IMPROVING THE RELIABILITY OF MICROPROCESSORS UNDER BTI AND TDDB DEGRADATIONS

by

Lin Li

BS, Beihang University, 2005

MS, Beihang University, 2007

Submitted to the Graduate Faculty of

the Swanson School of Engineering in partial fulfillment

of the requirements for the degree of

PhD in Computer Engineering

University of Pittsburgh

2014

UNIVERSITY OF PITTSBURGH

SWANSON SCHOOL OF ENGINEERING

This dissertation was presented

by

Lin Li

It was defended on

October 18, 2013

and approved by

Jun Yang, Ph. D., Associate Professor, Department of Electrical and Computer Engineering

Youtao Zhang, Ph. D., Associate Professor, Department of Computer Science

Steven P. Levitan, Ph. D., Professor, Department of Electrical and Computer Engineering

Sangyeun Cho, Ph. D., Associate Professor, Department of Computer Science

Jingtao Wang, Ph. D., Assistant Professor, Department of Computer Science

Dissertation Director: Jun Yang, Ph. D., Associate Professor, Department of Electrical and Computer Engineering

IMPROVING THE RELIABILITY OF MICROPROCESSORS UNDER BTI AND TDDB DEGRADATIONS

Lin Li, Ph.D.

University of Pittsburgh, 2014

Reliability is a fundamental challenge for current and future microprocessors with advanced nanoscale technologies. With smaller gates, thinner dielectric and higher temperature microprocessors are vulnerable under aging mechanisms such as Bias Temperature Instability (BTI) and Temperature Dependent Dielectric Breakdown (TDDB). Under continuous stress both parametric and functional errors occur, resulting compromised microprocessor lifetime. In this thesis, based on the thorough study on BTI and TDDB mechanisms, solutions are proposed to mitigating the aging processes on memory based and random logic structures in modern out-of-order microprocessors.

A large area of processor core is occupied by memory based structure that is vulnerable to BTI induced errors. The problem is exacerbated when PBTI degradation in NMOS is as severe as NBTI in PMOS in high-κ metal gate technology. Hence a novel design is proposed to recover 4 internal gates within a SRAM cell simultaneously to mitigate both NBTI and PBTI effects. This technique is applied to both the L2 cache banks and the busy function units with storage cells in out-of-order pipeline in two different ways. For the L2 cache banks, redundant cache bank is added exclusively for proactive recovery rotation. For the critical and busy function units in out-of-order pipelines, idle cycles are exploited at per-buffer-entry level.

Different from memory based structures, combinational logic structures such as function units in execution stage can not use low overhead redundancy to tolerate errors due to their irregular structure. A design framework that aims to improve the reliability of the vulnerable

functional units of a processor core is designed and implemented. The approach is designing a generic function unit (GFU) that can be reconfigured to replace a particular functional unit (FU) while it is being recovered for improved lifetime. Although flexible, the GFU is slower than the original target FUs. So GFU is carefully designed so as to minimize the performance loss when it is in-use. More schemes are also designed to avoid using the GFU on performance critical paths of a program execution.

Then finally the TDDB reliability issues are analyzed and bit flipping technique is designed in addition to voltage scaling to improve TDDB reliability in memory based and combinational logic structures, leveraging TDDB's dependence on bit flipping frequency. The update counts in multiple units include matrix scheduler, caches and TLBs indicate significant potential of utilizing bit flipping circuit to mitigate TDDB stress. Although applying bit flipping technique on entry update improves the reliability under TDDB stress in most units, ITLB is the only unit that lacks natural frequent update activity. The local write circuit is a effective and light weight design to proactively boost the bit flipping frequency.

TABLE OF CONTENTS

ACKNOWLEDGEMENT xiii

1.0 INTRODUCTION 1

1.1 Thesis Topic 1

1.2 Microprocessor Reliability Research Overview 2

1.2.1 Memory based Structure Reliability Improvement 3

1.2.2 Logic Structure Reliability Improvement 3

1.2.3 Aging Induced Reliability Issues 5

1.2.4 Reliability Design Challenges for Future Technology 6

1.3 Reliability Design in State-of-Art Microprocessor 6

1.4 Chapter Overview 7

2.0 DEGRADATION MECHANISMS 10

2.1 Bias Temperature Instability: BTI 10

2.1.1 NBTI and PBTI 10

2.1.2 BTI Recovery Modes 12

2.1.3 BTI Modeling 14

2.2 Time Dependent Dielectric Breakdown: TDDB 15

2.2.1 Frequency Dependent Stress 16

2.2.2 TDDB Modeling 16

2.3 Summary 17

3.0 PROACTIVE RECOVERY AND *4PR* FOR BTI IN SRAM CELLS . 18

3.1 Effect of both NBTI and PBTI 18

3.2 *4PR* Recovery Circuit 22

3.2.1 Existing approach . . . 22
3.2.2 4T proactive recovery for SRAM cells . . . 23
3.2.3 Further discussions . . . 26
3.3 Summary . . . 29
4.0 *4PR* ON L2 CACHE AND OUT-OF-ORDER ENTRIES . . . 30
4.1 Using a spare bank for *4PR* in L2 Cache . . . 30
4.2 *4PR* opportunity for Busy FUs . . . 31
4.2.1 FU Activities . . . 31
4.2.2 Idle Cycle Analysis . . . 33
4.2.3 Recovery Opportunities with Latency Overhead . . . 34
4.2.4 Hardware Design for *4PR* in Busy FUs . . . 36
4.3 Overhead Analysis . . . 39
4.3.1 Recovery Control Logic . . . 39
4.3.2 L2 Cache Recovery Logic . . . 40
4.4 Simulation and Results . . . 41
4.4.1 Simulation Setup . . . 41
4.4.2 Reliability model . . . 42
4.4.3 V_{th} shift comparison . . . 43
4.4.4 Cell failure probability analysis . . . 45
4.4.5 MTTF improvement . . . 47
4.4.6 IPC . . . 48
4.5 Summary . . . 49
5.0 GENERIC FUNCTION UNIT . . . 50
5.1 Challenges of Function Unit Reliability . . . 50
5.2 GFU: Generic Functional Unit . . . 51
5.2.1 The Overview of GFU Design . . . 51
5.2.2 GFU/m: A Macro-based GFU . . . 54
5.2.3 GFU/d: A Replication based GFU . . . 57
5.2.4 GFU/s: A STT-based Reconfigurable GFU . . . 58
5.2.5 Integrating GFU in the Execution Stage . . . 59

5.3 Reliability-aware Scheduling . . . 61
5.3.1 Step 1: Determine the FU to Recover . . . 61
5.3.2 Step 2: Minimize Performance Degradation . . . 62
5.3.2.1 Handling GFU/s based Recovery . . . 62
5.3.2.2 Handling GFU/m based Recovery . . . 63
5.3.3 Discussion on OS/Compiler reliability scheduler. . . . 71
5.4 Experimental Methodology . . . 71
5.4.1 Architecture Modeling . . . 71
5.4.2 GFU/m Circuit Modeling . . . 72
5.4.3 Reliability Modeling . . . 72
5.5 Experimental Results . . . 73
5.5.1 Hardware Overhead . . . 73
5.5.2 GFU-introduced Performance Degradation . . . 75
5.5.3 Instruction Steering . . . 79
5.5.4 Lifetime Improvement . . . 80
5.6 Summary . . . 83
5.7 Related Work . . . 83
5.7.1 Processor Core Logic Reliability . . . 83
5.7.2 Reconfigurable Function Unit . . . 85
6.0 BIT FLIPPING IN TDDB RELIABILITY IMPROVEMENT . . . 87
6.1 Matrix-based Scheduler . . . 88
6.2 Bit flipping Circuit . . . 91
6.3 Bit Flipping Frequency in Multiple Units . . . 92
6.3.1 Matrix Scheduler . . . 93
6.3.2 Cache and TLB . . . 93
6.4 Local Write Circuit for ITLB . . . 94
6.5 Results . . . 96
6.5.1 Analysis of Local Write Circuit . . . 96
6.5.2 LWC Overhead Evaluation . . . 97
6.5.3 MTTF improvement for Scheduler and L2 Cache . . . 99

6.6 Summary 99
7.0 CONCLUSION 101
BIBLIOGRAPHY 104

LIST OF TABLES

1 Overview of Solutions for BTI and TDDB 7
2 Recovery Comparison 28
3 Energy for test 128_bit×256 SRAM bank. 40
4 Processor configurations. 42
5 Detailed FUs in Super-FU block in Fig. 19 from [26] 52
6 Control Signals and Added Overhead for Reconfiguring GFU/m. 56
7 The extra latency of GFU/m. 57
8 Simulation configuration. 72
9 GFU/m Hardware Area/Delay Evaluation 73
10 Evaluation of DIV, CRC and AES 75

LIST OF FIGURES

1 Three NBTI modes . . . 11
2 Vth shift rise/drop in stress/recovery . . . 12
3 TDDB failure distribution . . . 16
4 Standard 6T SRAM Cell . . . 19
5 Cell stability . . . 20
6 Proactive recovery circuits . . . 22
7 Mode control design . . . 24
8 Transition between PR and Normal mode . . . 25
9 BTI proactive recovery strength . . . 27
10 Idle time percentages for SPEC2000 INT and FP benchmarks . . . 33
11 Idle time distribution . . . 34
12 Idle time distribution . . . 35
13 Per-entry based recovery control logic. . . . 37
14 V_{th} shift for varying signal probability and inactive ratio . . . 43
15 V_{th} degradation for L2 and FU . . . 44
16 Failure probability analysis. . . . 46
17 MTTF Improvement . . . 47
18 Impact of proactive recovery on IPC. . . . 49
19 Intel Nehalem-like core execution units from [11] . . . 52
20 FU usage examples in 3 groups. . . . 53
21 GFU/m Structure . . . 55
22 Integrating GFU in the execution stage . . . 60

23 A simple instruction dependency graph 64
24 Issue Slack 65
25 Commit Slack 66
26 Comparisons of Slowdown Cycles and IS1 CS1 Instructions 67
27 CS1/CS2+ instruction count vs distance to ROB tail (INT bmk) in ALU2 . . 68
28 CS1/CS2+ instruction count vs distance to ROB tail (FP bmk) in ALU2 . . 69
29 Normalized IPC when FP Add/Mul/Div gets replaced respectively. 76
30 Normalized IPC when ALU2/0 gets replaced respectively. 77
31 2 ALUs vs 3 ALUs 79
32 Performance overhead reduction of using instruction steering 80
33 MTTF improvement 81
34 Wakeup Matrix in [63] 89
35 Wakeup Matrix Circuit in [63] 90
36 Bit Flipping Support in Wakeup Matrix Circuit 92
37 Signal Probability in Wakeup Matrix 93
38 Update Count for Multiple Structures 94
39 Local Write Circuit 95
40 Update Activity of ITLB with Local Write Circuit 96
41 ITLB MTTF with Local Write Circuit 97
42 Comparison of RFW and LW Power in ITLB 98
43 MTTF Improvement for Scheduler and L2 Cache 99

ACKNOWLEDGEMENT

I'm holding my breath writing these words since it's after years of dedicated work in computer architecture research. This thesis is not possible without generous support from my advisors, as well as professors in committee and others from the Department of Electrical and Computer Engineering. Doing research in Ph.D study is a long and lonely journey. It's long because one has to master both the breadth of literature and depth of certain field before truly understand the important problems and whether they are low hanging fruits (usually they are not since other smart and competitive researchers already took them). It's lonely because the further one search and ponder, he will find himself the only one crawling in tunnel with uncertainties. He has be creative to figure ways out so he is making contribution at the end of journey.

Thanks to all supports from my advisor Prof. Yang and co-advisor Prof. Zhang who relentlessly devote their time and effort in guiding me researching frontiers of computer architecture. Prof. Yang always trusts her students' potential and gives invaluable advice in all respects to encourage them to achieve fully, since she is a perfect role model of diligence and endless pursuit for perfection. In 1:1 and group meetings I learned how to think and work like a researcher. Prof. Zhang has endless sharp insights and rigorous investigations in discussions. Without their support this thesis is nowhere close to existence.

I owe my gratitude to professors in committee. Prof. Cho introduced the computer architecture foundation in my early Ph.D study and has always been very constructive and nice in all seminars I attend. Prof. Levitan set high standards and guides my work to be solid and at high level. Prof. Wang shared valuable perspective and suggested revision for easier access. Their advice and suggestions are essential to this thesis.

Also I learned a lot from Prof. Levitan, Prof. Jones, Prof. Mickle and Prof. Yang with which I spent many hours in teaching undergraduate students in labs and lecture classes. They showed the passion when sharing knowledge as well as the way of thinking to students. I must thank my colleagues and friends that I have been working with in my course of my Ph.D study, Xiuyi Zhou, Ping Zhou, Bo Zhao, Yi Xu, Lei Jiang, Vicky Li and Yu Du. I enjoyed the time spent together discussing research as well as relaxing. I also get a lot of help from them. With them I'm not lonely any more.

