components of a superscalar processor ranges between 65% to 90% [14]. This leads to a accelerated aging of PMOS transistors due to NBTI Stress.

- Modern superscalar processors have aggressive microarchitecture and implement complex state machines which handle several scenarios. Such designs tend to have several corner cases which get synthesized to timing paths having several gates along the path. Paths having many gates have higher cumulative switching delays and hence a higher probability to violate the timing guardband leading to timing failures. Moreover paths with larger switching delays experience accelerated NBTI stress which further exacerbates the problem[16].

We demonstrate the existence of above three observations in Priority Scheduler, a common design component in a sequential circuit which implements a priority among the various scenarios (quantified by input signals being logic 1). Figure 3.2 and figure 3.3 illustrate the block diagram and the gate level netlist of a priority scheduler module with 4 inputs which establishes priority between the inputs and functions as an arbiter with higher precedence for certain inputs.

The Priority Scheduler enables only one output signal at any time. Assuming

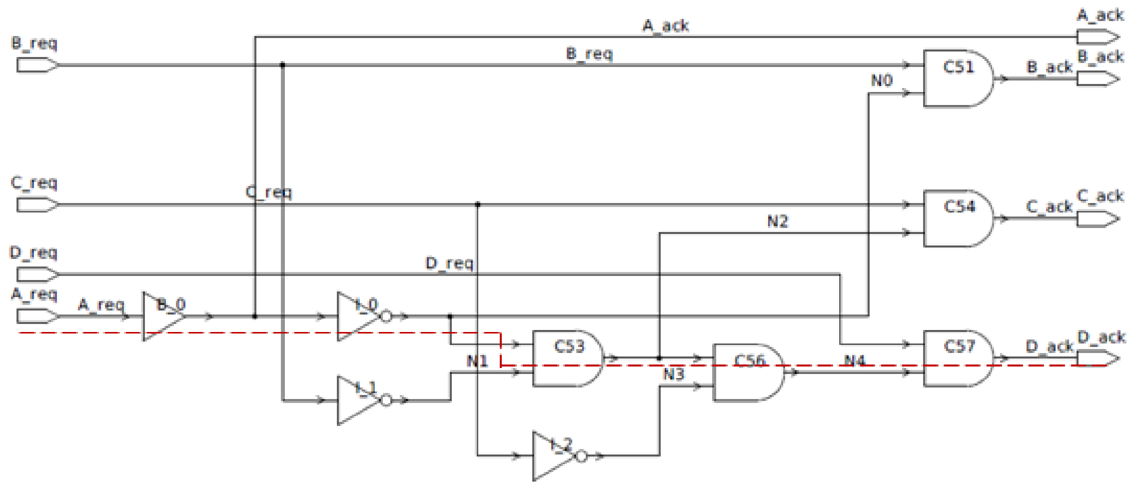

Figure 3.3: Synthesized netlist of priority scheduler module with the critical timing path highlighted in red

random service requests from the four inputs, the probability of `D_ack` to be high is small since the corresponding input `D_req` will be served only when the other inputs `A_req`, `B_req` and `C_req` are not requesting for service. This follows the observation 1. Most of the internal nets are also biased to logic 0 confirming observation 2. Also it is visible that `D_req` lies in the critical path and has the longest switching delay as seen in the figure 3.3 with a red dashed line which confirms the observation 3. This is evident from the Boolean equations for `D_ack` as listed here

$$\texttt{A_ack} = \texttt{A_req} \tag{3.3}$$

$$\texttt{B_ack} = \overline{\texttt{A_req}}.\texttt{B_req} \tag{3.4}$$

$$\texttt{C_ack} = \overline{\texttt{A_req}}.\overline{\texttt{B_req}}.\texttt{C_req} \tag{3.5}$$

$$\texttt{D_ack} = \overline{\texttt{A_req}}.\overline{\texttt{B_req}}.\overline{\texttt{C_req}}.\texttt{D_req} \tag{3.6}$$

4. DETERMINISTIC VECTOR GENERATION ALGORITHM

The efficiency of Input Vector Control in exercising the internal nets on the timing critical path to balance the corresponding duty cycles determines the efficiency of our NBTI mitigation approach in reducing the transistor degradation and improving the processor lifetime. As mentioned earlier in chapter 3, exercise vectors are injected into the combinational circuit during the exercise (standby) mode when the processor is in idle state. In an ideal scenario, a single input vector should be able to exercise (switch to logic 1) all the internal nets on the timing critical path. However as stated in section 3.1, multiple vectors are needed to activate all the nets. An individual vector exercises more than one internal net on a timing path, however the rest of the nets are deactivated (switched to logic 0) simultaneously. In order to ensure the nets are activated for majority of the time during standby mode, the number of input exercise vectors should be kept as low as possible.

The task of finding input vectors to activate/exercise internal nets resembles the Automatic Test Pattern Generation (ATPG) process, a well-known NP-complete problem used in developing test vectors used for validation of integrated circuits [8][9]. The process of ATPG is intended to generate a set of test vectors which are used in testing the integrated circuit for possible manufacturing defects such as stuck-at faults. Each test case intends to check if a particular internal net in a circuit comprising of logical gates such as NAND, NOR, NOT is stuck at either logic 0 or at logic 1 due to possible shorts occurring during the chip fabrication [9].

4.1 Basics of ATPG

The process of generating an ATPG test vector for catching a stuck-at-fault comprises of two phases: fault activation phase and fault propagation phase. Figure

4.1(a) depicts the idea behind activation and propagation phases. During the fault activation phase, the corresponding internal net f (fault location) is forced to the opposite logic level of the fault value i.e. if the net is stuck-at-0, it is forced to logic 1 during the activation phase and vice versa. The primary inputs and all the gates which lie in the fan-in of the internal net are called as the activation cone. Note that only few of the primary inputs are sufficient to activate an internal net. For example, as visible in the figure 4.1, input pins I[2], I[1], I[0] do not lie in the fan-in of the internal net n_1. In order to confirm a valid stuck-at fault, the signal from the activation phase must be propagated to any primary output signal where it is observable such that a comparison can be made with the anticipated output signal. The combinational logic which lies between the internal net and the output signals and forms all the possible paths for propagating the fault to observable output is referred to as propagation cone. Activation cone and propagation cone for the fault location f in the gate-level Netlist of device under consideration are illustrated in Figure 4.1(a).

In the context of ATPG vectors, the procedure of enabling each of the activation and propagation phase using appropriate input vectors is referred to as Justification procedure. Fault activating Justification procedure determines the input signals (test vector) such that the stuck-at-fault is activated. Justification during fault propagation phase determines the values of the corresponding input signals to make sure the fault is propagated to a particular output signal through a certain propagation path. One working example of justification procedure for activation and propagation of fault f is illustrated in Figure 4.1 (b) and (c). As a part of justification during activation phase, input signals B and C are set to 1 and 1 respectively in order to activate (switch to logic 1) the stuck-at-0 fault location f. Also, as part of justification for propagation phase, input signal D is set to 1 in order to propagate the fault to the

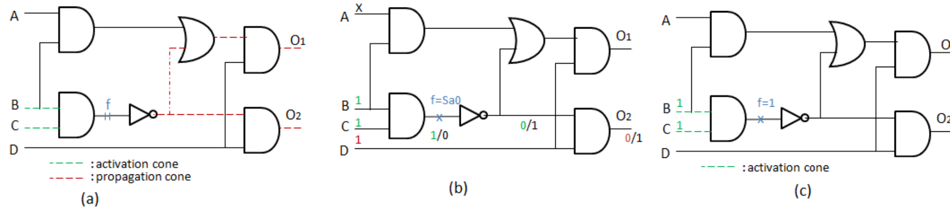

Figure 4.1: (a) Activation and propagation cones for fault location f; (b) Stuck-at-0 fault location f; activated with B=1 and C=1, propagated by D=1 to output O2; effective test vectors ABCD of the form X111, (c) Assuming the critical net f to be stuck-at-0 fault, it can be activated by inputs B=1, C=1.

output signal O2. From the figure 4.1(a), it is evident that input signal A does not play a role in either activation or propagation of fault. Hence signal A does not have a deterministic value and is considered to be a don't care value (X) which means it can be set to either logic 0 or logic 1. In case of stuck-at-0 fault at location f, the value at the output O2 is '1', otherwise it is '0' as illustrated in figure 4.1 (b). This is denoted by the symbol vff/vf where vff stands for fault-free-value and vf stands for fault-value which is 0/1 in this illustration.

Justification procedure during both activation and propagation phase is a NP-Complete problem and the time complexity of such non-polynomial problem is exponential to the number of input signals in the worst case. However, our aim in mitigating NBTI stress by balancing the duty cycles only incorporates the activation phase of the ATPG problem. We only need to activate (i.e. exercise) the internal nets using appropriate input signals (exercise vector) since there is no need to propagate the internal value of the signal to any output signal. Hence, it is sufficient to justify the activation phase of ATPG in order to exercise the internal nets. For example, it sufficient to set B=1 and C=1 in Figure 4.1 (c) in order to exercise signal(i.e. internal net) f . The possible input signals (vector) which can achieve this are of the form ABCD= X11X where X stands for dont cares.

4.2 Exercise vector compaction

Exercise mode vectors need to be stored in an On-Chip ROM in order to be accessed during standby mode for NBTI mitigation. The size of the On-chip ROM adds to the hardware complexity of our technique implementation. Moreover, several multiplexers are needed to select between the exercise vectors in standby mode and the operational vectors during active mode. The operational vectors are the input signals from either a primary input or from a pipeline stage of the processor. The size of the ROM and the number of MUXes depend on the dimensions of the exercised vector. In order to reduce the hardware complexity due to additional ROM and MUXes, it is necessary to reduce the size of individual vector and the number of exercise vectors. Consider the figure 4.2 which illustrates one example of a On-chip ROM matrix having 3 vectors each of which are 10 bits wide. The number of exercise vectors determines the depth of the matrix whereas the width of each exercise vector determines the width of the matrix. The number of MUXes needed to select between normal and exercise vectors is equal to the width of the vector (number of input signals which lie in the activation cone of the internal net) which is 10 in this example. Hence the vector generation algorithm should try to achieve a set of minimum number of vectors, each of which can activate a larger number of critical nets. This will help to activate each internal net for major fraction of the time as well as helps to reduce the hardware complexity by lowering the number of MUXes. This is also referred to as the test vector compaction problem in the ATPG domain [9].

Consider the table in figure 4.2 (a). It represents an On-Chip ROM with 3 vectors each of which are 10 bits wide. The exercise mode will need 10 MUXes for each of the input signal bit in a simple implementation. However, it can be noticed that several

Possible MUX locations										
Vector	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
V1	X	1	X	X	1	1	1	X	0	X
V2	0	1	X	1	X	1	1	X	1	X
V3	1	1	X	X	0	0	1	0	1	X

Optimized ROM				
Vector	I1	I5	I6	I9
V1	X	1	1	0
V2	0	X	1	1
V3	1	0	0	1

Figure 4.2: (a) On-Chip ROM with 3 vectors each with 10 bits; Size = 3 X 10, possible MUXs needed = 10, (b) After vector compaction: ROM Size = 3 X 4, MUXes needed = 4(unique values) + 4 (constants) = 8.

bits of the exercise vectors are dont cares and hence ideally there is no need of MUX to select these signals(I3, I10 in our example). Also consider the columns I2, I7 in which all the vectors have same value. There is no need to store a constant bit of all the vectors in the table and can be removed from the ROM but a MUX is still needed which has to be set to a constant value. An even more interesting observation can be made in this vector table for the columns I4, I8. These columns have only one valid logic value and the rest are dont cares. These columns can be removed from the table and a MUX can be used with its corresponding value set to a constant value for each of the columns. The rest of columns, I1, I5, I6 and I9 represent vectors with unique value for each of the bits. Hence the optimized ROM will can be compressed to a size of 3 x 4(three vectors each 4 bits wide) as illustrated in figure 4.2(b). The number of MUXes needed will be equal to the number of columns having at least one constant value, which is 8 in our example: Columns I1, I2, I3, I4, I5, I7, I8 and I10. Figure 4.2 (b) illustrates the final size of ROM and MUXes incorporated to achieve exercise mode using compressed vectors.

Based on the above example, it is evident that the vector generation algorithm should strive for generating a minimum number of exercise vectors with as many dont care bits as possible in each vector. One such algorithm has been proposed by Kim et al and is described here.