At last and most importantly, many thanks to my wife, Xi Yang and our little son, Austin. You are always supportive in everyday life for my decision to pursue the Ph.D. I'm not alone, and this thesis is for also you.

1.0 INTRODUCTION

1.1 THESIS TOPIC

As technology scales, computer hardware is increasingly susceptible to both soft and hard errors. Soft errors are transient malfunctions (bit flip in memory and timing error in execution path) due to external random disturbance such as high energy particle strike, temperature and voltage fluctuations. Soft errors are more likely to happen in smaller gates: it takes less energy to flip one bit in smaller gates with lower voltage and there are even more multi-bit soft errors [68], as observed in both memory and execution units. Hard errors are hardware defects induced both in manufacture and in-field aging. Smaller gates with thinner oxide are stressed under stronger electrical field and higher power density. Together with temperature, voltage and process variations, both yield loss and aging problems become increasingly severe. Consequently, reliability caused by both soft and hard errors becomes more important than ever.

The soft error detection and correction have been studied extensively in past. The random nature of soft errors makes them correctable by re-execution from checkpoints once errors are detected [12] in execution path, or by ECC in memory. Although expensive, checkpointing and re-execution circuits are used in high-end mission critical microprocessors. The techniques for soft errors are well established and not the topic of this thesis.

The hard error detection and correction are becoming more challenging in scaling technology. Hard errors include both function faults and parametric faults, and both may occur in manufacture and in-field use. Defects in manufacture are inevitable and the root cause of yield loss. Besides function faults, process variation in manufacture also incurs parametric faults: although circuits are functional, parameters such as performance and power fail to

meet target. Both functional and parametric faults in manufacture reduce effective yield and increase chip cost.

At runtime, circuits degrade under stress of multiple aging mechanisms on gates during in-field use. Bias Temperature Instability (BTI), Time Dependent Dielectric Breakdown (TDDB), Hot Carrier Injection (HCI), etc (discussed further in Section 2) are such mechanisms. Some mechanisms (TDDB) introduce permanent functional faults, while others (TDDB and BTI) exacerbate parametric faults, jeopardizing the lifetime of a processor. New generations of technology with stronger electric field due to thinner dioxide and higher operating temperature exacerbate the reliability across the entire processor including both memory and logic structures.

This thesis focuses on aging induced parametric hard errors (BTI) and functional hard errors (TDDB and BTI). Soft errors and manufacturing time hard errors are not covered. Based on the unique characteristics of BTI and TDDB, circuit and architectural level techniques are designed to alleviate circuit aging process and improve microprocessor lifetime. A brief overview of current status in academia and industry is discussed to map the thesis contribution in general picture of microprocessor reliability.

1.2 MICROPROCESSOR RELIABILITY RESEARCH OVERVIEW

The microprocessor reliability problem has attracted extensive efforts in past decades for tackling soft and hard errors. Different designs are proposed in memory based and random combinational units due to different structure and complexity by incorporating redundancy. Memory based structures are generally regular and it is straight forward to apply minor redundancy to improve reliability. Random combinational structures such as function units in execution stage are irregular and low overhead redundancy is generally not practical or available. This section briefly discusses existing methods that mitigate aging induced errors in memory based and random combinational circuits.

1.2.1 Memory based Structure Reliability Improvement

Memory based structures such as on-chip caches, branch predictors, register files, TLBs are critical since they store important state information. They occupy large chip area (over 50%) and are more vulnerable to soft/hard errors. Due to the regular and repetitive structure in data array, memory units are typically protected by Error Correction Code (ECC) and simple redundancy (spare bits, lines) from hard errors (both induced in manufacture and in-field again) and soft errors, at minor to moderate overhead in chip area.

Error correction code (ECC) is widely used to detect and correct soft/hard errors in memory structures. Single Error Correction Double Error Detection (SECDED) Hamming code is light weight and effective to detect at most 2 errors and correct 1 error. The SECDED encoding and decoding add extra cycles in data access, and were traditionally used in lower memory hierarchy such as L2/L3 and main memory where extra latency is amortized in overall longer access latency than L1 cache.

Recently, many variants of ECC are proposed in different scenarios. 2-D ECC [34] uses ECC in columns as well as rows to tolerate more bits in one row or column. Multi-bit ECC [76] use stronger BCH code such that up to 5 bit errors are correctable in eDRAM with lower refresh frequency. [84] claims error detection is critical and more frequent but error correction is less likely and proposes splitting error detection on chip and error correction off chip.

Besides ECC, another approach to improve memory reliability is adding light weight redundancy such as extra bits, columns, and rows as backup or spare. The spare components are reconfigured to replace defective ones in test stage to improve yield. [77] proposes one method to combine multiple defective lines to form one functional logic line. All above techniques take advantage of regular memory structure and lightweight error detection and correction to tolerate errors with minor overhead.

1.2.2 Logic Structure Reliability Improvement

Contrary to the regular memory based structure with mild and effective redundancy, irregular, non-memory core logic structures impose obstacles to detect and correct errors. They

cannot be protected using ECC, as errors manifest as faulty execution rather than faulty stored bits. Tremendous efforts for improving core logic reliability can be coarsely grouped to 2 categories: handling soft errors and handling hard errors.

Soft errors in core logic are tolerable by check-pointing and re-execution that is first proposed in Razor [12]. Shadow latch is used to detect timing errors in execution path, and the stage with error is re-executed from checkpoints after the pipeline is stalled and flushed. Although the recovery is expensive in terms of performance due to the pipeline flush and check-pointing, as long as timing errors are rare, the performance degradation is tolerable while the performance improvement is significant by eating-up conservative guard-bands and aggressively running microprocessor faster than specification with increasing minor risk of errors. Hence, Razor uses minor time redundancy to improve reliability and it usually works well for soft errors which occur randomly.

However, for hard errors, re-execution does not work since the error tends to happen again on the faulty hardware and even infinite re-execution could not make it correct. For timing related parametric errors, reducing the frequency is a remedy by trading off performance. While for functional hard errors, redundancy of different granularity (core, function unit) is necessary to improve reliability.

The present multi-core processors provide natural redundancy at the core level. A core can be disabled if errors are found post fabrication, resulting better yield than abandoning the entire chip. Considering throwing away faulty cores is too expensive, [55] and [3] try to salvage and use faulty cores: [55] observes some faulty function units are not frequently used and it's safe to run applications that do not utilize the faulty FU on the core; [3] observes a faulty core is likely to be an animator core in sufficient long period and provides useful hints to accelerate the execution of normal cores.

In a similar vein, a coarse-grained replication using Triple Module Redundancy (TMR) is adopted for highly reliable and mission critical systems [2, 6]. Although core-level or coarse-grained redundancy achieves high fault coverage, they are often over-designed and unnecessary, as a fault may occur in only one functional unit (FU) and it is unlikely that all FUs are faulty. Replicating a large area of logic is hence not cost-effective. However, fine-grained redundancy at the FU level is also challenging as it is difficult to determine which

FU is most vulnerable at design time, and which FU is stressed most online. Protecting irregular logic structure with redundant is not trivial task in terms of complexity and cost.

1.2.3 Aging Induced Reliability Issues

Previous study on microprocessor reliability focus on soft errors and manufacture time hard errors. Recently with technology scaling aging mechanisms receive a lot of attention and become emerging sources for reliability problems. Multiple aging mechanisms discussed in Section 2 stress circuits and finally cause parametric and function errors and compromised lifetime.

Although existing methods in above discussions could be used to tolerate aging induced hard errors as well as other errors, they are not quite effective in extending lifetime if only used after hard errors occur. Many research works take a promising path that proactively mitigates the degradation on circuits to slow down aging mechanisms, which is also the key topic of this thesis.

Since aging mechanisms (BTI and TDDB) are highly dependent on voltage and temperature (more details in Chapter 2), circuit level voltage tuning and architectural level scheduling are straight forward and effective approaches. [70] tried to hide and slow down aging by 1) steering hot jobs on faster cores and cold jobs on slower cores; 2) changing voltage by adaptive scaling voltage (ASV) and threshold voltage by adaptive body bias (ABB). [32] uses finnier granularity dynamic voltage scaling to slow down NBTI aging in core logic.

Besides circuit level methods, architectural level methods for mitigating NBTI are proposed since threshold voltage degradation caused by NBTI depends on signal probability: alternating "0" and "1" on PMOS is stressed less than always "0" on PMOS. [1] utilizes idle cycles to recover NBTI induced degradations by balancing signal probabilities on gates, which is be quite effective. [16] studies NBTI stress in context of process variation (PV). Due to PV, some FUs are inherently weaker, slower and vulnerable to NBTI stress. By using the vulnerable FU less and the robust FU more, the vulnerability difference initially introduced by PV is evened out in field use and the lifetime is greatly improved.

1.2.4 Reliability Design Challenges for Future Technology

Although this thesis focuses on current 28nm technology (High-k and bulk), the 20nm SOC and 14nm FinFET are coming in following years. [35] discusses the design challenges due to reliability on VMIN/VMAX, wires and CMOS devices. BTI is still a problem and gets worse in limiting the VMIN. Also the TDDB is worse since the new effect of TDDB that increases the sensitivity of 1x BEOL metal pitch to bias-temperature stress finally limits VMAX. For wires the electro-migration (EM) is also a worse problem with smaller interconnect cross-sections and higher current. Although the next generation fully-depleted FinFET devices improve performance at lower voltages, the FinFET device quantization poses challenges for 6T embedded SRAM memory in that it's more difficult to balance both write and read margins. With technology advancing, the reliability keeps challenging for processor designers.

1.3 RELIABILITY DESIGN IN STATE-OF-ART MICROPROCESSOR

Not only in academia, many microprocessor manufactures also become aware of the increasing challenging reliability issues, and reliability features are added in core logic and memory although more complexity and cost are incurred. In IBM zEnterprise CPU [75], extensive parity and residue error checking are employed against soft errors, which adds roughly 20 to 25% to core logic area. In Intel Itanium Poulson [60], residue protection and radiation hardened sequential latches are implemented on key logic and data-path. Despite large area is devoted to recovery circuits, it is necessary for these high-end processors to guarantee top level reliability.

Stronger ECCs are also implemented in high-end microprocessors. In Intel Westmere CPU [64], heavy portion of area is dedicated to ECC. The L3 cache data arrays are strengthened with Double Error Correcting, Triple Error Detecting codes (DECTED), and tag arrays are protected using Single Error Correcting, Double Error Detecting (SECDED) codes. In Intel Itanium Poulson microprocessor, L3 cache are protected by DECTED, MLI/MLD data are protected by SECDED code. Even the register file have extensive parity and interleav-

Table 1: Overview of Solutions for BTI and TDDB

Degradation	Memory based Structure	Combinational Logic Structure
BTI	*4PR* and application	GFU
TDDB	Voltage⇓, Bit Flip Frequency⇑	Voltage⇓, GFU

ing parity. Niagara T4 use ECC on register files, coherency states and all cache hierarchies. Although stronger ECCs incur higher complexity, area and cost, it is worthy to implement to overcome the reliability problems.

State-of-art high-end mission critical processors require reliability aware design and recovery, while it's not common practice yet for commercial processors. With the technology scaling and increasingly worsen reliability issues, a lightweight and effective reliability aware design technique is required for commercial processors.

1.4 CHAPTER OVERVIEW

The literature study shows extensive efforts have been invested to mitigating aging in both memory based and random combinational structures. However, most existing techniques only target NBTI induced degradation and only natural recovery (to be discussed shortly in Chapter 2) is exploited to enhance reliability. The recently surfaced and equally important PBTI becomes another important reliability issue and should be addressed properly. Moreover the faster, more effective proactive BTI recovery is left unnoticed and hardly exploited. Besides BTI, another important degradation mechanism, TDDB, although a hot topic in device community, hasn't received enough attention in circuit and architecture community so far. This thesis targets in solving the emerging aging induced reliability issues in both memory based and random combinational structures in modern complex out-of-order cores.

Table 1 illustrates the overview of solutions for BTI and TDDB degradations on memory based and combinational logic structures. For the BTI stress on memory based structure,

4PR, an innovative BTI proactive recovery mode for SRAM cell is implemented in Chapter 3. And the application of *4PR* in cache banks and storage entries of function units in out-of-order core is elaborated in Chapter 4. For combinational logic units, GFU is carefully designed and implemented to replace the vulnerable logic unit under recovery with minimal overhead and it works for both BTI and TDDB stress. The bit flipping technique is designed to mitigate the TDDB stress on memory based structures.