Procedure Exercise Vector Generation ( )
Inputs: Baseline Processor core netlist $\boldsymbol{P}$, critical nets list $\boldsymbol{N}$, duty cycle per critical net $\boldsymbol{D}$
Outputs: Set of exercise vectors $\boldsymbol{V}$, list of exercised critical nets $\boldsymbol{N}_e$

```
01: Sort the elements of the critical nets list N based on D
02: Ne = NULL;                      // list of exercised critical nets
03: Nred = NULL;                    // list of redundant critical nets
04: j=1;                            // exercise vector index
05: while (N ≠ ∅)
06:     vj = X;                     // initialize vj with all unassigned values (don't cares)
07:     ∀ critical net ni ∈ N       // for each net not exercised yet
08:         vj'= justify (R, ni, vj);   // justify additional values of vj in order to exercise ni
09:         if (vj' != NULL )
10:             add ni in Ne and delete ni from N
11:             simulate vj' on R
12:             ∀ nk ∈ N            // for each net not exercised yet
13:                 if (nk == 1)
14:                     add nk in Ne and delete nk from N
15:             vj = vj';           // update current vector
16:         else
17:             add ni to Nred and delete ni from N
18:     add vj in V
19:     j++;
20: return V, Ne;
```

Figure 4.3: Algorithm for deterministic vector generation .

4.3 Algorithm for exercise vector generation

Figure 4.3 illustrates the deterministic vector generation algorithm proposed by Kim et al.[16]. The algorithm takes two inputs: (1) the combinational logic cone of the critical timing path which is available from the synthesized gate-level Netlist of superscalar processor P, (2) list of the internal nets N on the critical timing path with corresponding average duty cycles (fraction of the net is at logic 0) D. The

average duty cycles are obtained from the gate level simulations explained in section 6.2. The algorithm prioritizes to exercise the internal nets with highly skewed duty cycles (closer to 1.0) since NBTI stress is highly sensitive to duty cycles. The next set of vectors aims to exercise all other critical nets. The algorithm outputs a set of exercise vectors V along with the list of the critical nets Ne which are exercised at least once by any one of the vectors. The algorithm initially starts with a vector having all bits as dont cares(vj = X) and then iteratively attempts to exercise (lines 7-17) as many critical nets as possible by trying to justify the individual bits of the current vector vj similar to the justification procedure for the activation phase of ATPG tests. Each justification vector may be able to activate/exercise more than one critical net on the timing path. All the nets which are exercised by the current vector vj are removed from the list of critical nets N and the current vector is added to the set of deterministic exercise vectors V. The procedure is iteratively executed with all the bits of the new vector being set to dont care (X) as seen in line 6 of the algorithm. The algorithm runs until all the critical nets are exercised by any of the vectors (line 5). It is highly unlikely that the algorithm does not terminate due to lack of exercise vector which can exercise the critical nets. Redundant nets Nred consists of such nets that cannot be exercised under any possible input vector justification and existence of such nets indicates a possibility of problem in synthesis of the gate level netlist. In our analysis, we did not encounter any redundant nets in the critical path logic circuit which could not be exercised using an exercise vector from the justification procedure.

The above algorithm proposed by Kim et al. aims to achieve the two goals of vector generation. Firstly, it generates a minimum number of vectors to exercise all the critical nets. It achieves this criterion by making sure that a large number of critical nets are exercised simultaneously by a single vector. The algorithm initially

targets a particular set of critical net explicitly and then simulates the vector values for any other internal nets that could be activated without explicitly being targeted in current iteration as detailed in lines 7-17. Secondly, the algorithm uses a large number of unspecified bits in the exercise vectors to reduce the vector size and helps in improving the hardware complexity due to additional ROM and MUXes. This is achieved using a novel justification procedure which specifies only the necessary bits of the current vector in each iteration of the algorithm to target a new set of critical nets as enumerated in line 8. The justification procedure is developed using a powerful PODEM-based ATPG tool developed by Neophytou et al.[21]. This tool accepts the gate level netlist having only universal gates such as NOR, NAND, NOT, AND, OR. We used a Python script to convert the Verilog netlist having compound gates into a bench format netlist containing only universal gates.

4.4 Use of existing DFT infrastructure

Design for Testability (DFT) is a crucial part of Integrated Circuit(IC) Design intended to add testability features . It makes the development of manufacturing tests for ICs easier. DFT infrastructure such as scan chains are incorporated in most modern processor designs. Scan chains can be used instead of the additional hardware overhead of PRITEXT microarchitecture to exercise the combinational logic using the appropriate input patterns (vectors). We give a brief introduction to scan chain testing here.

4.4.1 Scan chains

Scan Chains are the crucial components in the scan-based DFT techniques that are inserted in the IC designs to shift the test data into the chip and out of the chip. This ensures that every point in the combinational logic of the chip is controllable and observable. A scan chain is formed by a series of flip-flops connected back to

back in a chain formation with output of one flop Q connected to the input of another adjacent flop. The input of first flop , called as scan-in , is connected to the input pin of the chip from where scan data is fed. The output of the last flop, called scan-out, is connected to the output pin of the chip (which is used to take the shifted data out. The scan chains help to achieve two important goals of DFT. First, to test the stuck-at-faults in the manufacturing process of the semiconductor devices. Second, to test the paths in the manufactured circuits for possible switching delays to ensure the device works for the maximum frequency specification.

In order to enable scan chains, the regular flip-flops are modified in order to enable scan-in and scan-out of the input data. An additional multiplexer is introduced for every flip-flop to select between the scan input SI and the normal input D. The scan-enable input acts as the control signal. It has the following major steps:

- Assert scan-enable to enable SI to Q path for every flip-flop.
- Setup the desired inputs at each flip-flop by shifting in the scan data.
- De-assert scan-enable for one clock cycle to enable D to Q path so that the combinational cloud output can be captured at the next clock edge. All the target flip-flops have the intended values.
- Re-assert scan-enable and shift out the data through scan-out to check if the combinatorial test passed.

5. SUPERSCALAR MICROARCHITECTURE

Our work targets the reliability concern in superscalar processors due to accelerated physical wearout of devices in process technologies with nanoscale transistors. Such efforts need design and verification infrastructure at gate level (netlist) to incorporate new microarchitectural techniques which can mitigate the transistor aging due to NBTI Stress. Hence for our research, we need a synthesizable RTL design of superscalar processor core. We have used FabScalar[10], an open source toolset useful for computer architecture research.

5.1 FabScalar

FabScalar is an open source project which provides synthesizable Verilog RTL code and physical designs of superscalar processors with various microarchitectures and is developed by Choudary et al[10]. The various processor cores differ in three major dimensions of a superscalar design: size of internal structures for performing aggressive Out-of-order execution, pipeline depth and superscalar width. FabScalar automatically builds several superscalar configurations using Canonical Pipeline Stage Library (CPSL) which consists of multiple designs of logical processor pipeline stages having various pipeline widths and depths. Each logical pipeline stage corresponds to one of the canonical template stages: fetch, decode, rename, dispatch, issue, execute, writeback, and retire as illustrated in figure 5.1.

A particular superscalar processor core is built by picking one canonical pipeline stage design for each of the logical pipeline stages from the CPSL. These canonical stages are then stitched together to form a fully synthesized superscalar pipeline. Each pipeline stage can be sub-pipelined into multiple logical pipeline stages in order to improve clock frequency. For example, Fetch stage is pipelined into Fetch-1 and

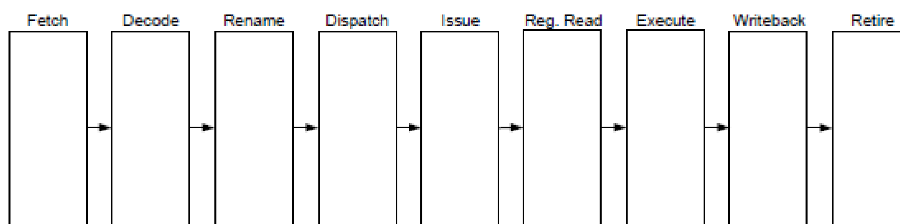

Figure 5.1: FabScalar processor core with canonical template stages

Fetch-2 as it implements a complex logic corresponding to several structures such as instruction cache, branch target buffer, branch predictor, next-PC etc.

5.2 Core microarchitecture

Using the above automated process of core generation, we obtained a synthesizable RTL design of a core with the configuration listed in figure 5.2. FabScalar uses PISA Instruction Set Architecture which is RISC based machine. PISA closely resembles MIPS ISA excluding load and branch delay slots. Figure 5.3 depicts various pipeline stages of a superscalar out-of-order processor. A brief description of the several canonical pipelines stages is provided here:

5.2.1 Fetch

In a superscalar processor, fetch stage is capable of fetching more than one instruction from I-Cache for every clock cycle, which in our case fetches 4 instructions. The primary responsibility of fetch stage is to feed the execution pipeline as many instructions as possible every cycle by ensuring maximum instruction fetching bandwidth. The performance of a superscalar core is limited by the fetch stage because the throughput of all the other pipeline stages depends on the fetch stage and hence the instruction completion bandwidth is bounded by fetch bandwidth. In every clock cycle, the fetch stage accesses the Instruction Cache using the Program counter and

Width							
Fetch	Decode	Rename	Dispatch	Issue	Register Read	Execute	Writebacl
4	4	4	4	4	4	4	4

Fetch Queu	Active List (ROB)	Physical Reg File	Issue Queue	Load Queue	Store Queue	Branch Predictor	Return Address
16	128	96	32	32	32	Bimodal	16

Branch order Buffer	Depth				
	Fetch	Issue	Wakeup select	Register Read	Fetch to Execute
16	2	2	2	1	10

Figure 5.2: Configuration of FabScalar processor core

fetches at most four instructions provided there is no misalignment in cache access. The next PC address is computed based on the branch predictor.

5.2.2 Decode

Instruction decoding stage performs extraction of individual instructions from the fetched group, identification of type of instruction, operands and dependencies among the instructions fetched. In a fixed length ISA, the instructions are quite simpler to be decoded and for dependence checks. The opcode field of each instruction gives enough information about the type of instruction and the operands. In simpler designs, decode stage is merged with register read stage. Since the existence of a branch is only known after decoding the instruction group, decode stage provides feedback to fetch stage in order to change the instruction flow in case of conditional and unconditional branch instructions.

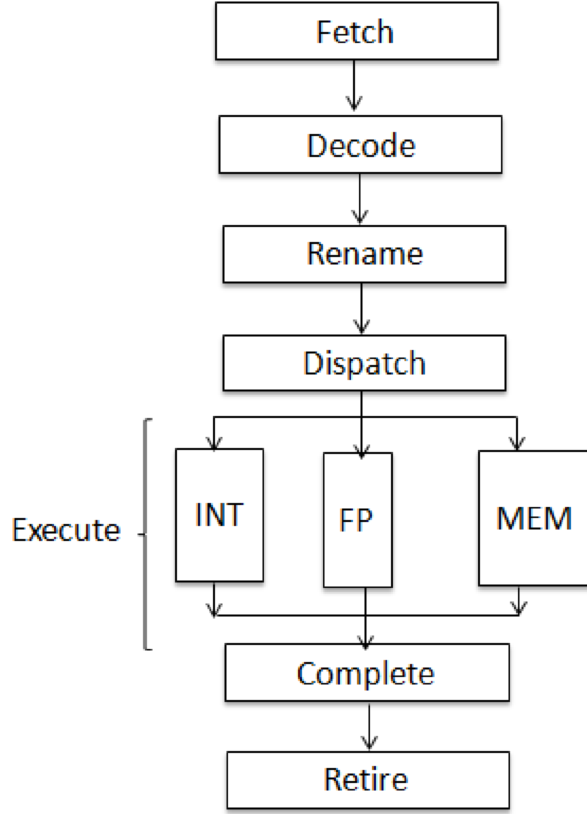

Figure 5.3: Generic superscalar out-of-order pipeline stages

5.2.3 Rename

It is a crucial phase of out-of-order execution. It performs renaming of register operands in order to avoid data hazards between instructions. Since the number of architectural registers is small and constant, several instructions use these limited number of registers without any necessary dependency between two instructions. Renaming helps to replace the architectural register name with a new name from Physical register file for each of the instruction being processed. This helps in eliminating the false dependencies (output dependence and anti dependence) between instructions. This allows the processor to issue instructions in out of program order

without loss of true dependence thereby keeping processor pipeline full even when the older instructions are not yet completed.

5.2.4 Dispatch, issue

In a superscalar pipeline, more than one instruction is processed in any pipeline stage. Considering a mix of heterogeneous functional units and different instruction types, each decoded instruction must be dispatched to corresponding functional unit for its execution. In an out-of-order processor, the dispatch/issue stage split the pipeline into two: in-order frontend and out-of-order backend. In a diversified superscalar pipeline, each functional unit can operate independently and only depends on the operands availability. Dispatch helps in efficient utilization of all functional units by feeding instructions which have all their operands in a valid state. Since there are inter-instruction dependences, instructions are buffered in reservation station before they are ready to be issued. Two types of reservation stations are commonly designed based on the placement of instruction buffers relative to instruction dispatching: centralized and distributed reservation stations. In several microarchitectures, dispatch and issue stages are merged. In general, dispatch refers to association of instruction types with the respective functional unit types. Issue refers to initiation of instruction execution in the functional unit. In a design with centralized reservation station, the instructions are associated with the functional units at the same time as their execution is initiated. Hence, dispatching and issuing are often used interchangeably.

5.2.5 Execute

It performs the actual instruction execution using several functional units. Specialized functional units are designed based on the instruction types to improve the execution throughput and overall performance. These functional units can be pipelined in order to improve the latencies of complex instructions such as floating

point operations. Broadly two types of functional units are used: Integer units and Floating point Units. Additionally branch and load/store units are used in the execute stage. Branch unit is used for updating Program Counter while Load/Store unit is used to access Data Cache. The choice of number of functional units is dependent on the application domain based on the observed instruction mix. Large numbers of functional units lead to complicated design due to broadcast and bypass stages.

5.2.6 Complete, retire

At the end of the instruction execution, the destination (renamed physical) register is updated and the instruction now resides in the completion buffer (Reorder Buffer). Depending on the instruction sequence, the instruction leaves the completion buffer and the results are updated to the architectural registers, thereby updating the architectural state of the machine. In case of store instructions where the destination is a memory location, the instruction is architecturally completed when the write is performed to a local store buffer. This buffer helps in two ways: holds the store to wait for the D-Cache write operation availability and helps to update the memory in order when stores are completed out-of-order with respect to program sequence. In the retire stage, the store operation completes the memory write and is said to be architecturally retired, thus updating the memory state of the machine. Hence, for non-store instructions, completion marks the retirement as well. The in-order update of memory and architectural state of the machine is necessary to honour the precise interrupts.

6. PRITEXT: PROCESSOR RELIABILITY IMPROVEMENT THROUGH EXERCISE TECHNIQUE

NBTI stress does not lead to hard failures but causes switching delay degradation in the long term leading to timing violations and reduced lifetime of processors as detailed in chapter 2. NBTI degradation is not uniform across all the paths in the device because different paths have different timing delays and different duty cycles along the transistors as pointed in the three observations from section 3.3. In short, the workloads stress each path differently leading to highly skewed duty cycles across some of the timing paths. Only the paths which do not have enough slack to overcome the degraded switching delay are highly prone to timing failures due to NBTI stress. All the paths having less than 10% slack are considered to be NBTI critical[13][7][14][11]. Hence, our work only targets critical timing paths of a superscalar processor design.

6.1 Timing critical paths in superscalar core

As mentioned in section 5.2, we have used a synthesisable Verilog RTL of a superscalar core for our work. We have synthesized the processor core for a clock frequency of 500MHz using Synopsys Design Compiler with 45nm TSMC standard cell library. All the paths with less than 10% slack were considered critical and were obtained using static timing analysis tool from Synopsys. We obtained 83 critical paths which can be broadly classified into three groups based on the corresponding pipeline stages as listed below:

- Group A consists of 70 paths. All these belong to Load Store Unit and are responsible for memory disambiguation logic.

- Group B consists of 8 paths. All these belong to Simple ALU unit which acts as a carry ripple adder.
- Group C consists of 5 paths. All these belong to Decode unit. The internal logic checks if each of the 4 instructions fetched in the current cycle are complex instructions (such as LDW: Load Double Word) which need to be split into two regular instructions (LW: Load Word).

It must be noted that the critical paths are highly sensitive to the design choices and the underlying microarchitecture of the superscalar core. A design with a different size of OOO structures (such as ROB, LSQ, Wakeup select logic) can have a completely different set of critical paths. However, our methodology is transparent to the microarchitecture of the superscalar processor which will be shown in further sections.

6.2 Workload characterization of critical paths

Since NBTI is highly sensitive to duty cycles of the transistors across the timing path, we have performed gate level simulations of the synthesized superscalar processor using six workloads (bzip2, gap, gzip, mcf, vortex and parser) from the SPEC CPU2000[1] benchmark suite to obtain the duty cycles. We have used the co-simulation environment provided by FabScalar in which a C++ based functional simulator runs concurrently with a cycle accurate RTL Verilog simulation of superscalar core[10]. The co-simulation environment helps in reducing the simulation time of standard benchmarks using check-pointing and avoiding a full-system simulation. Using the waveform dumps obtained from the gate level simulations of synthesized superscalar Verilog netlist, we have calculated the average duty cycles of all the internal nets along the critical timing paths listed in section 6.1.

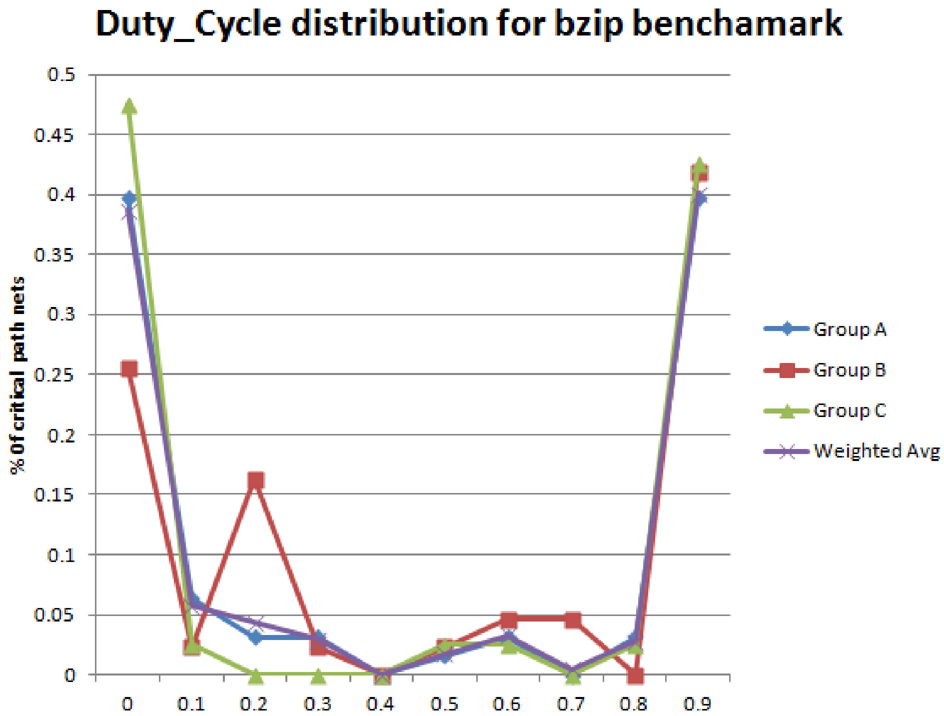

Figure 6.1: Duty cycle distribution of three critical path groups for bzip workload

Figures 6.1 through 6.6 illustrate the distribution of duty cycles of internal nets across all the critical paths of the three groups for each of the workload. X-Axis represents the duty cycle bin ranging from 0.0 to 0.9 while Y-Axis represents the fraction of the total internal nets having their duty cycles in a particular bin. The bin width is 0.1. For example, bin 0.5 represent all the duty cycles in the range [0.5, 0.6).

As illustrated in figures 6.1 through 6.6, the duty cycles of these critical paths are highly skewed towards the boundaries. On an average, 40% of critical nets have duty cycles in the range of [0.9, 1.0). This highlights the under utilization of several paths in the processor since these paths represent the corner cases in a processor workload and the probability of these cases is very small. This is analogous to the critical path in the priority scheduler module discussed in section 3.3 where the probability of the

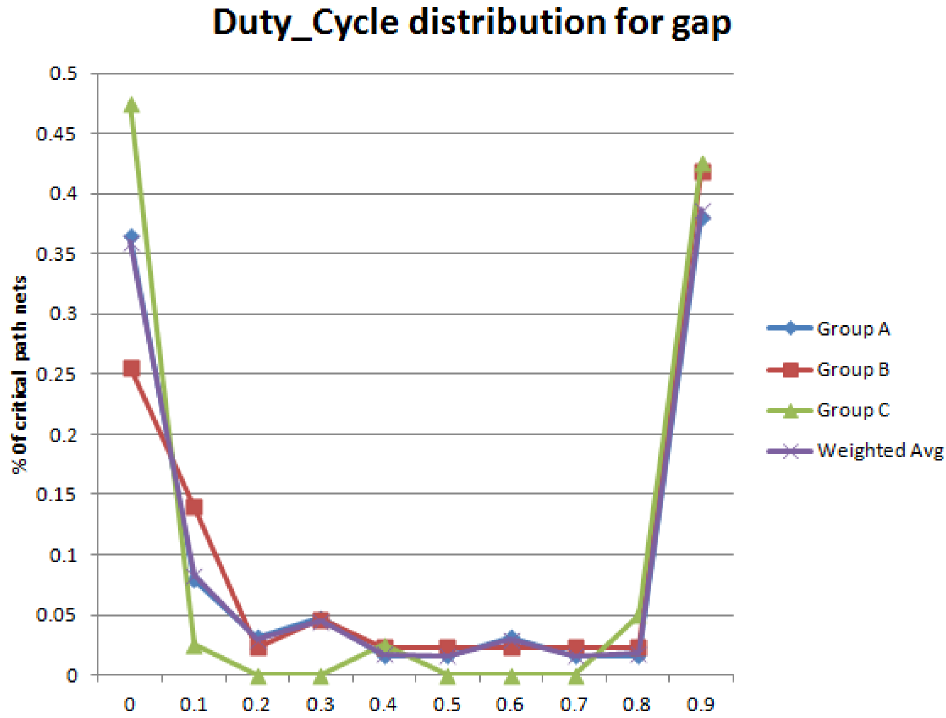

Figure 6.2: Duty cycle distribution of three critical path groups for gap workload

path representing the `D_req` is very small. A transistor with a duty cycle closer to 1.0 experiences maximum NBTI stress while a duty cycle closer to 0.0 leads experiences minimum NBTI stress as evident from the equation 2.6.

As mentioned earlier, the first timing failure in the processor determines the lifetime of the processor. Since the total delay degradation of each path is dependent on the duty cycles of the internal nets along the critical path, we computed the relative delay degradation dg for each of the 83 paths which are broadly classified into three groups in section 6.1. We found that paths in group A degrade 100x faster than the paths in Groups B and C because of the highly skewed duty cycles, thereby making Group A paths to be NBTI critical and life determining. All these paths correspond to the Load Store Unit. Figure 6.7 illustrates the reason behind selecting Group A paths to be NBTI critical. The slope of the line represents the

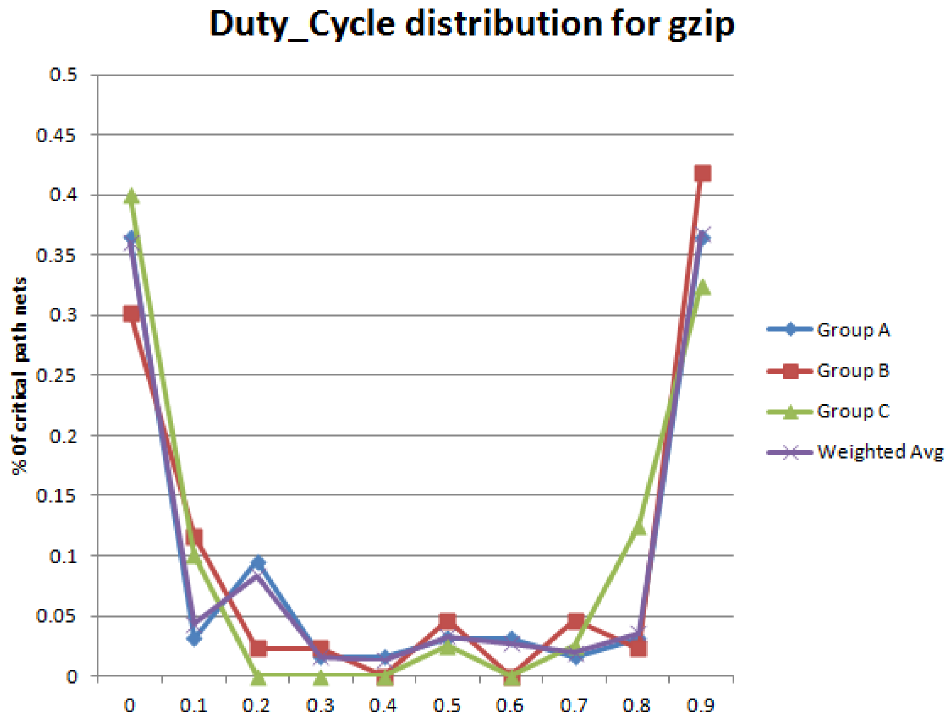

Figure 6.3: Duty cycle distribution of three critical path groups for gzip workload

delay degradation rate.