Chapter 2 studies the characteristics of two dominating aging mechanisms, BTI and TDDB. A favorable feature of BTI (both PBTI and NBTI) is its recovery mode, i.e, when the gate is off its shifted threshold voltage induced by BTI stress is partially recovered which we denote as natural recovery mode. A faster and more effective recovery mode (proactive recovery) is achieved when reverse bias is applied based on which Chapter 3 further develops *4PR* to be used in Chapter 4. For TDDB, the bit flipping frequency dependence makes the reliability improvement possible: under faster bit switching the TDDB stress is less. This feature provides the clue to tackle TDDB problem in memory based structure in Chapter 6. When such features are not practical to use, Chapter 5's GFU framework resort to shutting down the FU for natural recovery or stopping stress.

In Chapter 3, an innovative proactive recovery mode, *4PR*, is designed. The *4PR* is motivated by the fact that both NBTI and PBTI are important in high-k metal gate technology. The effect of both NBTI and PBTI on read flip failure, write failure and access time failure are discussed. Results show PBTI dominates the most likely access time failure and a *4PR* address the requirement by proactively recovering all 4 gates in a 6T SRAM cell. HSpice simulation shows *4PR* is an effective and low overhead circuit technique to improve BTI reliability.

In Chapter 4, *4PR* mode is applied in cache and storage cells in Out-of-Order pipeline. A spare bank is added in L2 cache to replace the bank under *4PR*. However, applying *4PR* in storage cells in pipeline is hard since re-routing the busy function units as register files and reorder buffers incurs large performance overhead. Inspired by [1], per-entry level idle cycles are exploited to apply *4PR*. Results show there are abundant idle cycles, and combining effective *4PR*, significant reliability improvement is achieved.

In Chapter 5, Generic Function Unit (GFU) is designed to utilize natural recovery and stopping stress to improve the reliability of function units in execution stage. *4PR* can not be easily used for function units due to the irregular structure of combinational logic. Instead an extra GFU is added to provide fine-grain function unit level redundancy to facilitate using natural recovery for BTI and stopping stress for TDDB. GFU has high fault coverage due to its flexibility: it can be configured as most FUs. Depending on different usage FUs are partitioned into 3 groups and GFU is accordingly designed to cover most FUs in 3 groups. Also GFU has low area overhead since it only targets 1 or 2 FUs at a time instead of using spare copies for all FUs which incurs 100% area overhead. An important design issue is how to reduce the performance overhead when the generally slower GFU is introduced. Evaluation shows performance overhead is small and further reduced by steering less critical instructions to GFU. The reliability lifetime improvement is significant for the combinational logic units.

In Chapter 6, reliability loss due to TDDB stress in processor core is studied with focus on memory based structures. Although voltage scaling can be applied on many less busy units to mitigate TDDB stress, some structures on critical path that is used in every cycle can not use voltage scaling. Scheduler, ITLB and L1 cache are identified to be such structures that voltage scaling is not feasible due to potentially significant performance overhead. Based on the frequency dependent TDDB stress, the bit flipping circuit is implemented to increase the bit flipping frequency as an essential alternative to mitigate TDDB stress. Results show in many units the entry update activity is sufficiently frequent to utilize the bit flipping circuit. However, one exception is ITLB in which the natural bit flipping frequency is extremely slow. Then the local write circuit is designed to enable proactive bit flipping, although the area overhead is not non-negligible. Evaluations show promising lifetime improvement in scheduler, cache and ITLB.

2.0 DEGRADATION MECHANISMS

In scaled technologies, although transistors are smaller, faster and consume less power, they tend to suffer intense and severe stress due to thinner gate oxide, stronger electric field, and higher temperature. Bias Temperature Instability (BTI) and Time Dependent Dielectric Breakdown (TDDB) are two such degradation mechanisms. BTI stress increases threshold voltage and hence propagation delay is larger and memory cell is less stable. TDDB stress gradually increases leakage current and finally gate oxide breaks down such that gates fails to control channels conductivity. In this chapter we discuss unique characteristics of BTI and TDDB that are to be used in circuit or micro-architectural level techniques in following chapters to slow down aging and improve reliability. Although there are other degradation mechanisms such as Hot Carrier Injection (HCI, not as severe as BTI [49]) and Fowler-Nordheim tunneling (FN, stress floating gate in FLASH), due to less severity, they are not discussed further in the thesis.

2.1 BIAS TEMPERATURE INSTABILITY: BTI

2.1.1 NBTI and PBTI

BTI includes NBTI (Negative BTI) on PMOS transistors and PBTI (Positive BTI) on NMOS transistors. For traditional SiO2 dielectric technology, NBTI has been identified as the dominant stress mechanism while PBTI is negligible. Efforts have been taken to study NBTI intensively. These include NBTI modeling [72], studying its impact on circuits [30, 37], and designing mitigation methods [37] for PMOS gates.

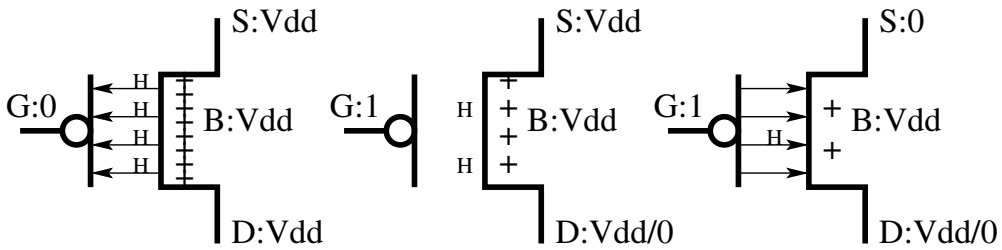


Figure 1: Three NBTI modes: Stress (left), Natural Recovery (middle), Proactive Recovery (right). PBTI has similar 3 modes. Proactive recovery is both faster and stronger than Natural Recovery

In sub-45nm technologies, high-κ metal-gate stack is widely used to gain high speed and low leakage. For example, Intel Westmere-EX Xeon and Itanium Poulson use 32nm high-k technology. Instead of using silicon dioxide (SiO_2) as the dielectric and poly-silicon gate, the new stack is composed of high-κ material dielectric and the metal gate, with a very thin layer of SiO_2 in between. The high-κ material has higher relative dielectric constant than that of SiO_2. The metal gates are used on top of the high-κ dielectric. With the new stack structure, scaled gates are less leaky while keeping high performance.

In high-κ metal gate stack, the NBTI occurs in PMOS transistors when the input of the gate is "0" at elevated temperature, as shown in the leftmost case in Fig. 1. Similarly, the PBTI occurs in NMOS transistors with the gate input "1". In short, the gates are stressed under BTI when ON, for both PMOS and NMOS. Under BTI stress (negative/positive gate-source bias i.e., $V_{gs} = -/+V_{dd}$ for PMOS/NMOS), a series of physical-chemical processes are triggered such that 1) positively charged traps are created in interface of SiO_2 and SI near the gate oxide [85] for NBTI; 2) negatively charged traps are created in the added high-κ dielectrics for PBTI. These traps raise the threshold voltage, which gradually slows down the gate transition speed and reduces the noise margin of the circuit. Eventually the circuit will fail as the transistors become too slow to meet the timing or stability requirement. Studies showed that BTI-induced failure becomes more severe under higher temperature and stronger electric field [67].

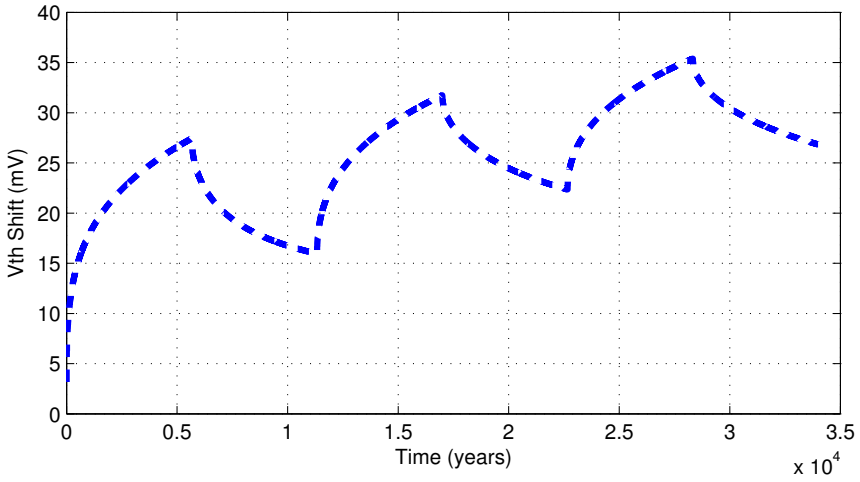


Figure 2: Vth shift rise/drop in stress/recovery in 3 iterations with 50% signal probability. This is also called AC stress, compared to continued stress called DC stress

However, the reliability of the high-κ metal gate stack is yet to be solved. For SiO_2 dielectric, NBTI in PMOS has been identified as the limiting reliability issue, while the effect of PBTI in NMOS is negligible. For high-κ metal-gate stack, recent measurements [86] on V_{th} shift show that NBTI in PMOS is as severe as that in SiO_2/poly-silicon. Moreover, the effect of PBTI in NMOS is now comparable to NBTI in PMOS. While there are works on PBTI measurements and modeling [86, 28], only limited attempts have been conducted to studying the impact of PBTI on circuits and its mitigation methods for high-κ material.

2.1.2 BTI Recovery Modes

Besides stress mode, one key characteristic of BTI is two recovery modes, which have been investigated to increase the lifetime of transistors. We use NBTI on PMOS as example. The first recovery mode, referred as *Natural Recovery*, is shown in the middle case of Fig. 1. It happens when the input of the gate is "1" with normal V_{dd}. This is the mild recovery used in [1]. When the input changes to "1", the negative bias stress is removed, the traps in the gate oxide near the interface are healed, and the NBTI degradation on the V_{th} of PMOS

can be recovered partially. Note that when the input switches between logic "0" and "1" on the gate, the PMOS transistor undergoes an alternation of stress and recovery, as shown in Fig. 2.

A stronger and faster recovery mode for PMOS transistors, referred as *Proactive Recovery*, is shown in the last case of Fig. 1. It uses positive voltage bias, i.e., $V_{gs} = V_{dd}$ [45] and forces the gate input to be "1" and S terminal to be "0". The MOS transistor cannot enter the strong recovery mode in normal operation since (1) the S terminal of the PMOS is connected to physical V_{dd}; (2) even if S terminal can be driven to ground, the gate voltage is discharged through the preceding PMOS and cannot be kept at "1". Therefore the cell circuit needs to be revised properly to enable proactive recovery.

The recovery mechanisms of NBTI used to mitigate the degradation are also observed for PBTI in high-κ dielectrics in experimental conditions. For example, V_{th} shift under PBTI is naturally healed after removing the stress and making V_{gate}, or $V_g = 0V$ [29, 28], which is same as "Natural Recovery" in NBTI. Stronger and accelerated recovery is achieved when an inverse bias (discharge bias in [28, 50]) is applied to V_g.

Thus, the impact of NBTI is determined by the operation of the circuits, and varies with different applications, or diffident phases in one application. An early analytical study [38] showed that the NBTI progress is determined by the signal probability of the input signal being "0", temperature and supply voltage, but independent of how frequently the transistor switches between "0" and "1". As a result, having sustained recovery span is as equally effective as having interrupted recovery periods as long as their accumulated recovery cycles are the same.

Both NBTI and PBTI are caused by high temperature, strong electric field, and most importantly, high stress duty cycles. Typical recovery schemes for NBTI exploit the opportunity to maximize the recovery cycles for PMOS gates [16]. However, in the presence of both NBTI and PBTI, the duty cycles for NMOS and PMOS are complementary: a gate input of "1" incurs stress on an NMOS and recovery on PMOS, while a gate input of "0" incurs recovery on NMOS and stress on PMOS. Therefore, maximizing (natural) recovery cycles for one type of gate could stress the other gate even more. For SRAM cells, [4, 5] showed that PBTI has a larger impact than NBTI on the degradation of a SRAM cell under the worst

case BTI stress. This has made previously designed NBTI-oriented mitigating techniques inadequate, and calls for the development of new schemes for high-κ based circuits.

2.1.3 BTI Modeling

Although detailed and accurate BTI modeling and search the root of BTI mechanism are continued efforts in device and reliability research community, circuit level modeling often use a simplified model, which just describes the relationship of resulting threshold voltage shift and parameters such as temperature, voltage and signal probability.