It is evident that

$$\texttt{t_A} << \texttt{t_B} < \texttt{t_C} \tag{6.1}$$

where `t_A` is the time at which the first timing failure occurs in the processor having only Group A paths, `t_B` is the time at which the first timing failure occurs in the processor having only Group B paths ,`t_C` is the time at which the first timing failure occurs in the processor having only Group C paths . Hence we need to reduce the wearout rate of Group A paths such that the first timing failure is delayed long enough to achieve the intended operational lifetime. Since the wearout rate of Group B and C paths is much smaller, the lifetime of the device is not limited by the wearout across these paths. Note that we use 10% of the cycle time T as the threshold for

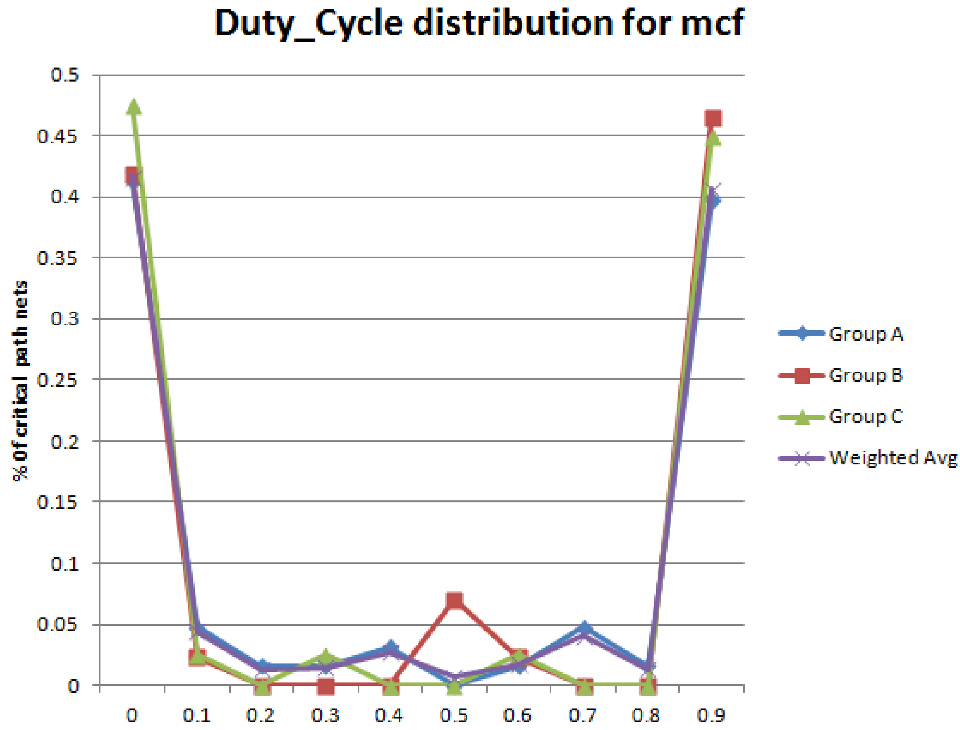

Figure 6.4: Duty cycle distribution of three critical path groups for mcf workload

timing failures. Using our PRITEXT method, we intend to balance the duty cycles in order to mitigate NBTI stress and improve Processor lifetime.

6.3 Analysis of load store unit critical path

All the 70 paths of the Group A lie in the Load Store Unit which performs a memory disambiguation check as illustrated in figure 6.8 .

Memory disambiguation is a key step in out-of-order superscalar processors [24]. In order to achieve high performance, Superscalar processors need to execute instructions in an order different from the program sequence. Load and Store instructions which can exhibit true data dependence need to be speculatively executed out of order in order to exploit the memory level parallelism. Consider the sequence of instructions listed below:

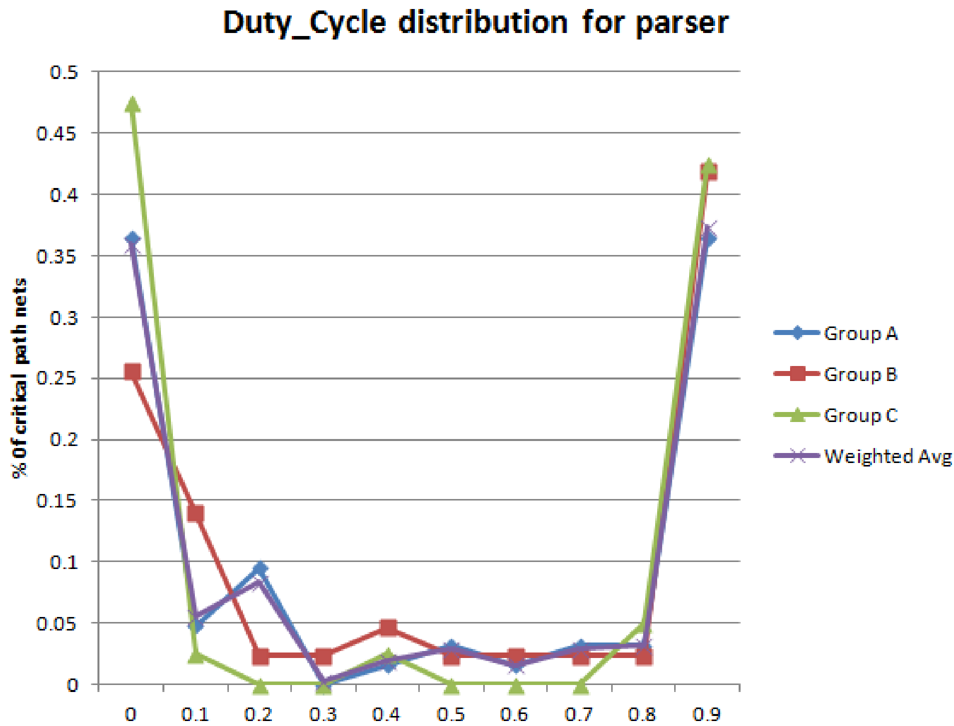

Figure 6.5: Duty cycle distribution of three critical path groups for parser workload

(1) ST R1, 5(R2)

(2) ST R5, 0(R6)

(3) LD R3, 10(R4)

(4) LD R7, 4(R8)

Consider the instruction 3 which performs a load operation. Since the memory address is not known ahead, executing Load instruction after all the previous Store instructions can lead to serious performance issues. It is difficult to predict if a load instruction performs a memory access from the same address which is written by a previous store until all the address are resolved. There are two approaches widely practised in superscalar designs[24] as listed below:

- Store-Load Forwarding: In case a store instruction is not yet retired due to

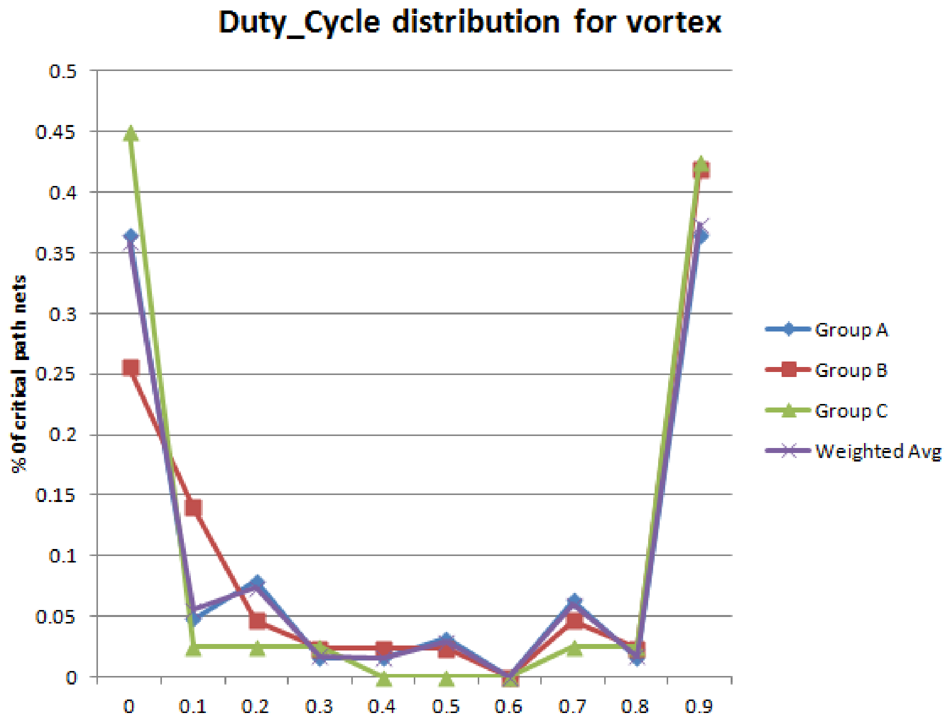

Figure 6.6: Duty cycle distribution of three critical path groups for vortex workload

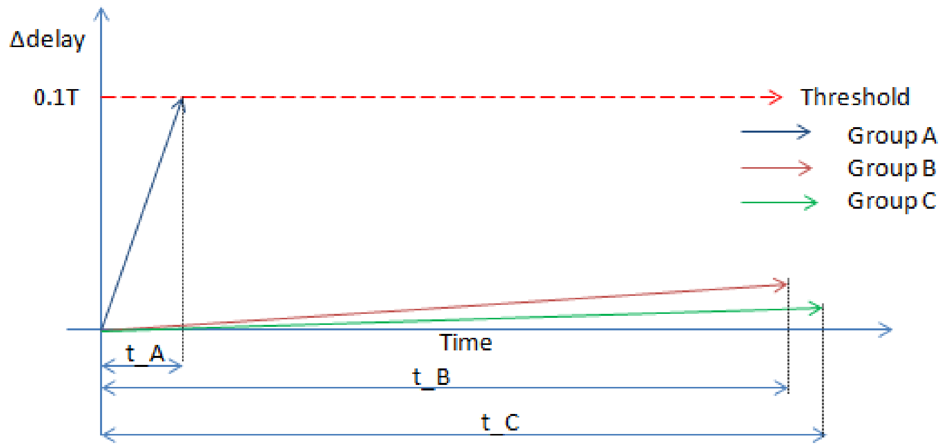

Figure 6.7: Increase in delay degradation for three critical path groups with time. The time at which the increase in delay exceeds the guardband (0.1T) is marked.