A BTI model [43] shows the relationship of time to failure due to BTI (TF_{BTI}) and parameters such as temperature and voltage in Equation 2.1([43]). A_{NBTI} is transistor's gate-oxide area, T is temperature, V_{gs} is gate-to-source voltage. Details of temperature functions $B_1(T)$ and other parameters are discussed in [43].

$$TF_{BTI} = A_{NBTI} V_{gs}^{-\frac{1}{\beta_1}} B_1(T) \tag{2.1}$$

Although reducing temperature and voltage [70] has been demonstrated to be effective to slow down BTI aging, our techniques focus on proactive recovery during idle cycles. Thus, we use a more accurate model proposed in [31], in which the V_{th} shift under AC NBTI stress is calculated by Eqn (2.2).

$$\Delta V_{t,AC} = K_{AC} \cdot t^n = \alpha(S, f) \cdot K_{DC} \cdot t^n \tag{2.2}$$

Here K_{DC} is the technology dependent constant. $\alpha(S)$ is the AC factor determined by the signal probability of the cell S (extracted from architectural simulations), independent of operating frequency f. $\alpha(S)$ is calculated using the multiple cycle stress and recovery model: the stress phase is modeled by the popular accepted power-law model with the exponent $n = 0.16$ for P/NMOS [28, 19]; the recovery phase is modeled according to [72]:

$$\text{Stress: } V_{th}(t) = \sqrt{K^2 (t - t_0)^{\frac{1}{2}} + V_{th}(0)^2} \tag{2.3}$$

$$\text{Recovery: } V_{th}(t) = V_{th}(0)(1 - \sqrt{\eta(t - t_0)/t}) \tag{2.4}$$

The $V_{th}(t)$ is V_{th} at time t when the stress or recover mode begins at t_0 with the $V_{th}(0)$. The parameters η for P/NBTI under natural and proactive recovery are individually calibrated to match the experiment data in [58]. This model would be used in evaluations in following chapters.

2.2 TIME DEPENDENT DIELECTRIC BREAKDOWN: TDDB

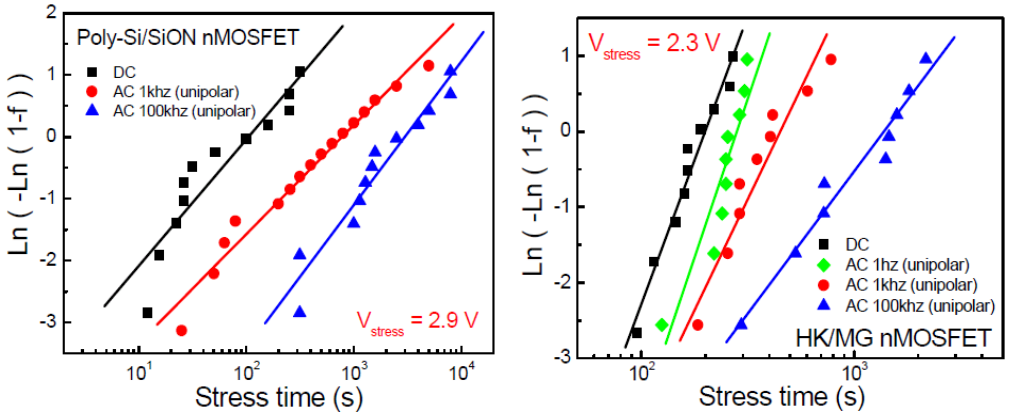
Gate-oxide breakdown is another important failure mechanism in CMOS technologies [43]. TDDB stress experiences 3 stages: soft break down (SBD), wear out (WO) and finally hard bread down (HBD). When gate is on, SBD happens when two-trap perlocation cluster is created due to tunneling current flow though gates. SBD could be detected as an abrupt current increase. Although gate leakage current due to SBD is negligible, it starts the WO process on percolating path. At operating conditions where the gate leakage current increase due to SBD is negligible, the WO phase becomes the dominant factor in degradation. At the end of WO, HDB happens. The gate-voltage controllability is lost which further results in excessive dissipated energy or even thermal runaway.

In high-κ metal gate technology, TDDB degradation is much worse than traditional poly-Si technology [61]. In high-κ technology, the runaway phase in WO is much faster than that in poly-Si technology, resulting greatly reduced reliability margin. Ultra-thin gates with high-κ dielectric, high temperature and high voltage accelerate the TDDB failure rate and hence TDDB is also a serious degradation mechanism.

Although TDDB is at least as important as BTI, it received much less attention than BTI. Unlike the interesting characteristics such as signal probability dependence and proactive recovery in BTI, people conceived that TDDB is only highly dependent on temperature and voltage, and circuit techniques to reduce temperature/voltage suffices are used to slow down aging. However, recent research shows an interesting characteristic of TDDB, Frequency Dependent Stress, which is to be exploited in this thesis to improve TDDB reliability.

2.2.1 Frequency Dependent Stress

Recent experiments [41] show TDDB degradation depends on bit flipping frequency. In Fig. 3(a) and Fig. 3(b), both poly-Si/SiON and HK/MG suffer less TDDB degradation under unipolar AC stress over DC stress in accelerated test with $2.9V$ and $2.3V$ stress. The degradation is further reduced when higher AC stress frequency used. For example in HK/MG, the lifetime improvement is 1.8x/3.5x when frequency increase from 0 to 1Khz and to 100KHz. According to [41], the frequency dependency is due to that trapped charges in HK can easily be detrapped once a relaxation bias is applied. This new interesting and favorable finding motivates circuit and micro-architectural level designs to slow down TDDB degradation.



(a) Poly-Si/SiON nMOSFET (b) HK/MG

Figure 3: TDDB failure distribution under DC and unipolar AC stress for poly-Si/SiON (a) and HK/MG (b) [41]. In both cases the unipolar AC conditions exhibit longer time to breakdown than DC conditions.Vertical axis is accumulative failure probability in Weibull distribution.

2.2.2 TDDB Modeling

Extensive researches show SBD is Weibull distributed and voltage acceleration accords to a power law. The time to failure due to TDDB (TF_{TDDB}) under different temperature, voltage gate area is calculated in Eqn 2.5. A_{TDDB} are transistor's gate-oxide area, T is

temperature, V_{gs} is gate-to-source voltage. Details of temperature functions $B_1(T)$, $B_2(T)$ and other parameters are discussed in [43].

$$TF_{TDDB} = A_{TDDB}(\frac{F}{A})^{-\frac{1}{\beta_2}} V_{gs}^{a+bT} B_2(T) \tag{2.5}$$

Eqn. 2.5 with the parameters in [43] shows the strong and sensitive dependence of TDDB lifetime on supply voltage: reducing $0.1V$ supply voltage improves the TDDB lifetime by 100x, as confirmed in [61]. Hence less frequently used or less critical function units such as last level cache would enjoy extended TDDB lifetime by just reducing supply voltage. However, for frequently used and busy function unit where reducing voltage incurs intolerable performance loss and temperature is usually high, the bit flipping technique could be used to improve TDDB lifetime.

However, the frequency dependency is not modeled in Eqn. 2.5 and no available such models are proposed so far. In evaluations in following chapters we treat Eqn. 2.5 as baseline lifetime estimations, and use calibrated parameters in Fig. 3 ([41]) to evaluate the benefit of increasing bit flipping frequency.

2.3 SUMMARY

This chapter discusses BTI and TDDB degradation mechanisms in traditional poly-Si and latest high-κ technology. BTI, TDDB that were tolerable in traditional technology become a more serious threat to device, circuit, chip and system reliability. Accurate models show that temperature, voltage, stress patterns play important role in degradation. Properly adjusting temperature, voltage, and stress patterns paves a way to alleviate degradation and improve lifetime of microprocessors. The BTI natural and proactive recovery modes are exploited in chapter 3 and 4 to improve the SRAM cell based structures, both in cache and in entries in pipeline such as register file, reservation stations and robs. Increasing bit flipping frequency is exploited in chapter 6 to improve scheduler and itlb reliability.

3.0 PROACTIVE RECOVERY AND *4PR* FOR BTI IN SRAM CELLS

In this chapter, a novel design is proposed that mitigates both NBTI and PBTI at the same time in all 4 transistors of SRAM cell in high-κ technology with which PBTI is as severe as NBTI. In Section 3.1, it's identified that for a SRAM cell, while NBTI on PMOS shows degradations to read static noise margin and read's statistical stability [31], PBTI on NMOS is more critical to the cell's access failure and/or read flip failure. Existing NBTI-only approaches [65] recover some but not all transistors. Then in Section 3.2, we devise a proactive recovery technique that recovers all four transistors in a cell simultaneously. A new recovery mode, *4PR*, is introduced in which the inverse bias is created for all four MOS gates and the voltage level of the internal gate inputs is approximately $0.5V_{dd}$. *4PR* helps to recover all four MOS gates proactively at the same time, with a medium strength but 100% effective recovery time during the recovery period.

3.1 EFFECT OF BOTH NBTI AND PBTI

The direct consequence of BTI stress is the MOS V_{th} increase due to the generated traps. The increased V_{th} then causes the performance and the reliability of both combinational circuits and SRAM cells to degrade.

For combinational circuits, the V_{th} increase in the PMOS/NMOS slows down the low-to-high/high-to-low transition due to a weaker drive strength. If V_{th} increases too high, the transition may fail to meet the timing requirement of the circuit. Previous study has shown that although the increased delay caused by NBTI is significant for a single gate, its impact on a combinational circuit is relatively low(3.3% in 3 years [30]). However, considering

both NBTI and PBTI, their impact is doubled and a re-evaluation is needed, which is not covered in this thesis. On the other hand, NBTI in combination with process variation can severely degrade the reliability of memory cells [30]. Such degradation can greatly reduce the statistical read stability and read static noise margin of the cell [31]. Therefore, we target the on-chip SRAM cells and mitigate both their NBTI and PBTI degradations in this work.

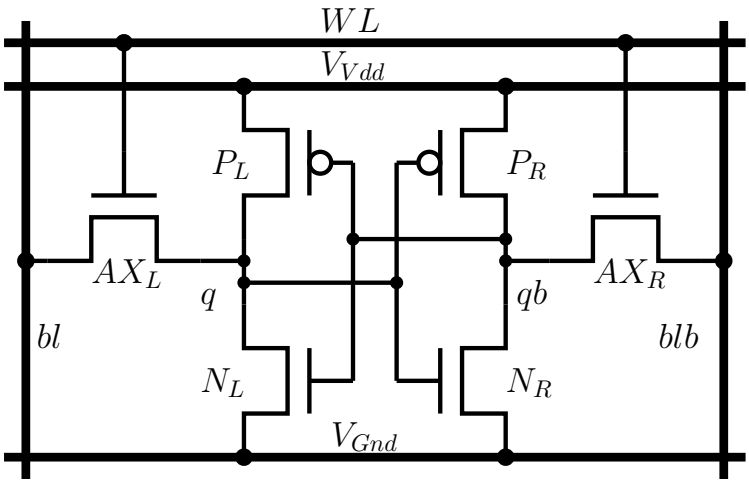
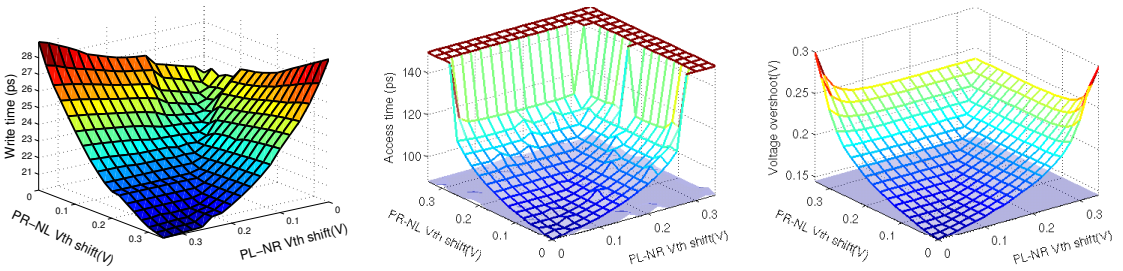


Figure 4: Standard 6T SRAM Cell

Fig. 3.1 shows a conventional 6T SRAM cell. When "1" is stored ($q = 1, qb = 0$), P_R and N_L are under natural recovery, and P_L and N_R are under stress. When "0" is stored, P_R and N_L are under stress, and P_L and N_R are under natural recovery. As we can see, the signal probability of the cell determines the V_{th} shift of the P_R-N_L pair and P_L-N_R pair. We ignore the stress on the access transistor due to its average low duty/busy ratio. We will assume a cell storing value 0 with $q = 0, qb = 1$ for the ease of discussion in the remainder of this section.

V_{th} variation on gates can lead to the following 3 failure mechanisms in a SRAM cell [51]: write failure, access failure, and read flip failure. The write failure happens when writing an inverse value to the SRAM cell cannot finish before the word-line closes and the transaction rewinds. The access failure happens when the bit-line does not discharge below the sense amplifier's trigger threshold such that it fails to sense the correct data. The read flip failure happens when an inverse value is written in the cell during a normal read operation, since



(a) Write time

(b) Access time

(c) Voltage overshoot of q in read

Figure 5: Cell stability under 1. NBTI only (surface plot) 2. Both NBTI and PBTI (mesh plot)

the voltage at q (storing a "0") is driven too high by the pre-charged bit-line and shoots over the trip point, initiating the flipping process. The SRAM cell fails if any of the above 3 failures occurs. We do not include the hold failure analysis since it is for special operations. Note that process variations (PV) during fabrication can also lead to V_{th} shifts which are additive to BTI degradation induced shifts. We will first discuss BTI effects without PV for clarity, and later present experimental evaluations with PV for completeness.

To evaluate failures caused by V_{th} shifts, we measured the write time, access time and voltage overshoot magnitude at q during a read (which can cause read flip failure) for an SRAM cell while V_{th} increases, shown in Figure 5. The dynamic simulation-based failure criteria is used instead of static criteria such as SNM and WSNM, since RSNM tends to overestimate the dynamic read failure and WSNM underestimates the dynamic write failure [33]. In simulating one cell, we used equivalent resistance and capacitance load extracted from a 128_bit×256_word SRAM bank, to obtain reasonably accurate timing and expedite the simulation. We will also compare our observations with [4, 5].

In Figure 5, the mesh plot shows results when only NBTI induced V_{th} shift is considered, and the surface plot shows the results when both NBTI and PBTI are considered. We do not put any timing constraints on read or write to show clearer trends with extreme V_{th} ranges ($0 \sim 300mV$) in the figure and no variations on access transistors) for a clear comparison.

Note that in reality, V_{th} is in a much smaller region, e.g. $0{\sim}100mV$, but its impact on those failures still complies with the observations we make.

Fig. 5(a) shows write time variations with V_{th} increases. The lower the better. The mesh and surface plots almost overlap since with or without PBTI, the write times only differ by 2ps at most, indicating that PBTI does not have significant impact on write time. In other words, the NBTI effect on PMOS dominates the write time. Also, the weakened PMOS due to NBTI further weakens the pull-up strength, which makes the pull-down relatively easier. This is because the PMOS in an SRAM cell is designed weaker than the NMOS, and is more sensitive to V_{th} shifts. As a result, the write time even improves when V_{th} shifts becomes larger. This is also observed in [5]: writability is improved under symmetric degradation; only in worst case write time degrades marginally with PBTI.

Fig.5(b) shows the access time changes. Similar to Fig. 5(a), the lower the value the better the reliability. The plot clearly shows that when both NBTI and PBTI are considered (the mesh plot), access time increases quickly. However, if only the NBTI stress is considered (the surface plot), the changes in access time is not noticeable. This is because the access time is mainly determined by the driving capability of the pull-down path of the cell which does not involve the two PMOS. Hence, the degradation in the PMOS has little effect on the access process. In [4], the read access time is more sensitive to PBTI which plays an more important role. As we can see, when considering the PBTI stress on NMOS gates, the access failure becomes a serious reliability problem, which was not surfaced when NBTI was considered only.

Fig.5(c) shows the voltage overshoot of q in a read operation with respect to V_{th}. The higher the worse. Again, we see that the mesh plot rises above the surface plot, meaning that the read flip failure is more likely to happen when PBTI is counted. Besides PBTI affecting read stability by raising the voltage overshoot, the NBTI also degrades the read stability by reducing the flip trip point. In [5], read SNM is more sensitive to PBTI compared to NBTI. Hence, PBTI worsens the read flip failure that is already threatened by NBTI.

To summarize, the NBTI plays an important role in write time, while the PBTI plays an important role in access time. The read stability is both affected by NBTI and PBTI. Among all the failures, the access time failure dominates other SRAM cell failures [51] (in

Monte-Carlo simulation). Without considering PBTI, the access reliability would be treated overly optimistically. Hence, considering both NBTI and PBTI are mandatory to evaluate the reliability of high-κ metal-gate stack technology.

3.2 *4PR* RECOVERY CIRCUIT

In this section, we elaborate the *4PR* design and compare it with pure natural recovery (a simple approach that simply turns off the circuit and let all MOS transistors recover) and the existing approach [65] that proactively recovers NBTI-introduced degradation.

3.2.1 Existing approach

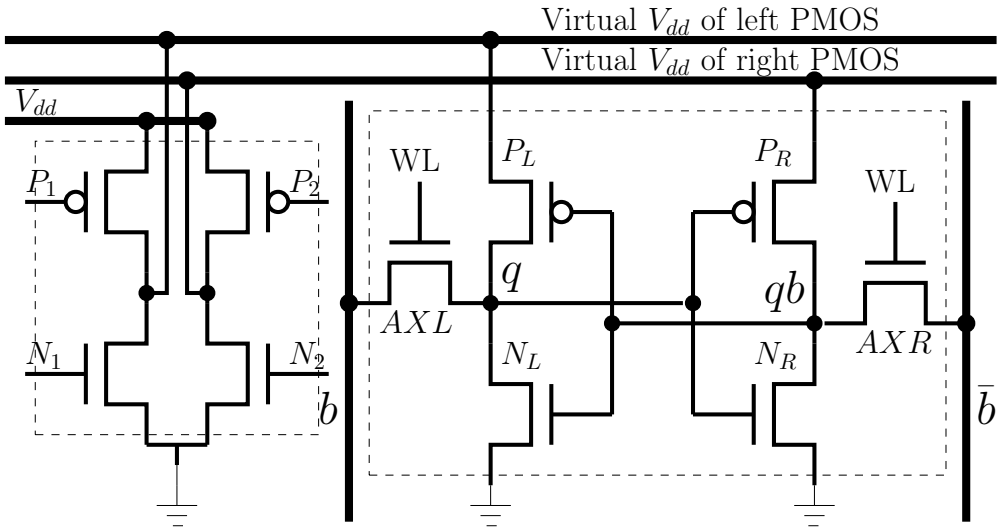


Figure 6: Proactive recovery circuits (left dash line box) for SRAM cells [65](right dash line box)

Under normal condition, a SRAM cell only stresses one PMOS-NMOS pair ($source = drain = 1$, $gate = 0$ for PMOS) while having the other pair under natural recovery ($source = gate = 1$ for PMOS). The proactive recovery technique proposed in [65] turns natural recovery into strong recovery ($source = 0$, $gate = 1$) to suppress PMOS V_{th} shift.

This is achieved by forcing the virtual power (VV_{dd}) to ground for the PMOS that is being recovered. Since the other PMOS is still under stress, the proactive recovery needs to be applied to two PMOS transistors alternatively, which requires two virtual power lines to separate the two sides.

To mitigate both NBTI and PBTI, a simple extension of the above scheme is to apply proactive recovery to one PMOS-NMOS pair, instead of just one PMOS. Take P_L-N_R pair as an example: besides driving the virtual power of P_L to ground, the virtual ground of N_R is also driven to V_{dd}. Two virtual ground lines are necessary to separate N_R and N_L. Hence, the gate is positively/negatively biased to the source for P_L/N_R, resulting in proactive recovery on P_L-N_R pair. We refer this technique as "Single Pair PR" ("SP PR" for short). Since PBTI stress on NMOS cannot be neglected when using high-κ technology, we will compare our scheme to the extended proactive recovery "SP PR" instead of the original one in [65] in the rest of the discussion.

"SP PR" has two limitations. First, the recovery mode recovers one PMOS-NMOS pair but stresses the other. Thus, its improvement over pure natural recovery is limited. Second, this approach connects sources of 4 MOS to 4 virtual source/ground wires, resulting in increased wiring overhead.

3.2.2 4T proactive recovery for SRAM cells

In this subsection, we discuss the design to recover all 4 transistors in one SRAM cell simultaneously (not counting the 2 access transistors), as opposed to only 1 or 2 transistors ([65] or "SP PR"). In a regular SRAM cell storing a "0", P_L and N_R are under natural recovery while P_R and N_L are under NBTI and PBTI stress respectively. Storing a "1" incurs the mirror effect of recovery and stress. By turning all four transistors into a stronger recovery mode, we tackle both NBTI and PBTI degradation in those 4 transistors, achieving much higher reliability improvement.

We refer to the new recovery mode as *4T-Proactive-Recovery (4PR) mode*. It pulls the source terminals of the 2 PMOS gates down to ground while pulling the ground terminals of the 2 NMOS gates up to V_{dd}. As shown in Fig. 7, the power and ground terminals of

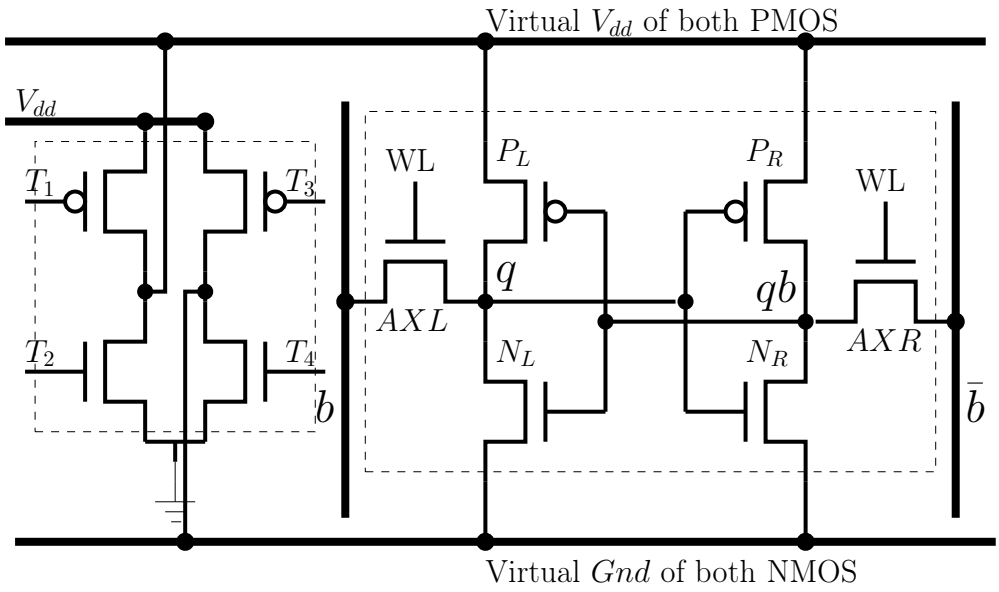


Figure 7: Mode control design: "Normal" when T1 and T4 are on; "4PR" when T2 and T3 are on.

all SRAM cells in a cache bank are connected to virtual power (VV_{dd}) and virtual ground (VG_{nd}). We add a mode control circuit including T_3-T_4, for VG_{nd}, and T_1-T_2, for VV_{dd}. We also add drivers to drive T_1-T_4. The area overhead of these drivers is small since only one such control circuit is needed per cache bank. Our evaluation also shows that the drivers and virtual power/ground wires have negligible impact on write time (0.56%), access time (0.33%) and read stability (0.36%) when V_{th} degrades from 0 to $100mV$. Here we did not consider PV of drivers since they are less sensitive to PV due to the large size.

By setting different inputs to T_1-T_4, the SRAM cell can switch between *Normal mode* and *4PR mode* as follows.

- *Normal mode*: T_1 and T_4 are on, T_2 and T_3 are off, virtual power/ground connects to physical power/ground.
- *4PR mode*: T_2 and T_3 are on, T_1 and T_4 are off, virtual power/ground connects to physical ground/power.

The *4PR* mode is intuitively errant since in SRAM cell the pull-up PMOS is connected to ground and pull-down NMOS is connected to power. Fig.8 (left figure) shows the transient analysis when entering from the *Normal* mode to the *4PR* mode. The *Normal* mode control signal "normal" is connected to T_1 and T_4, and *4PR* mode control signal "pr" is connected to T_2 and T_3. Initially, the SRAM cell is in *Normal* mode, storing a "1". When "normal" and "pr" commands flip, the cell enters the *4PR* mode. With falling "normal" signal, both VV_{dd} and q drop. With rising "pr" signal, both VG_{nd} and qb rise. When q falls and qb rises such that $q - qb < V_{th}$ is approached, P_L and N_R are off, and the cell enters the *4PR* mode. Note that N_L and P_R are always off during this process. After all 4 transistors are turned off, the q and qb slowly approach to approximate $0.5V_{dd}$, due to the sub-threshold leakage discharge of q by P_L, and charge of qb by N_R. Thus, in the *4PR* mode, all the 4 gates are off, and the internal q and qb reach about $0.5V_{dd}$. Note that we used 45nm high performance model [8] (lower V_{th} for MOS) in this study. If the low power model (higher V_{th} for MOS) is used, there is a voltage gap between q and qb when they stabilize.