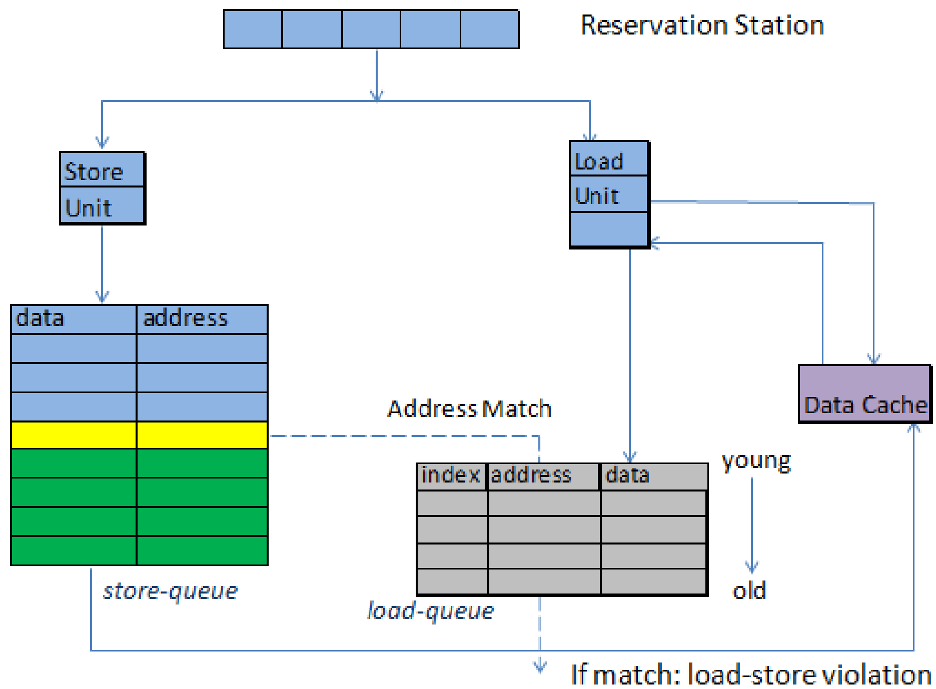

Figure 6.8: Critical path in load store unit: Memory violation check for load-store bypass

lack of D-Cache availability, the load instruction can be forwarded the result from the store buffer.

- Store-load Bypass: Speculatively execute the load instruction and place all the completed load instructions in the load queue. When a store is retired (finished), the address of the store (from the store-queue) is checked with every entry in the load-queue for a possible match. If a match is found, the speculative load instruction was fed a wrong data and hence is marked as memory dependence violation. If multiple entries in the load queue match with the address of the current store instruction, then the oldest load instruction is marked. Whenever the load instruction is about to be retired, the pipeline is flushed and all the following instructions are re-executed.

Since the address match performs an age-based priority search across the load queue having 32 entries, the timing path has several gates leading to long switching delays. Since the probability of a store instruction matching a load instruction at the end of load-queue (i.e. 32nd entry) is very small, the probability of utilization of the timing path, representing this corner scenario, is quite small leading to highly skewed duty cycles for the internal nets along this path making this the most critical path in our superscalar core.

6.4 PRITEXT

As mentioned in section 3.1, Input Vector Control can be used to exercise the internal nets on the critical path to balance the duty cycles, thereby mitigating the NBTI stress and improving the lifetime of the processor. Note that the exercise mode is enabled only when the processor is quiescent (i.e. in standby mode). In order to exercise all the critical nets, it is necessary to obtain the activation cone of all the critical nets. Hence the combinational logic cone which lies in the fan-in of the critical path is obtained using the synthesized netlist. Once the logic cone is obtained, we use the algorithm described in section 4.3 to obtain a minimum set of deterministic vectors to exercise all the critical nets. The resultant logic cone has 1426 inputs.

Figure 6.9 illustrates the PRITEXT microarchitecture. Along with the Load Store Unit combinational logic, it has additional Multiplexers and On-Chip ROM with the exercise vectors. The input vectors are fed to the extracted critical path logic cone when the exercise mode signal is enabled such that all the critical nodes are exercised with an aim to improve the average duty cycles. Since it is necessary to keep the architectural state of the processor unchanged, the output flip-flops are disabled during exercise mode.

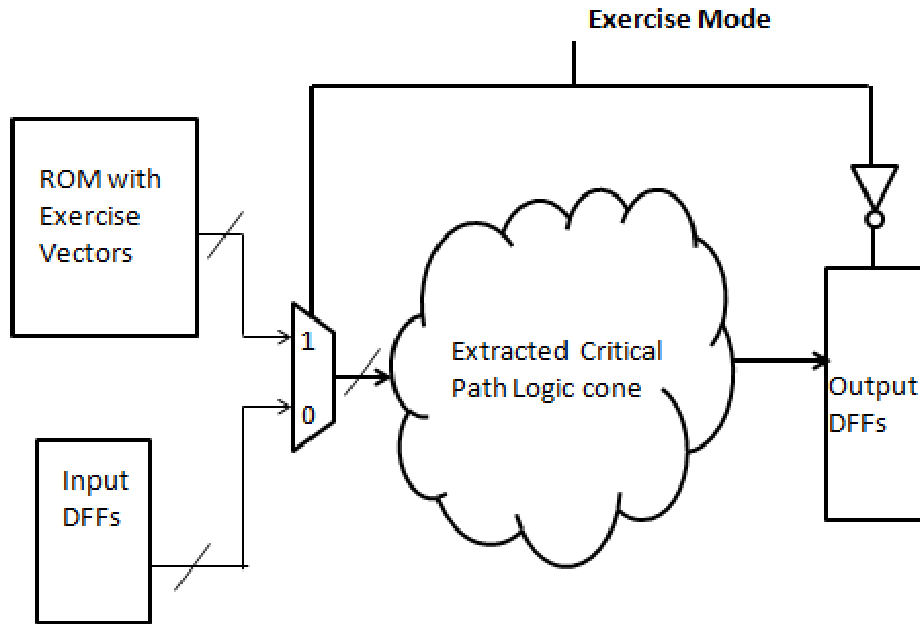

Figure 6.9: Proposed PRITEXT technique with On-Chip ROM , multiplexers and combinational logic cone of the critical path

In order to exercise all the critical nets, the input vectors are rotated after a pre-defined period of time(ROTATION PERIOD). Since the input vectors lead to additional internal switching, the rotation of the input exercise vectors is kept as minimum as possible by rotating the vectors only after 1024 cycles. This is achieved by using a hardware counter which switches the current vector to the next vector in the On-Chip ROM after every 1024 cycles. The vector rotation period has no impact on improving the duty cycles of the internal nodes because the duration of each vector remains the same irrespective of the frequency of rotation. Our PRITEXT microarchitecture has minimum hardware complexity since we perform vector compaction as described in section 4.2.

6.5 Exercise mode during processor standby

Modern processors are not used continuously throughout their lifetimes. Most of the times, processor is quiescent waiting for an I/O access, Network access, TLB miss or off-chip DRAM access. This provides us the opportunity to enable the exercise mode whenever the processor is idle (Standby mode). In order to make sure all the instructions in the superscalar pipeline are retired before enabling the exercise mode, we delay the exercise mode signal by 100 clock cycles to finish the execution of all the instructions in the processor pipeline. Unlike power gating which needs significant idle time in the order of milliseconds (Millions of clock cycles)[18], PRITEXT can balance the duty cycle in a smallest window of opportunity when the processor is idle for even a small window of time in the order of 100s of clock cycles.

Based on the real-time statistics collected for processors in several server clusters, we observe that processors experience much higher idle time when compared to active time. Figure 6.10 provides a snapshot of the processor usage statistics in one of the departmental servers used for academic purposes. The processor is idle for 98% of the time while it is in active execution (USER) mode for only 2% of the time. The ratio of Active to Standby time is termed as RAS. Based on above statistics, RAS is equal to 2:98 = 1:49. However, in the evaluation of the proposed technique, we have considered a conservative value of 1:1 and 3:1 for RAS ratio. A higher standby time gives a better opportunity to balance the duty cycles as illustrated through a working example here.

The average duty cycle of a critical net which depends on various factors as given by the following equation:

$$AverageDutycycle = \left(\frac{a * RAS}{RAS + 1}\right) \tag{6.2}$$

```
top - 23:41:25 up 90 days, 42 min, 20 users,  load average: 1.21, 1.72, 1.89
Tasks: 983 total,   2 running, 979 sleeping,   2 stopped,   0 zombie
Cpu(s):  2.1%us,  0.0%sy,  0.0%ni, 97.9%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:  264148004k total, 39417088k used, 224730916k free,   197720k buffers
Swap: 266237204k total,   194644k used, 266042560k free, 34372256k cached
```

Figure 6.10: Real-time statistics of processor utilization in a server

where a is the duty cycle of the internal net in active mode, RAS is the time ratio of the active to standby mode and e is the duty cycle of the internal net in Standby(i.e. exercise) mode. For example, lets assume the following values in calculation of average duty cycle: a = 0.99, e= 0.1, RAS = 1

$$AverageDutycycle = \left(\frac{0.99 * 1 + 0.5}{1 + 1}\right) = 0.745 \tag{6.3}$$

The ratio of active to standby mode has a significant impact on the average duty cycle and hence on the lifetime of the device. Considering a RAS of 1:3, the average duty cycle is equal to

$$AverageDutycycle = \left(\frac{0.99 * 0.333 + 0.5}{0.333 + 1}\right) = 0.622 \tag{6.4}$$

As stated in section 3.2, even a small improvement in beta (duty cycle) can lead to a significant improvement in lifetime of device.

6.6 PRITEXT enhanced ASIC design flow

Figure 6.11 shows the enhanced ASIC design flow using the proposed PRITEXT technique. All the steps in the flow are automated using scripts which helps the designers to integrate our methodology into the processor design flow without the necessity of any additional efforts.

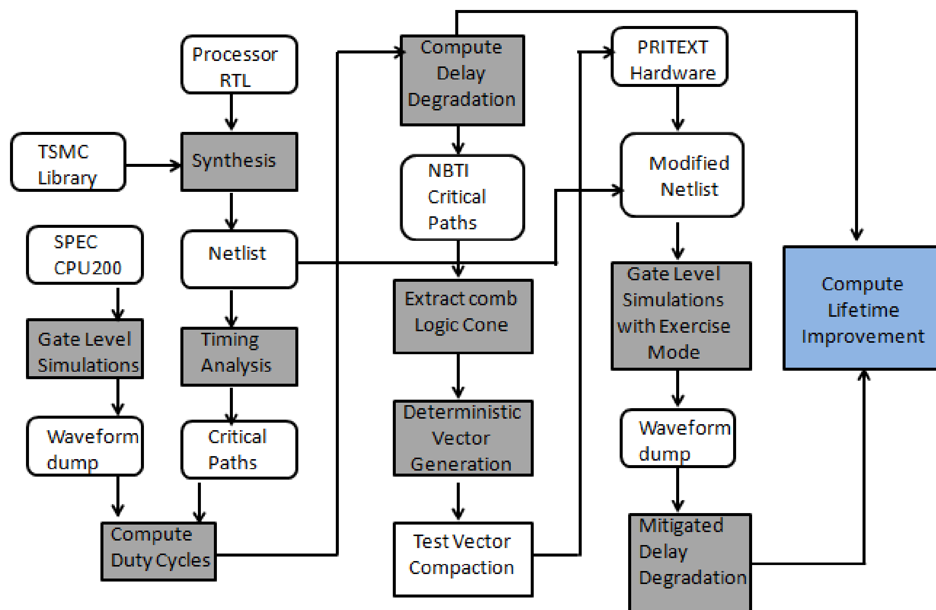

Figure 6.11: Automated ASIC design flow enabled with PRITEXT lifetime improvement technique

7. EVALUATION

In this chapter we discuss the experimental setup used in evaluating the benefits of PRITEXT technique. We then outline the lifetime improvement achieved in various scenarios and give an estimation of Power and Area overheads of the proposed technique.

7.1 Experimental setup

As mentioned in section 5.2 and 6.2, we have used a Synthesizable Verilog RTL code of a superscalar core obtained from the FabScalar project for our evaluation. We have synthesized the processor core for a clock frequency of 500MHz using Synopsys Design Compiler with 45nm TSMC standard cell library. All the paths with less than 10% slack were considered NBTI critical and were obtained using static timing analysis tool from Synopsys. We have performed gate level simulations of the synthesized superscalar processor using six real-time workloads(bzip2, gap, gzip, mcf, vortex and parser) from the SPEC CPU2000 benchmark suite to obtain the duty cycles. We have used the co-simulation environment provided by FabScalar in which a C++ based functional simulator runs concurrently with a cycle accurate RTL Verilog simulation of superscalar core. Using the waveform dumps obtained from the gate level simulations of synthesized superscalar Verilog netlist, we have calculated the average duty cycles of all the internal nets along the critical timing path listed in section 6.2.

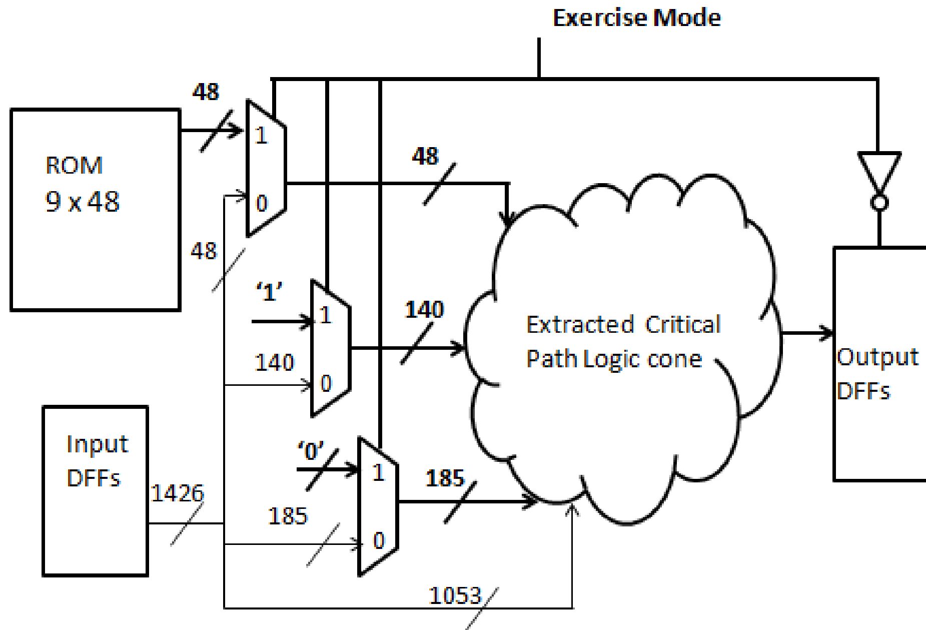

Figure 7.1: PRITEXT hardware with appropriate ROM vectors and multiplexers after vector compaction

7.2 Results

7.2.1 Vector generation results using deterministic algorithm

We have used the algorithm discussed in section 4.3 to obtain a minimum set of deterministic vectors with an aim of balancing the duty cycles of the critical nets. As mentioned in section 6.2, only the critical path in the Load Store Unit was found to be the lifetime deciding path. Hence our vectors used in the PRITEXT technique balance only the nets along the load store unit critical path. Figure 7.1 illustrates the PRITEXT microarchitecture containing the Multiplexers and On-chip ROM with exercise vectors. The extracted combinational logic cone of Load Store unit critical path consists of 1426 primary inputs, 15,194 internal nodes and 63 unique internal nets on the critical path.

Using the algorithm outlined in section 4.3, we have obtained nine exercise vectors

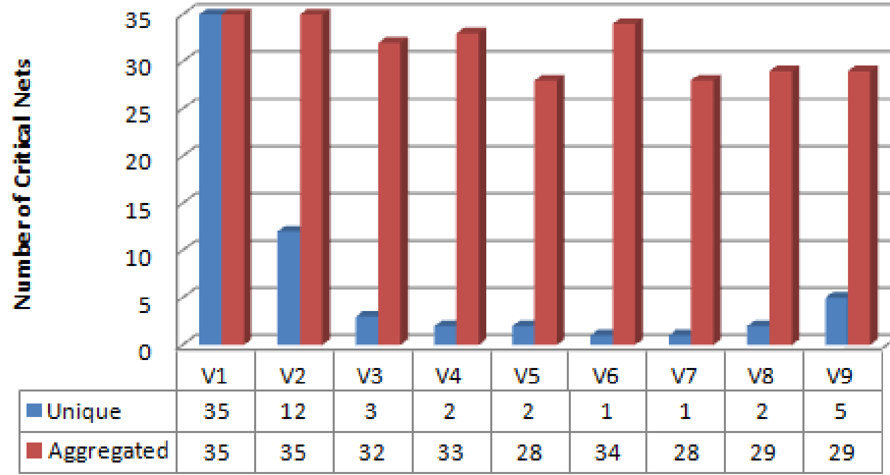

Figure 7.2: Accessibility of internal nets with each input vector.

which can exercise these 63 critical nets during the exercise mode. In order to minimize the hardware overhead of PRITEXT technique, we have performed test vector compaction to minimize the Multiplexer and On-Chip ROM overhead. Note that the width of each exercise vector is equal to the number of primary inputs of the extracted combinational logic. From 1426 primary inputs, each of which can possibly contribute to a MUX and On-Chip ROM entry, 1053 have dont care values which do not contribute to the MUX and On-Chip ROM complexity. 140 inputs can be set of constant logic value of 1 and 185 can be set of constant logic value of 0 while the rest of 48 inputs have unique values for each of the nine vectors. Therefore, the On-Chip ROM (complexity = 9 x 48) contains nine vectors each with 48 bits. A total of 373 (= 140 + 185 + 48) Multiplexers with 140 MUXes having a constant value of 1 and 185 having a constant value of 0 are needed as shown in figure 7.1.

As mentioned in section 3.1, each vector can only activate/exercise a small number of critical nets which is the reason why the algorithm generates multiple vectors.

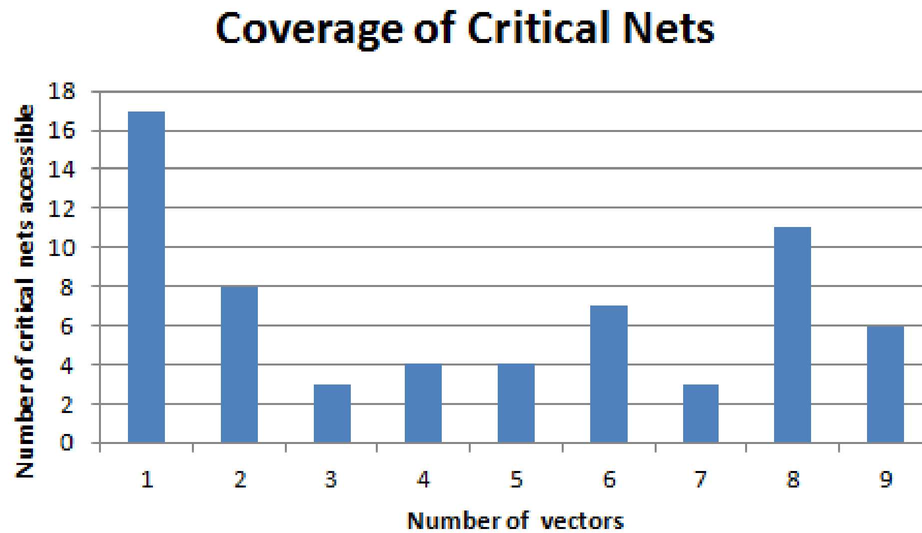

Figure 7.3: Coverage achieved by a set of input vectors. X-axis represents the number of vectors; Y-Axis represents the number of critical nets activated. Example: two of the nine vectors can activate 8 critical nets individually.

Figure 7.2 illustrates the coverage achieved by each vector in balancing the duty cycle of 63 critical nets in the Load Store unit path. Vector V1 exercises 35 critical nets and so does Vector V2 (Aggregate Histogram). However, Vector V2 exercises only 12 new critical nets (Unique histogram) that were not exercised by V1, which leads to 23 (=35 -12) of the critical nets to be exercised by both V1 and V2.

Figure 7.3 illustrates an interesting plot obtained from the above distribution and gives an intuitive explanation of the activation ability of the critical nets. 17 out of 63 critical nets were only accessible by one out of the nine vectors, while 8 critical nets were accessible by two vectors(need not be the same two vectors). Similarly 6 out of 63 nets were accessible by all the nine vectors. In an ideal case, we wish to develop a vector which can access/activate all the 63 critical nets. Since each vector is enabled for a constant pre-determined amount of time in exercise mode (1024 cycles) , 17 of the 63 critical nets are at logic 1 for a fraction of 1/9th of exercise

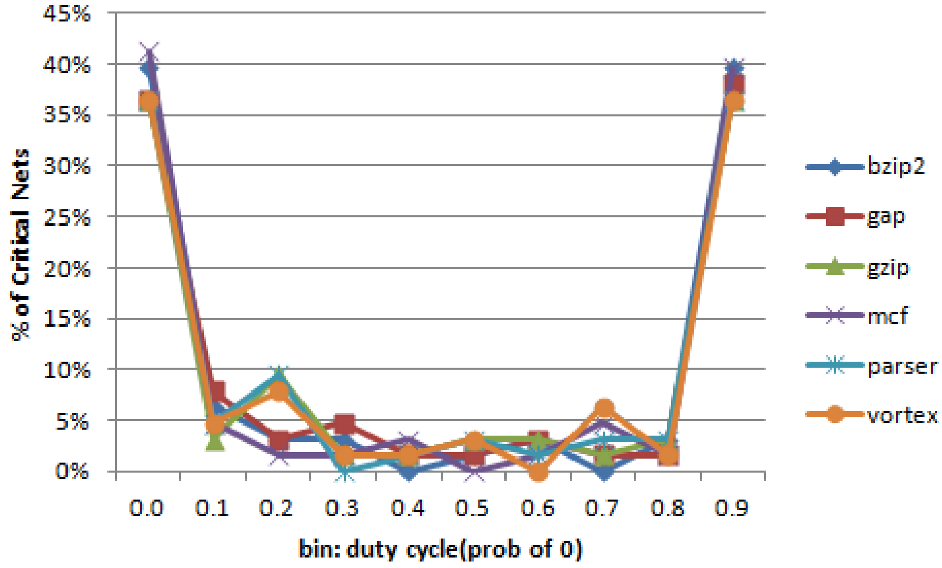

Figure 7.4: Duty cycle distribution of load store unit critical path of reference system.

mode duration, 8 of the critical nets are at logic 1 for a fraction of 2/9th and so on. The average duty cycle (fraction of time at logic 0) of the all the critical nets is calculated as below:

$$\beta = 1 - \left(\frac{1}{63}\right) * \Big[\big(17 * \frac{1}{9}\big) + \big(8 * \frac{2}{9}\big) + \big(8 * \frac{2}{9}\big) + \big(3 * \frac{3}{9}\big) + \big(4 * \frac{4}{9}\big)$$
$$+\big(4 * \frac{5}{9}\big) + \big(7 * \frac{6}{9}\big) + \big(3 * \frac{7}{9}\big) + \big(11 * \frac{8}{9}\big) + \big(6 * \frac{9}{9}\big)\Big]$$

$$\beta = 1 - (31.44/63) = 0.51 \tag{7.1}$$

7.2.2 Balanced duty cycles using deterministic exercise vectors

Figure 7.4 illustrates the duty cycle distribution of critical nets in the Load Store unit across six real-time workloads for a reference system with no lifetime improvement technique.

It can be inferred that approximately 40% of the critical nets have their duty

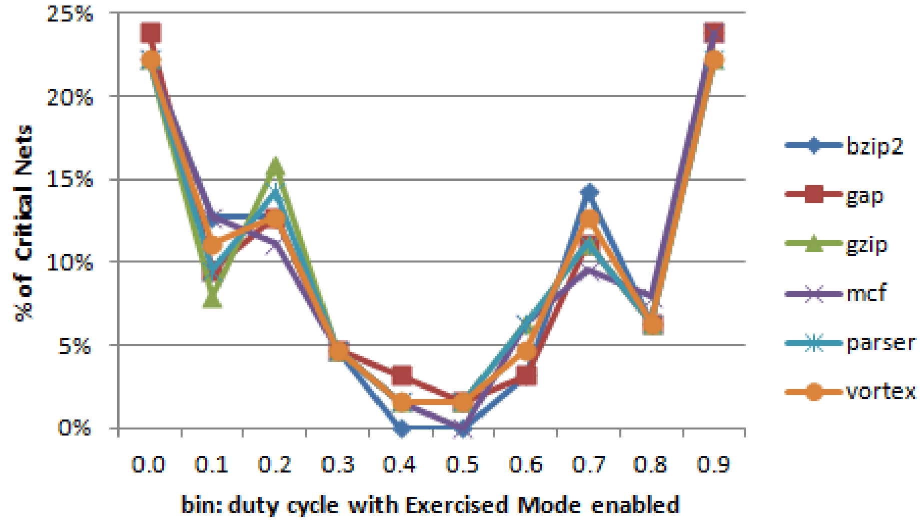

Figure 7.5: Duty cycle distribution of load store unit critical path with RAS = 3:1 in a system enabled with PRITEXT.

cycles in the range of [0.9, 1.0) which leads to accelerated NBTI degradation. Our aim is to reduce the average duty cycle of the critical nets using the exercise vectors. As mentioned in section 6.5, RAS has a significant impact on the average duty cycle.