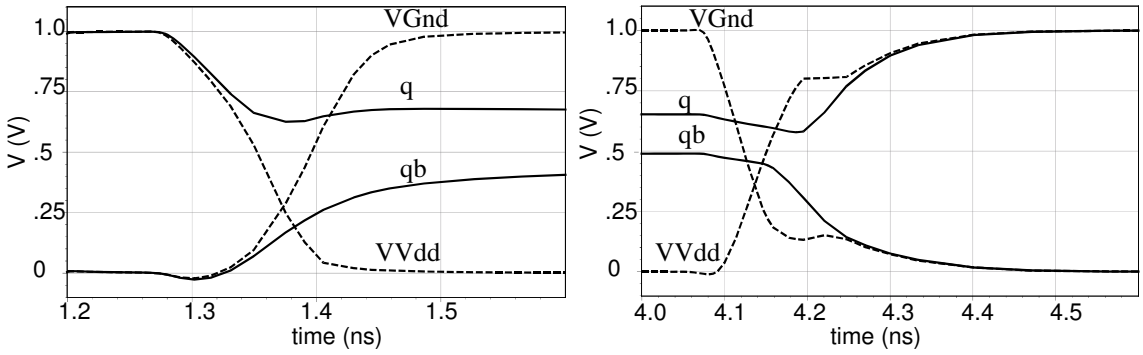


Figure 8: Entering (left) and exiting (right) 4PR mode from the Normal mode. The transitions are fast enough.

Note that we have been using "off" to indicate that the gates are not in the saturation mode. However, they are not entirely off. In fact, all 4 gates are in the *proactive recovery* mode. The "S" and "G" terminals of the two PMOS are 0 and $\sim 0.5V_{dd}$, forming a positive bias (inverse) of V_{gs}. The "S" and "G" terminals of the two NMOS are 1 and $\sim 0.5V_{dd}$, forming the negative bias (inverse) of V_{gs}. Positive V_{gs} on PMOS and negative bias V_{gs} on NMOS (all are inverse bias) result faster and stronger recovery than natural recovery

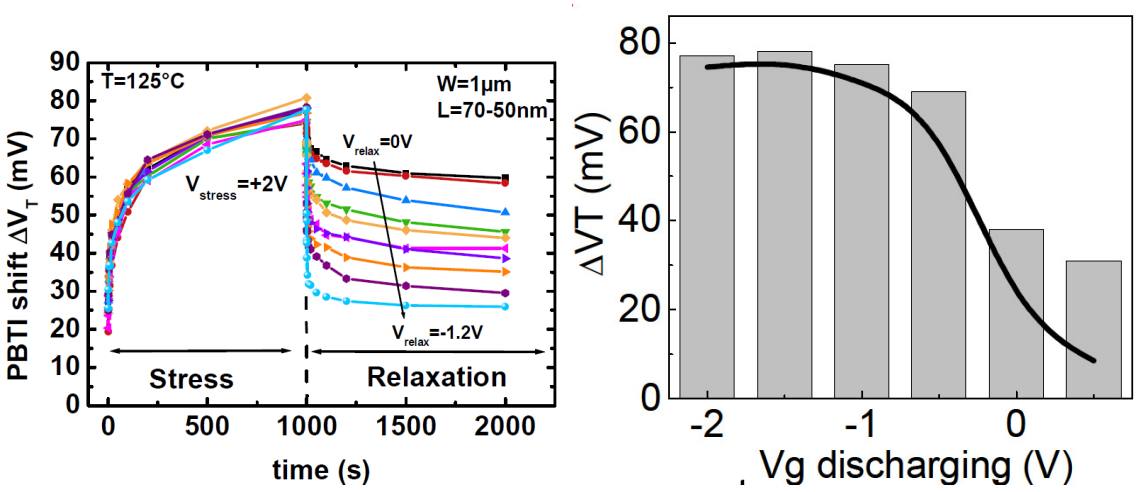
($V_{gs} = 0$) [29, 28, 50]. Although this recovery strength is weaker than using full inverse bias ($V_{gs} = V_{dd}$ in [65] and $V_{gs} = -1V$ in [50]), the measured results in [50] indicate that we can still achieve 89% of V_{th} shift recovery compared to using full positive bias, while the natural recovery can only achieve 50%. Moreover, since there are no transistors under stress in our *4PR* mode, we can double the recovery time that was achieved in [65]. Also, the concurrent recovery simplifies the circuit design.

3.2.3 Further discussions

To evaluate the effectiveness of our *4PR*, we need to closely examine the recovery characteristics under different magnitudes of discharge bias in real SRAM cell. As mentioned earlier, in multiple experiments [28, 58, 50, 29] the recovery is accelerated under inverse bias of V_g with source, drain and substrate terminals grounded. Although the settings (temperature, stress time, stress voltage, measurement method) of these experiments vary, it is clear that the inverse bias accelerates the PBTI recovery significantly. In [28], at 10^4 seconds, applying V_g of $-1.2V$ results in 43% recovered V_{th} shift, comparing to 8% for grounded gate. In [58], the final degradation is reduced by 31%, compared to that after same cycles of alternating stress and natural recovery. In [50], applying $-1.5V$ bias achieves full recovery, while applying $-0.5V$ achieves 87% recovery and natural recovery only achieves 48%.

However, in real SRAM cell circuits both the natural recovery and proactive recovery under *4PR* mode differ from those in previous experiments. In natural recovery, inverse bias occurs at the gate-drain overlap (for NMOS, $V_g = V_s = 0, V_d = V_{dd}$), contrary to the setting of natural recovery in experiments without any bias (for NMOS, $V_g = V_s = V_d = 0$). In *4PR*, a mild inverse bias is forced at the original gate-source overlap (for NMOS, $V_g = V_d = 0.5V_{dd}, V_s = V_{dd}$), which is different from the mild/full inverse bias on both gate-drain and gate-source ends.

For natural recovery, in [58], the inverse bias at the gate-drain end shows negligible impact on recovery strength: even raising the V_{ds} from $1.2V$ to $1.6V$ only results in limited improvement of recovered V_{th} from 3% to 10%. The great diminish is due to the reduced electrical field in the channel where there is no inverse bias for V_{gs} or V_{gb} and consequent



(a) PBTI proactive recovery [17]

(b) PBTI proactive recovery [50]

Figure 9: BTI proactive recovery strength

reduced electron tunneling current for charge neutralisation. Although inverse V_{gd} marginally accelerates recovery, it might be hasty to assume the inverse V_{gs} bias in *4PR* has more impact on recovery strength than V_{gd}. Since there are no experiments performed under varying gate-source bias for authors' best knowledge, it is hard to quantify the individual impact of 2 inverse biases (V_{gs}, V_{gd}) on the detrapping process and the recovery strength. Since *4PR* puts gates into a new recovery state, we conservatively studied two extreme cases to include possible scenarios where V_{gd} might play a role.

1. *4PR*-best. The first case is that V_{gd} has little impact on the gate recovery, so only V_{gs} dominates the recovery strength, as with all previous researches. The *4PR* can achieve strongest recovery in this case.
2. *4PR*-worst. The second case is that V_{gd} has the same impact on gate recovery, so combined V_{gs} and V_{gd} should be used to determine the recovery strength. The *4PR* can achieve weakest recovery in this case.

However, even under the worst case, it is expected that more V_{th} shift is recovered when the inverse V_{gs} bias is applied in experiment of [58] after multiple cycles of stress and natural

recovery. Since in real circuits the gate-drain has the inverse bias while the gate-source has only 0 bias, the recovery on gate-drain is always stronger than gate-source. Therefore, the accumulated trapped charge in dielectrics at the gate-source end is denser than that at the gate-drain end. As a result, periodically applying inverse V_{gs} effectively removes the charges, compared to the saturated recovery at the gate-drain end.

We list V_{gs}, V_{gd}, recovery strength and recovery time for different modes of P/NMOS in Table 2. Take the NMOS as an example: in the stress mode, $V_{gs} = V_{gd} = V_{dd}$. In the natural recovery mode, $V_{gs} = 0$ and $V_{gd} = -V_{dd}$. In the "SP PR" mode, $V_{gs} = V_{gd} = -V_{dd}$. Previous study [50] has shown that a full inverse bias to the gate results in full recovery of V_{th} while a zero bias results in a 50% of recovery. Hence, the recovery strength ratio of Natural vs. SP PR is 1:2. Regarding recovery time, the SP PR mode dedicates certain percentage of time for recovery on SRAM cache bank. Within this recovery time, only one PMOS is recovered, so the total effective recovery time for PMOS gates is 50%. Outside of the proactive recovery interval, the PMOS still enjoys the natural recovery opportunities. For natural recovery, the recovery time of any gate in a cell depends on the cell's signal probability x. Since the recovery time is limited by the weakest gate in the cell, the effective recovery time is always less than 50%.

Table 2: Recovery Comparison: the magnitude of V_{gs}, V_{gd}, recovery strength and time under different modes. P represents PMOS. N represents NMOS.

condition	V_{gs} (P/N)	V_{gd} (P/N)	strength	time
Stress	-/+V_{dd}	-/+V_{dd}	N/A	1-x
Natural	0	+/-V_{dd}	0.5	min(x, 1-x)
SP PR	+/-V_{dd}	+/-V_{dd}	1	50%
4PR-best	+/-0.5V_{dd}	0	0.89	100%
4PR-worst	+/-0.5V_{dd}	0	0.52	100%

The recovery strengths of *4PR*-best and *4PR*-worst are shown in Table 2. The full recovery under SP PR mode is normalized as 1. In *4PR* mode of an NMOS, $V_g = V_d = 0.5V_{dd}$ and $V_s = V_{dd}$. Hence, $V_{gs} = -0.5V_{dd}$ and $V_{gd} = 0$. In *4PR*-best, V_{gd} can be ignored and V_{gs} is half of that in SP PR mode. Hence, *4PR* can achieve 89% of the recovery strength of SP

PR mode, for all 4 gates. The *4PR*-worst mode might be weaker than the Natural mode but due to the effective recovery on gate-source end, the recovery score is 52%.

When it comes to recovery time, the *4PR* mode has a clear advantage over all other modes. As we will explain later, both *4PR* and SP PR modes have the same dedicated time for SRAM bank recovery. During this time, *4PR* ensures recovery for all 4 gates while SP PR provides recovery for only one PMOS gate. Outside this recovery time, both schemes enjoy natural recovery opportunities.

The last issue is the potential impact of body terminal on the gate recovery. Although this impact is likely to be small, it can still be easily solved by applying a proper body bias to cancel its effect on the gate.

3.3 SUMMARY

This chapter analyzes the new reliability problem in SRAM cell when PBTI is as important as NBTI in high-κ metal gate technology. The analysis shows PBTI introduces the access time failure and dominates read flip failure, which are more important and were not surfaced when only NBTI is considered. Hence we design innovative *4PR* mode to proactive recover both NBTI and PBTI in all 4 MOS gates in SRAM cell at the same time. The design of applying *4PR* on cache and storage entries in pipelines are discussed in following chapter4.

4.0 *4PR* ON L2 CACHE AND OUT-OF-ORDER ENTRIES

To take advantage of the *4PR* recovery mode, we exploit the opportunities in SRAM cache and storage cells in out-of-order pipeline respectively. In SRAM L2 cache, we utilize the spatial redundancy of a spared bank to hold the contents of the bank being proactively recovered. However, adding spatial redundancy on cells in out-of-order pipeline hurts the performance significantly since the pipeline is on critical path. Instead, we identify the idle and inactive cycles of these cells and apply the *4PR* mode during the inactive cycles. We observed that the entries in reservation stations (RS), reorder buffers (ROB) and physical registers (PR) present on average 43∼76% idle cycles for SPEC2000 benchmark programs. In this way, we achieved better recovery effect without adding extra backups. A simple, entry level controller is designed to explore the inactive cycles in entry-level granularity, with <1% performance and area overhead.

4.1 USING A SPARE BANK FOR *4PR* IN L2 CACHE

We apply our proposed recovery technique in L2 cache, similar to the one in [65]. Since L2 cache is naturally sub-banked, the basic recovery scheme is to recover each bank in rotation. However, since the bank loses data once entering the recovery mode, we adopt similar strategy as in [65] to leverage the spare bank for data backup and restore. At any time, there are 64/32/16 cache banks (3 configurations) in the normal mode and 1 cache bank, either existing or spare cache bank, in the recovery mode. The recovery cycles varies from 10K to 5000K cycles.

With a spare cache bank, the overheads in our design include area, latency and energy. The latency and energy overheads come from copy data to/from the spare bank. Alternatively, we can also simply discard the bank's contents (with dirty lines flush) for recovery and use the spare to make up for the cache capacity. No data copying is necessary. The latency and energy overheads come from the cold start of the empty spare bank, which generates more memory accesses. We will consider the former design in this thesis.

4.2 *4PR* OPPORTUNITY FOR BUSY FUS

Within a processor core, many critical and busy functional units (FUs) such as RS (reservation stations, or issue queues), ROB (reorder buffers), and PR (physical registers). It is preferable to apply *4PR* recovery for these FUs for two reasons: (1) they are more vulnerable to BTI-induced failure than other FUs since they are busy and hot in an out-of-order microprocessor, and BTI effect becomes more severe with higher biased probability and higher temperature; (2) they are crucial FUs whose reliability degradation may directly fail the entire chip.