Figure 7.5 illustrates the duty cycle distribution of critical nets in the Load Store unit across six real-time workloads for a device enabled with PRITEXT technique having a RAS of 3:1. It is quite evident from the figure 7.5 that fraction of critical nets having skewed duty cycles i.e. in the bin 0.9 has reduced from 40% to 23% on average. A value of 3:1 for RAS is very conservative allowing the device to be in standby mode only for 25% of its lifetime.

Figure 7.6 illustrates the duty cycle distribution of critical nets for a device enabled with PRITEXT technique having a RAS of 1:1. It can be observed from figure 7.6 that the fraction of critical nets having skewed duty cycles i.e. in the bin 0.9 has reduced from 40% to 13% on average when compared to reference system with no lifetime extending support.

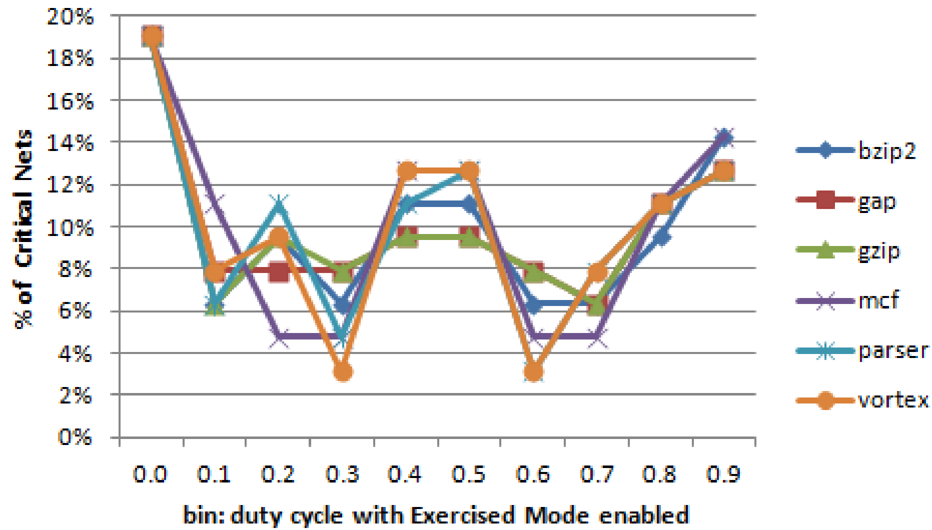

Figure 7.6: Duty cycle distribution of load store unit critical path with RAS = 1:1 in a system enabled with PRITEXT.

7.2.3 Lifetime acceleration with RAS = 3:1

Figure 7.7 depicts the lifetime improvement in superscalar core using PRITEXT technique for various real-time workloads with ratio of active to standby mode (RAS) equal to 3:1. Lifetime improvement is quantized using Acceleration Factor described in section 2.4. Our lifetime improvement technique achieved an average of 3.8x lifetime improvement over a reference system using deterministic vectors in exercise mode.

7.2.4 Lifetime acceleration with RAS = 1:1

Figure 7.8 depicts the lifetime improvement in superscalar core using PRITEXT technique with ratio of active to standby mode (RAS) equal to 1:1. The device is in standby mode for 50% of its operational time. Our lifetime improvement technique achieved an average of 4.5x lifetime improvement over a reference system using deterministic vectors in exercise mode. The maximum improvement of 12.7x is observed

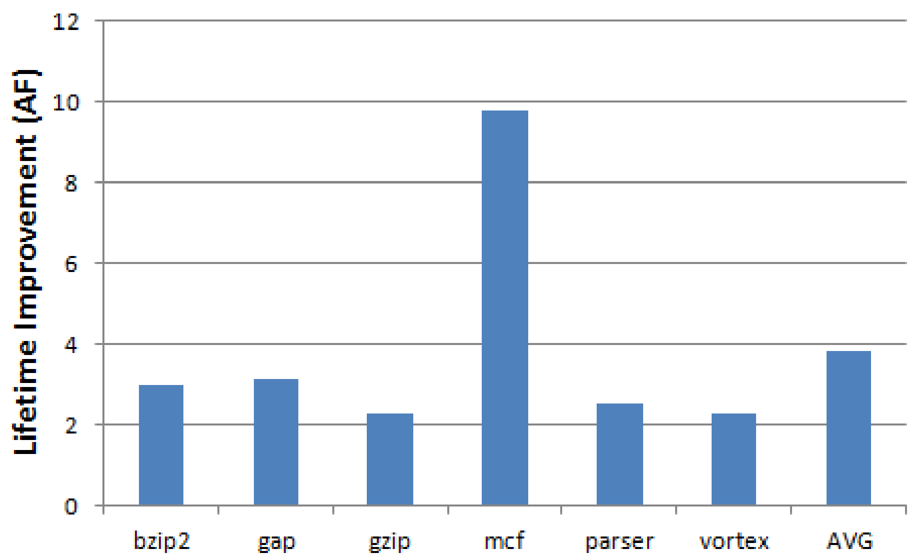

Figure 7.7: Lifetime improvement with RAS = 3:1.

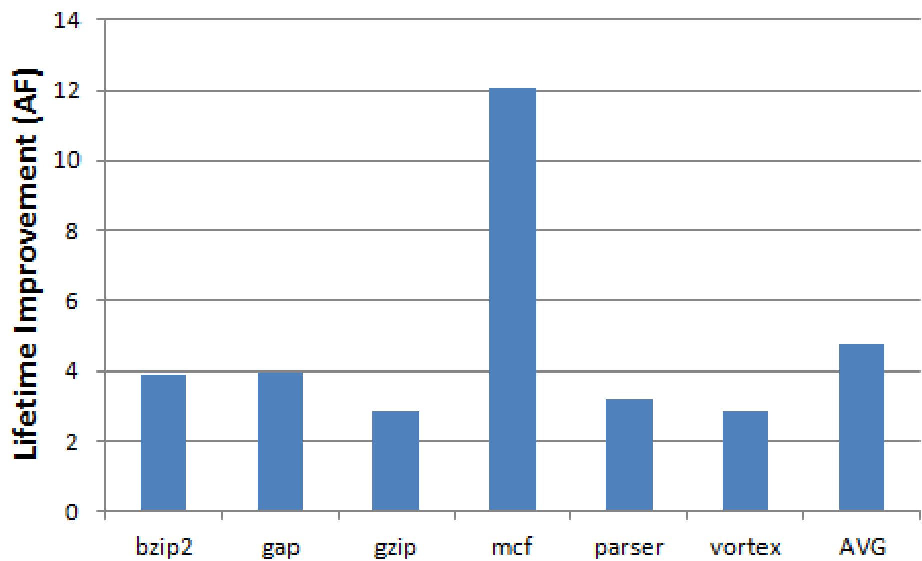

Figure 7.8: Lifetime improvement with RAS = 1:1.

for mcf workload.

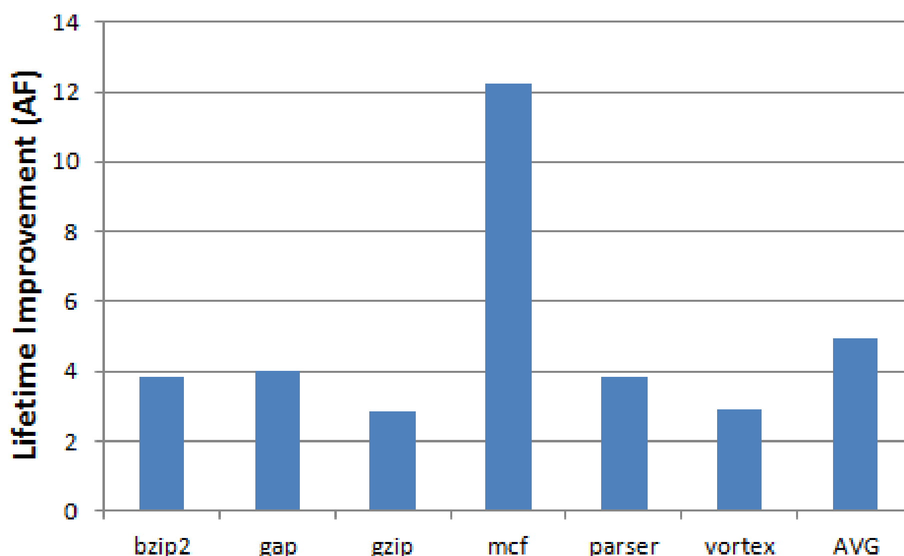

Figure 7.9: Lifetime improvement with unequal rotation periods

7.2.5 Lifetime acceleration with unequal vector rotation periods

As discussed in 6.4, each vector is rotated after a pre-defined number of cycles equal to ROTATION PERIOD. Based on figure 7.2, it is evident that certain vectors can activate more number of unique nets on the critical path when compared to other input vectors. Based on this observation, we explored the option of unequal ROTATION PERIOD giving more priority to Vector V1 and V2. Figure 7.9 illustrates the lifetime improvement for various benchmarks.

Our lifetime improvement technique achieved an average of 4.9x lifetime improvement over a reference system using deterministic vectors with unequal rotation periods in exercise mode.

The mcf workload constantly led to high lifetime improvement because of its memory intensive characteristic. It exhibits highly skewed duty cycle among the critical nets when compared to other benchmarks, thereby giving a better opportunity to balance the duty cycle and achieve a significant improvement in lifetime of the super-

scalar processor. The most skewed critical net had a duty cycle of 0.999999942 under mcf workload while the most skewed critical net had a duty cycle of 0.99994787 for all other workloads. Hence, based on equation 2.2, the worst case relative threshold voltage degradation under mcf workload compared to other workloads is proportional to

$$\left(\frac{\left(\frac{0.999999942}{1-0.999999942}\right)}{\left(\frac{0.99994787}{1-0.99994787}\right)}\right)^{n} = 898.8^{n} \tag{7.2}$$

where n is the time exponent. A small improvement in the duty cycle of the critical nets for mcf workload can improve the threshold voltage degradation significantly when compared to other workloads because of the sensitivity of the NBTI stress to duty cycle.

7.3 Power and area overheads of PRITEXT

7.3.1 Area overheard

PRITEXT uses additional Multiplexers and On-Chip ROM which contribute to area overhead of the superscalar processor as discussed in section 6.4. However we have used vector compaction technique as described in section 7.2.1 which helps minimizing the hardware complexity. We used Synopsys Design Compiler to estimate the additional area overhead and found it to be less than 0.5% of total die area. This is because the majority of the die area is occupied by other microarchitectural components such as Caches, Decoder, Rename, Physical Register File, Instruction window, ROB and other buffers described in section 5.2.

7.3.2 Power analysis

The additional hardware complexity due to Multiplexers and On-Chip ROM together with the switching of internal nets through exercise vectors in standby mode

leads to additional static and dynamic power consumption. We have performed power analysis of the superscalar core in active and standby modes using Synopsys PrimeTime tool. The additional power consumption was less than 1% as the majority of the power consumption was from complex microarchitectural components of the superscalar core and moreover the internal switching of the critical nets was kept to minimum by rotating the vectors infrequently (every 1024 clock cycles).

8. RELATED WORK

Broadly, there are three categories of work related to NBTI degradation: Modelling, Analysis and Mitigation of NBTI stress. Bhardwaj et al. [6] have proposed a comprehensive model for NBTI stress on PMOS transistor and predicted the threshold voltage degradation for short term and long term circuit operating conditions. The dependence of NBTI degradation on process and device parameters was accurately modeled and evaluated with respect to HSPICE simulations using 90-nm technology.

Wang et al. [28] proposed a unified reliability model of Hot Carrier Injection and Negative Bias Temperature Instability for FinFETs using reaction-diffusion theory. The authors also investigated the performance degradation in circuits due to HCI and NBTI in order to monitor the aging of Integrated Circuits. Lu et al. [19] have presented a statistical framework for characterizing the reliability of ICs fabricated using current CMOS process technologies by simultaneously accounting for aging due to operational NBTI stress and foundry process variations. Their work also provides a statistical analysis method for estimating the impact of individual circuit node on the overall reliability of the circuit functionality.

Ubar et al. [25] have proposed an approach to identify NBTI critical paths in nano-scale logic by analyzing three important parameters : delay-critical paths, gate input signal probability and the gate fan-out degree along the timing paths based on the observation that delay-critical paths are highly prone to timing failures. Ebrahimi et al. [11] have proposed a two step process to select Representative Critical paths to estimate the aging induced degradation in FPGA circuits. In the first step, all the paths , all the paths having delay greater than critical timing specification are

selected and are referred as Pseudo Critical Paths . In the second step, a subset of these paths are selected based on several factors such as fan-out, path delay, temperature, duty cycle, switching activity and physical location of the paths in the FPGA block. The authors also present a sensor insertion algorithm to monitor the aging of the critical paths.

Several authors have proposed techniques to mitigate NBTI degradation for improving the lifetime of processors. Abella et al.[5] proposed NBTI aware processor called Penelope which implements several generic strategies to reduce NBTI stress in storage and combinational blocks. They observed that many combinational blocks are idle for a significant fraction of operational time and proposed to use special inputs during idle periods alternatively. This ensures that the alternating special inputs tend to degrade different PMOS transistors such that the maximum degradation of a single PMOS transistor is well below the acceptable guardband. In order to mitigate PMOS degradation in Memory-like blocks such as bit cells, the authors proposed to write special values in empty entries such that the logic 0 and logic 1 are stored each for 50% of the time in the bit cells, thereby ensuring all the PMOS transistors in the inverters of the bit cells have similar degradation.

Gunadi et al.[14] have proposed Colt duty cycle equalizer which equalizes the usage frequency of devices in order to balance the utilization of internal components in a processor. It balances the duty cycle and the activity factor of circuits internal nodes in order to recover the NBTI stress across the aged transistors. The authors employ multiple techniques to mitigate the effects of aging stress. First, in Complement Mode Execution, true and complement forms of execution are employed in the storage structures, data path and control path for alternating coarse grained epochs which helps in balancing the duty cycles of the internal PMOS transistors in the circuit. Second, in Cache Set Rotation technique, LFSR Hashing is used to change

the cache index function with an aim of distributing the usage frequencies of cache entries. This helps in achieving uniform cache utilization for all memory accesses. Third, in Operand Identifier Swapping technique, the utilization of left and right operands on the data paths are equalized.

Several authors have exploited Input Vector Control mechanism discussed in section 3.1 to mitigate NBTI stress. Since the problem of finding the best input vectors is NP-Complete, the authors have taken several approaches to achieve approximate solutions such as Binary Integer Linear Programming, Mixed Integer Linear Programming, random simulations, heuristics based optimizations etc[27][13][7][12][29]. Yu Wang et al. [27] proposed a co-simulation flow for static leakage power and NBTI induced degradation by considering the ratio of active to standby time (RAS). They evaluated the transistor level NBTI modelling and path-based NBTI aware timing analysis in their simulations. They came up with efficient input vectors in order to achieve two objectives: mitigate NBTI induced circuit degradation and reduce the leakage power. Using exhaustive random vectors and probability based algorithms, the authors generated an ideal set of input vectors to meet the design objectives.

Firouzi et al.[13] proposed an efficient input vector generation technique using Linear Programming (LP) for reducing the NBTI stress in standby phase. The primary objective is to minimize the post-aging critical path delay. The authors represent the logic network obtained from the synthesized netlist and the NBTI-induced gate delay increase relations by Linear Programming constraints. All the path delays are represented as constraints in a linear function with the input vectors as the variables. The Objective Function (OF) is to minimize the aggregate increase in the circuit switching delay due to accelerated aging. The authors proposed three possible LP approaches: Binary Integer LP, Relaxed LP and Mixed LP. Binary Integer LP provides an optimal solution but has the drawback of huge runtimes for

large circuits. Relaxed LP converges in smaller runtime with reasonable accuracy in the results. Mixed Integer LP provides a trade off between the runtimes and accuracy of the input vector results.

In their next work, the authors [18] have proposed to use NOP (No Operation) Instruction to minimize NBTI effect since the processor spends a significant amount of time executing NOP instructions. They have observed that the source operands of the NOP instruction have a significant impact on the NBTI aging. Hence the authors have proposed to replace the original NOP with modified NOP instruction that has no impact on the program execution. Similar to their earlier work done by authors in [6], Linear Programming approach was used to find the best Maximum Aging Reduction (MAR) NOP instruction with appropriate opcode and source operand values. The authors have investigated two unique approaches to use the modified NOP in the program execution. In the software based approach, the compiler directives are modified to generate the binary code with modified NOP and the necessary operands are initialized using a set of additional instructions. In the hardware based approach, the pipelines stages are modified to include additional multiplexers which directly feed the optimized MAR NOP operands to the pipeline stages (such as ALU).

Bild et al.[7] have combined Input vector Control with Input Node Control techniques to mitigate NBTI stress in digital circuits. Input Node Control refers to the technique of inserting control nodes at the output of individual gates in order to drive the gate to a specific logic level. The outputs can be forced to logic 1 in order to reduce the stress on PMOS transistor. The problem of finding the optimum location for node control is proved to be NP-Complete by the authors. Hence the authors formulated the problem of finding the optimal set of internal nodes for insertion as a mixed integer linear problem. The task is formulated as finding the insertion points to minimize the critical path delay when the transistor is under NBTI degradation.

Each Boolean gate is modeled as a set of constraints that influence the output delay based on the input values (vectors).

All the prior approaches share the idea of using Input Vector Control in mitigating NBTI stress which is also the central idea of our work as described earlier. However the runtimes of these approaches could become intractable for large circuits. Our work uses existing ATPG algorithms in generating the input exercise vectors which are known to be accurate and fast. Further advancements in ATPG algorithms will help to improve the capability and runtimes of PRITEXT technique.

9. CONCLUSION AND FUTURE WORK

Negative Bias Temperature Instability (NBTI) is a prominent wearout mechanism in deep submicron process technologies which degrades the reliability of current semiconductor devices. Improving reliability of superscalar processors is necessary for ensuring long operational lifetime which obviates the necessity of mitigating these physical wearout mechanisms. Exercising the dormant critical components in the processor to balance the duty cycles of the internal nets has been proven to reduce the NBTI stress. We use a novel ATPG justification algorithm to generate a minimal set of deterministic input vectors. We then propose and evaluate a new microarchitectural technique PRITEXT, which uses these input vectors in exercise mode to effectively reduce the NBTI induced aging and improve the operational lifetime of superscalar processors. PRITEXT, which exploits Input Vector Control mechanism, leads to a 4.5x lifetime improvement of superscalar processor on average with a maximum lifetime improvement of 12.7x. Our work is transparent to the microarchitecture of the device and can be embedded as a part of ASIC design flow. All the steps in our proposed technique can be automated using scripts with minimal intervention of designers.

The efficiency of PRITEXT in balancing the duty cycles depends on the deterministic vectors. In future work, we wish to develop optimized algorithm using advanced ATPG techniques to generate a much smaller number of vectors with larger number of unspecified bits. This will result in reduction of On-Chip ROM size and number of multiplexers, thus leading to reduced power and area overheads.

REFERENCES

[1] SPEC CPU200 Benchmark suit : Standard Performance Evaluation Corporation. https://www.spec.org/cpu/.

[2] Process integration, devices, and structures (pids),. http://www.itrs.net/itrs%2019992014%20mtgs,%20presentations%20&%20links/2009itrs/2009chapters2009tables/2009pids.pdf, 2009.

[3] Failure mechanisms and models for semiconductor devices, jep122g,. http://www.jedec.org/sites/default/_les/docs/JEP122G.pdf, 2011.

[4] A. Abdollahi, F. Fallah, and M. Pedram. Leakage current reduction in cmos vlsi circuits by input vector control. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(2):140–154, Feb 2004.

[5] J. Abella, X. Vera, and A. Gonzalez. Penelope: The nbti-aware processor. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 85–96, Dec 2007.

[6] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula. Predictive modeling of the nbti effect for reliable design. In *IEEE Custom Integrated Circuits Conference 2006*, pages 189–192, Sept 2006.

[7] D. R. Bild, G. E. Bok, and R. P. Dick. Minimization of nbti performance degradation using internal node control. In *2009 Design, Automation Test in Europe Conference Exhibition*, pages 148–153, April 2009.

[8] M. A. Breuer, M. Abramovici, and A. D. Friedman. *Digital systems testing and testable design*. Wiley-IEEE Press, New Jersey, USA, 1st edition, 1990.

[9] M. Bushnell and V. D. Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer Science and Business Media, New York, USA, 1st edition, 2000.

[10] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwie, S. Navada, H. H. Najaf-abadi, and E. Rotenberg. Fabscalar: Composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template. *SIGARCH Comput. Archit. News*, 39(3):11–22, June 2011.

[11] M. Ebrahimi, Z. Ghaderi, E. Bozorgzadeh, and Z. Navabi. Path selection and sensor insertion flow for age monitoring in fpgas. In *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 792–797, March 2016.

[12] F. Firouzi, S. Kiamehr, and M. B. Tahoori. Nbti mitigation by optimized nop assignment and insertion. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 218–223, March 2012.

[13] F. Firouzi, S. Kiamehr, and M. B. Tahoori. Power-aware minimum nbti vector selection using a linear programming approach. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):100–110, Jan 2013.

[14] E. Gunadi, A. A. Sinkar, N. S. Kim, and M. H. Lipasti. Combating aging with the colt duty cycle equalizer. In *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 103–114, Dec 2010.

[15] W. M. Johnson. Super-scalar processor design. Technical Report No. CSL-TR-89-383, Stanford University, USA, 1990.

[16] H. Kim, S. B. K. Boga, A. Vitkovskiy, S. Hadjitheophanous, P. V. Gratz, V. Soteriou, and M. K. Michael. Use it or lose it: Proactive, deterministic

longevity in future chip multiprocessors. *ACM Trans. Des. Autom. Electron. Syst.*, 20(4):65:1–65:26, September 2015.

[17] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. Nbti-aware synthesis of digital circuits. In *2007 44th ACM/IEEE Design Automation Conference*, pages 370–375, June 2007.

[18] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis. Power management of datacenter workloads using per-core power gating. *IEEE Computer Architecture Letters*, 8(2):48–51, Feb 2009.

[19] Y. Lu, L. Shang, H. Zhou, H. Zhu, F. Yang, and X. Zeng. Statistical reliability analysis under process variation and aging effects. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 514–519, July 2009.

[20] S. Nassif, K. Bernstein, D. J. Frank, A. Gattiker, W. Haensch, B. L. Ji, E. Nowak, D. Pearson, and N. J. Rohrer. High performance cmos variability in the 65nm regime and beyond. In *2007 IEEE International Electron Devices Meeting*, pages 569–571, Dec 2007.

[21] S. N. Neophytou and M. K. Michael. Test set generation with a large number of unspecified bits using static and dynamic techniques. *IEEE Transactions on Computers*, 59(3):301–316, March 2010.

[22] T. Sakurai and A. R. Newton. Alpha-power law mosfet model and its applications to cmos inverter delay and other formulas. *IEEE Journal of Solid-State Circuits*, 25(2):584–594, Apr 1990.

[23] D. K. Schroder and J. A. Babcock. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of Applied Physics*, 94(1), 2003.

[24] J. P. Shen and M. H. Lipasti. *Modern Processor Design: Fundamentals of Superscalar Processors*, volume 17. Waveland Press Inc., Illinois, USA, 1st edition, 1999.

[25] R. Ubar, F. Vargas, M. Jenihhin, J. Raik, S. Kostin, and L. B. Poehls. Identifying nbti-critical paths in nanoscale logic. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 136–141, Sept 2013.

[26] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao. The impact of nbti effect on combinational circuit: Modeling, simulation, and analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(2):173–183, Feb 2010.

[27] Y. Wang, X. Chen, W. Wang, V. Balakrishnan, Y. Cao, Y. Xie, and H. Yang. On the efficacy of input vector control to mitigate nbti effects and leakage power. In *2009 10th International Symposium on Quality Electronic Design*, pages 19–26, March 2009.

[28] Y. Wang, S. Cotofana, and L. Fang. A unified aging model of nbti and hci degradation towards lifetime reliability management for nanoscale mosfet circuits. In *2011 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 175–180, June 2011.

[29] Y. Wang, H. Luo, K. He, R. Luo, H. Yang, and Y. Xie. Temperature-aware nbti modeling and the impact of standby leakage reduction techniques on circuit performance degradation. *IEEE Transactions on Dependable and Secure Computing*, 8(5):756–769, Sept 2011.