However it is challenging to perform *4PR* within the processor core. First, these FUs are frequently used and thus do not have abundant idle time to perform *4PR* at the bank level, i.e the way to recover L2 cache banks in [65]. Second, there are no spare copies of FUs to store the old values. To overcome these difficulties, we design a per-buffer-entry level proactive scheme that exploits the idle time of individual entries, and performs proactive recovery during their *inactive* cycles i.e. the contents are no longer used and can be safely discarded. We next analyze the recovery opportunities in these FUs, and present our hardware design in the next section.

4.2.1 FU Activities

Let us first analyze the idle and busy cycles of above FUs in different stages of execution: dispatch, issue, execute, complete and retire (commit).

Instructions are fetched and decoded sequentially. Prior to dispatching an instruction to the out-of-order engine, the hardware has to check resource availability based on three conditions:

1) is there a free PR to hold the output?
2) is there a free entry in RS to hold the operand tags? and
3) is there a free entry in ROB to hold related information?

If any condition is not satisfied, the instruction is stalled without allocating any entry in these FUs (PR, RS and ROB). Otherwise the instruction is dispatched, and each FU allocates a free entry for the instruction. Those entries turn from *idle* state to *busy* state.

In the issue and execute stages, all the allocated entries are busy, so recovery is forbidden. The transition from busy state to idle state is associated with the deallocation of the entries. In non-speculative execution, the RS entries are deallocated in the complete stage and the ROB entries are deallocated in the retire stage. A PR is freed only when all the following three conditions are satisfied:

1) it is not referenced by the register aliasing table (RAT);
2) there is no pending consumer (reader); and
3) there is no unresolved branch instruction.

In case of a mis-speculation, the entries in the RS and ROB on the mis-predicted path are squashed and deallocated. Also, the newly allocated PRs associated with the squashed ROB entries are freed. And the RAT is recovered to the state where the mis-prediction begins. However, no PRs prior to the mis-predicted branch are allowed to be freed because otherwise, their architectural states would be lost. This corresponds to the third condition of freeing a PR above.

In summary, when a RS/ROB/PR entry is allocated, it is in the busy state and recovery cannot be performed. Once it is deallocated, it becomes free, and the content can be discarded for recovery purpose – *4PR* destroys the stored information. Therefore, we keep tracking the entry states and capture the idle cycles to perform *4PR*. However, *4PR* incurs delays when driving down and up the virtual V_{dd} and *Gnd* of entries. Hence, the idle period of an entry should be sufficiently long to encompass effective recovery cycles.

4.2.2 Idle Cycle Analysis

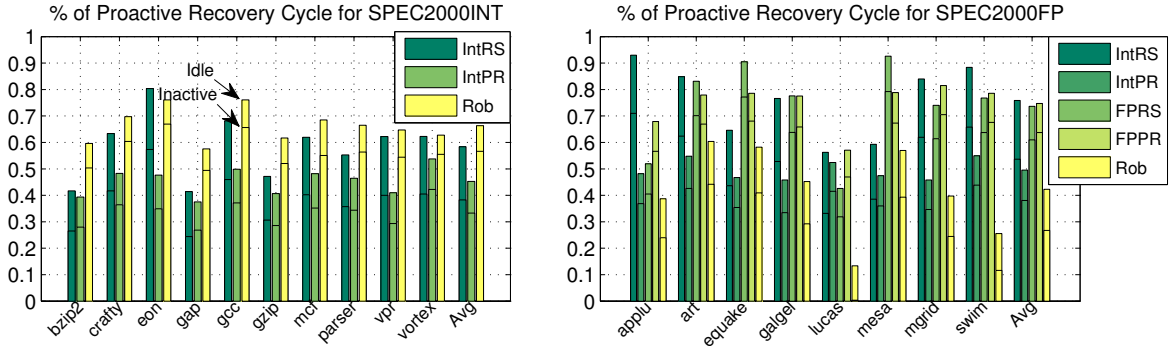


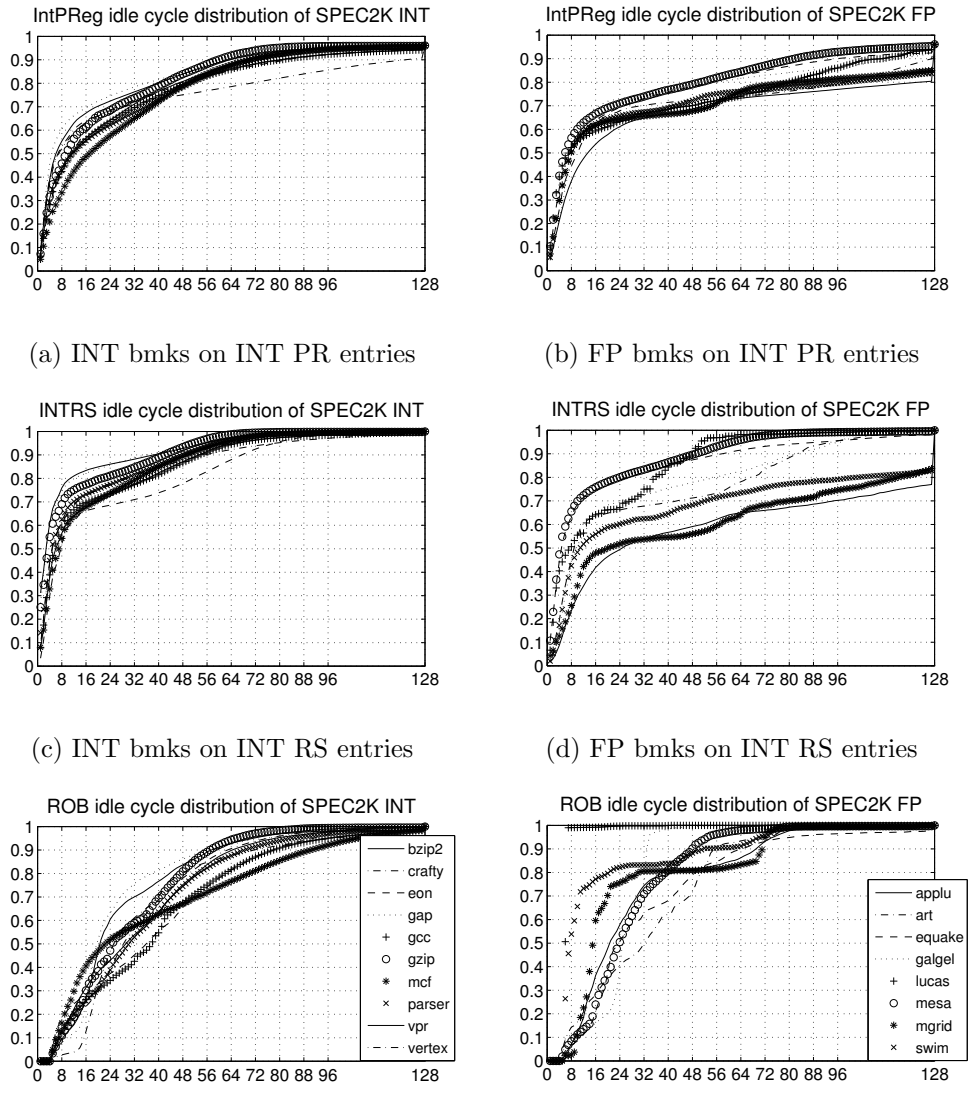
Figure 10: Idle time percentages for SPEC2000 INT and FP benchmarks; Cumulative Distribution of idle durations for Rob under SPEC2000 FP benchmarks.

Fig. 10 reports the percentage of time that RS/ROB/PR entries stay idle for SPEC2000 INT and FP benchmark programs. For each stacked bar in the figure, the top line (labeled as *idle*) indicates the idle cycle percentage; the lower line (labeled as *inactive*) indicates the portion that can be recovered — it will be discussed in 4.2.3. Here we applied *round-robin* to allocate buffer entries in each FU. A round-robin allocation scheme favors reliability-aware designs as it balances the idles cycles over all entries. Our results showed that the entries in each FU have similar busy/idle cycle distribution and the average is reported in the figure.

The results illustrate that there are plenty of opportunities for *4PR* in those FUs. For example, for benchmark `applu`, the average idle time per entry accounts for over 90% of execution time for integer RS (IntRS). From the figure, the average percentages of idle time for integer RS, integer PR, floating point RS, floating point PR, ROB are: 76% (58% for SPEC2000 INT), 50% (45% for SPEC2000 INT), 74%, 75%, and 43% (66% for SPEC2000 INT). Our observation is consistent with previous work that shows 54%/69% of idle time for integer/FP PR [1], despite of different benchmarks and processor configurations used in this thesis. Notice that we utilize idle cycles at the *entry level* of those FUs, which gives us more opportunities than does at the entire FU level. Idle cycles of those entries come from under utilization of the out-of-order pipeline due to reasons such as cache misses, mis-predictions, etc.

4.2.3 Recovery Opportunities with Latency Overhead

Since performing *4PR* incurs extra delay, i.e., it takes several cycles to change an entry to and from recovering mode, not all idle cycles can be exploited for proactive recovery. In particular, if the duration of an idle period is smaller than the transition overhead, it might be not beneficial to apply proactive recovery.



(a) INT bmks on INT PR entries

(b) FP bmks on INT PR entries

(c) INT bmks on INT RS entries

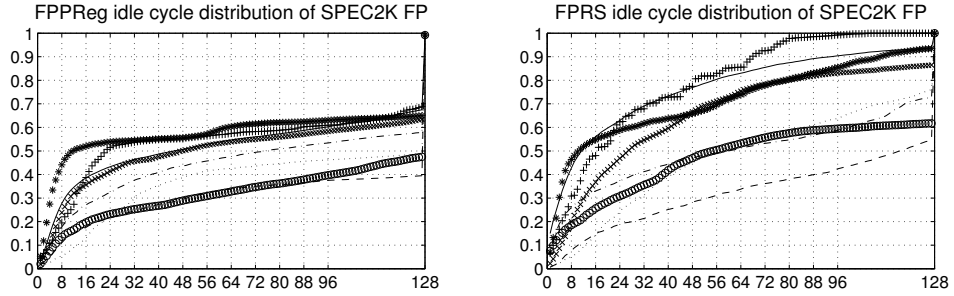
(d) FP bmks on INT RS entries

(e) INT bmks on ROB entries

(f) FP bmks on ROB entries

Figure 11: Idle time distribution of INT and FP benchmarks on different structures.

To study the recovery opportunities with transition overhead considered, we plotted the cumulative percentage distribution of idle periods for multiple FUs in SPEC2000 FP benchmark programs in Fig. 11 and Fig. 12. We use the results for ROB as example. Here the x axis is the duration of idle periods spanning from 1 to 128 cycles. We lump sum the duration beyond 128 cycles to 128, so all curves eventually reaches 1 at 128. In general, the slower a curve rises, the higher the percentage of longer idle durations. For example, ROB entries for *art* and *equake* tend to have longer idle durations than other programs, which corresponds to the highest average idle percentage shown in Fig.10(middle). A sharp rise of the curve indicates high percentages of the corresponding idle periods. For example, the *lucas* has a sharp rise for the idle period duration between 6 and 7 cycles, which indicates ∼50% of idle period durations are less than 6 cycles and almost all the rest are 7 cycles.



(a) FP bmks on FP PR entries

(b) FP bmks on FP RS entries

Figure 12: Idle time distribution of INT and FP benchmarks on different structures.

Having many short idle durations reduces the recovery opportunities. For example, if the overhead is 5 cycles in total, then it has no or negative benefit to proactively recover those idle periods that are less or equal to 5 cycles; and it has small benefits when recovering the idle periods slightly larger than 5 cycles. In Fig. 10, the lower portion (labeled as *inactive*) of each stacked bar indicates the percentage of idle cycles that can be exploited for recovery. Here we assume the transition overhead is 5 cycles as modeled in 4.2.4. As an example, for the ROB of `lucas`, there is little opportunity for proactive recovery, which gets explained by the sharp rise in Fig. 10(right). From the figure, assuming a fast per-entry based proactive recovery scheme can be designed, we found that a large amount of idle cycles

can be exploited. On average, out of the 43∼76% cycles identified as idle in aforementioned FUs, 27∼61% cycles can be exploited for proactive recovery.

In Fig. 10 we also see that RS and PR are busier than ROB when running SPEC2000 INT programs; and the ROB is the busiest when running SPEC2000 FP programs. This is because floating point instructions have longer latencies, and tend to stay in the ROB waiting for earlier long latency instructions to produce the results.

4.2.4 Hardware Design for *4PR* in Busy FUs

To exploit the exposed recovery opportunities in the preceding section, we next present our hardware design (used as IP block in the EDA flow) details and then model its latency and area overheads.

To assist the discussion with proactive recovery, we distinguish three states for each entry in RS/ROB/PR — a *busy* state indicating the entry has been allocated; a *ready* state indicating the entry can be allocated and enter the busy state immediately; an *inactive* state indicating the entry is in recovering mode and has to switch to the ready state before being allocated again. The *ready* and *inactive* states correspond to the *idle* state when no BTI recovery is considered. Switching from busy to inactive state (i.e. entering recovery), and from inactive to ready state (i.e. exiting recovery) incur the transition overhead. The transition between busy and ready states does not incur extra overhead.

Our *4PR* is performed at per-entry level. When an entry is marked as *inactive*, the virtual power/ground to this entry is driven to physical ground/power. To achieve maximal recovery, it is preferred to have more idle entries in the inactive state. However, inactive entries must exit recovery mode before being allocated to instructions. In order to avoid structural hazards in the dispatch stage, there should be enough ready entries available, and some inactive entries may have to switch to ready state in advance. This is especially necessary for multi-issue processors that allocate multiple RS/ROB/PR entries per cycle. Thus we need to design a control logic to achieve better trade-off between the attainable reliability improvement and the performance.

As shown in Fig. 13, the control logic for enabling per-buffer-entry consists of a power-down logic, a wake-up logic, and a ready entry counting logic.

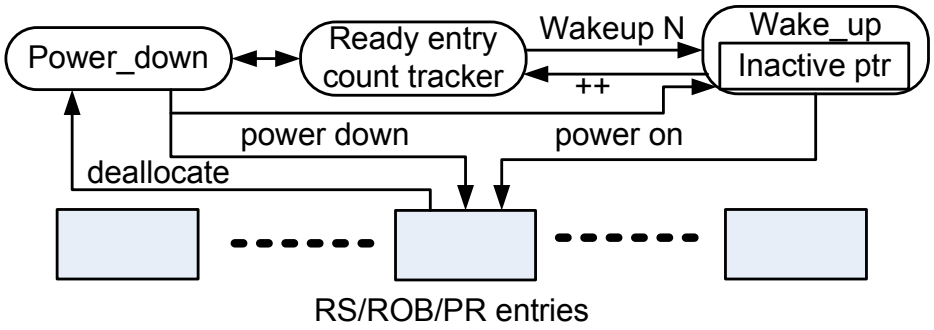


Figure 13: Per-entry based recovery control logic.

Power down logic. When a busy entry is to be deallocated, the power-down logic determines its next state which is either ready or inactive state. An inactive state automatically triggers *4PR* to recover the corresponding entry.

By monitoring the status of each entry, the logic detects its deallocation point as early as possible such that the recovery benefits can be maximized. For RS and ROB, the logic monitors the busy bit associated with each entry. The RS entries are deallocated right after the execution stage, and the ROB entries are deallocated right after the commit stage. For the PR entries, the logic needs to check the reference bit, the consumer list and the speculation mode bit. A PR entry can be deallocated only when it is not referred by any RAT, i.e. there are no pending consumers or unresolved branches.

To determine the appropriate next state, the power-down logic strives to ensure enough ready entries in each FU such that structural hazards can be avoided and performance loss can be minimized. It tracks the number of ready entries, and turns the state of a to-be-deallocated busy entry to ready state if the number is below a threshold. Otherwise the logic turns the entry to inactive state. The threshold number of ready entries correlates with the issue width and the transition latency from inactive state to ready state. The reason for the latter correlation is that inactive entries are forced to switch to ready state in the case if there are inactive entries, and instructions get stalled due to insufficient ready entries. Since this transition costs 2 cycles (as discussed in Section IV.C), we found that preparing three

times the issue-width of ready entries is sufficient and not too conservative. As an example, we maintain 12 ready entries in each FU for a 4-issue microprocessor.

An exception occurs when a mis-speculation is detected. At this time, the processor state is rolled back to the mis-predicted branch, and the corresponding RS and ROB entries are deallocated. These entries are highly likely to be re-allocated again along the correct branch path. Hence, they are turned to ready state directly, without additional checks. This decision favors performance, and also has trivial effect to recovery as the mis-prediction rate is low.

Wake-up logic. The power-down logic alone cannot maintain sufficient ready entries all the time since there are times when instructions do not commit for a long time due to a cache miss. We use the "wake-up" logic to dynamically monitor the number of ready entries. If the number of ready entries is below the threshold, the wake-up logic will locate several inactive entries and switch them to ready state. The wake-up logic and power-down logic work closely to balance the trade-off between attainable reliability and performance.

Ready entry counting logic. When a RS/ROB/PR entry is deallocated, the power-down logic gets notified. It then queries the *ready entry count tracker* to decide if it should apply the proactive recovery to the entry. If no action is taken, the entry is in ready state by default. The tracker monitors if there are sufficient number of ready entries in each cycle. If the repository drops below the threshold, a signal is sent to the wake-up logic to wake up necessary number of inactive entries. Since the entry allocation/deallocation runs in a circular queue manner, the wake-up logic keeps pointers to the first few inactive entries for ease of management. The pointers are updated by the power down signal (increment) and the power on signal (decrement). The *ready entry count tracker* maintains the current number of ready entries. The counter is updated when 1) a ready entry is allocated by the dispatcher (not shown); or 2) the power-down logic leaves a busy entry in the ready state; or 3) the wake-up logic powers on an entry.

4.3 OVERHEAD ANALYSIS

4.3.1 Recovery Control Logic

The latency overhead includes the delay in deciding whether to apply proactive recovery and the delay of driving the virtual V_{dd}/Gnd to ground/power. We synthesized the control logic for all RS/ROB/PR units using $45nm$ technology. The delays for making power down and up decisions are 2 and 1 cycles respectively. To quantify the delay of driving the virtual V_{dd}/Gnd to ground/power, we built the circuit of the register entry with 4 read ports and 2 write ports using 45nm technology library [15] in Cadence Virtuoso. With proper selection of the stages and sizes of the buffers, the rising (0→0.95 V_{dd}) and falling (V_{dd}→0.01V_{dd})) time of virtual V_{dd}/Gnd for a 64-bit register or ROB entry are 130ps and 130ps respectively (2 stage buffer). For a 256-bit RS entry [1], the falling and rising time are 150ps and 150ps respectively (3 stage buffer). That is, it takes 1 cycle for a 2Ghz core to drive the virtual V_{dd}/Gnd to ground/power for any single entry in these FUs. The time to drive a single entry is much faster than that to drive the whole FU as it requires smaller diffusion capacitance on the gating path and has stronger drive current in NMOS. In summary, it takes 3 and 2 cycles respectively for the transition of entering and exiting the proactive recovery mode.

The area overhead of the per-buffer-entry proactive recovery logic includes the idle detection logic, the buffer to drive the virtual V_{dd}/Gnd, and the center control logic. The idle detection logic consists of simple AND gates based on the analysis in Section 4.2.1, with the state bits from different entries as the input. The buffer logic incurs trivial area overhead due to the small load of a single entry. The storage cell in 45 nm technology is $1.144\mu m^2$, extracted from CACTI [40]. The buffer area of the 64/256-bit entry is $0.408/1.632\mu m^2$, adding only 0.56% overhead to the design. As for wring, using the similar design of controling the wordline drivers in [74], the per-entry control is implemented without adding wires in SRAM arrays. Hence, only wiring the control signals from the control unit to every entry adds the area outside the SRAM array. Besides the logic, virtual V_{dd}/Gnd rails are needed, which often exist [65] or can be minimized with proper layout planning. The area of controller unit for all FUs is $0.12mm^2$ ($<<$1% for most out-of-order processors), and power is $2.677mW$.

4.3.2 L2 Cache Recovery Logic

We built a 128_bit×256_word bit bank (without considering the peripheral circuitry) using 45nm technology to measure the design overhead, including the power, latency and area. The power includes mode transition power, data read/write power for backup and restore, and leakage power. The breakdown of each category is listed in Table 3.

We also measure the latency of mode switches between *Normal* and *4PR* . Based on our test SRAM bit bank, with a properly designed 5-stage driver, it takes less than 5ns to finish the transition. Compared to the recovery cycles of 10K to 1000K, the transition overhead can be omitted.

The area overhead mainly comes from the spare cache bank, virtual V_{dd}/Gnd wires and virtual V_{dd}/Gnd drivers. We use the same spare cache bank and same number of wires for virtual V_{dd}/Gnd as in [65]. The only difference is that our design uses two wires for virtual power and ground, and the previous design uses two wires for left/right side virtual powers. As shown in [65], the virtual power wires already exist in modern memory designs, so our wiring requirement does not incur too much overhead. The spare bank's area overhead is 1/17 at most since there are altogether 17 banks with 1 being the spare. The area of drivers is roughly proportional to bank size, and is found to be negligible.

Table 3: Energy for test 128_bit×256 SRAM bank.

Dynamic Energy		Leakage Power	
operation	energy(pJ/bit)	mode	power (mW)
read	0.043	Normal	1.906
write	0.315 (max)	SP PR [65]	1.424
pre-charge	0.78 (read preceded)	*4PR*	1.409
	4.37 (write preceded)		
mode switch	100.35 per bank		

The mode switch power is incurred on each mode switch: the mode control circuit (with 5-stage drivers for VV_{dd} and VG_{nd}) consumes $100.35pJ \times 2 = 0.201nJ$ for the whole bank. Additional cache bank copy and restore consists of 2×256 reads and writes which amount to 0.421 and 2.399nJ respectively. In summary, the dynamic energy overhead in each recovery

transaction is a little over 512 normal reads and 512 normal writes. Consider that this extra energy is charged on every $10^4 \sim 10^6$ cycles, the overall dynamic energy overhead is very small.

For the leakage power, the test bank under *SP PR* and *4PR* modes consumes 1.424mW and 1.409mW, respectively. The reduction over the *Normal* mode (1.906mW) is due to the reduced V_{DS} of the gates in cut-off mode. For *SP PR*, the V_{DS} of the MOS under proactive recovery is 0; for *4PR*, the V_{DS} of 4 transistors are $\sim 0.5V_{dd}$.

4.4 SIMULATION AND RESULTS

4.4.1 Simulation Setup

The design is evaluated on an out-of-order processor whose configurations are summarized in Table. 4. We used a cycle accurate simulator enhanced from Simplescalar Sim-Alpha [10] to study the recovery opportunity, the 0/1 input distribution and thus signal probability at runtime, and the performance impact of our design. The original Sim-Alpha simulator used a unified data structure to model the timing effect of RS, ROB and PR for simplicity. To accurately characterize these FUs, we modeled them independently following the discussion in Section 4.2.1. The ROB entries store the instruction order information. The RS entries track and resolve the operand dependency. The PR entries hold the source operands. We chose 9 integer (INT) and 8 floating-point (FP) programs from SPEC2000 benchmark suite. We skipped their warm up phases, and simulated one billion instructions for performance study.

We built the aforementioned SRAM bit bank circuit based on the PTM 45nm high-performance model [8] for high-κ metal-gate stack. To measure the failure events discussed in Section 3.1, we used the metric of dynamic stability through transient analysis in Synopsis Hspice [33] due to its high accuracy. We built the normal circuit with certain guard-band according to the specification, and then simulated the circuits with V_{th} shift. The write time, access time, voltage overshoot during a read were all measured and summarized in Fig. 5(no

Table 4: Processor configurations.

Configuration	Parameter
ISA	Alpha 21264
Reservation station	60 FP and 60 Integer
Reorder buffer	128
Physical register	128 INT and 128 FP
Load store queue	32
Fetch queue	32
L1 I-cache	32 KB 4way
L1 D-cache	32 KB 4way
L2 unified cache	4MB 8way

PV). The same framework was used in evaluating cell failure stability (Section 4.4.4 with PV).

Since the degree of stress in the 4 gates of a SRAM cell depends on the stored value, we extracted the signal probability of sample bits in a 256KB 16-way L2 cache by running the same SPEC CPU2000 benchmarks on the enhanced Sim-Alpha simulator. We fast-forwarded the initialization phase, and ran 1 billion instructions of all 17 benchmarks. The extracted signal probability was used to calculate the V_{th} degradation by Eqn (2.2) [31].

4.4.2 Reliability model

In order to quantify the reliability improvement after applying the 4PR on busy FUs and L2 cache, we modeled the V_{th} shift (threshold voltage shift), MTTF (Mean Time to Failure) for all the units.

In this section the failures are limited to those due to BTI (both NBTI/PBTI) induced V_{th} degradation. In the following discussion, we only focus on the access failure as it is the dominant effect (as confirmed in Section II). We treat busy FUs and L2 cache banks differently due to their different ECC configurations. Since there are no ECC bits in busy FUs, one single bit failure is a hard defect and can not be corrected. We conservatively assume the first hard bit failure cause the failure of the FU. For L2 cache, we assume the

common configuration that 2-bit correction ECC code to enhance the reliability. As a result, the 3^{rd} bit failure indicates the line failure.

The $V_{th}(t)$ is calculated in according to Eqn. 2.2 in chapter 2. The parameters η in Eqn. 2.4 for P/NBTI are individually calibrated to match the experiment data in [58]. For stronger recovery, both in the single-pair proactive recovery mechanism derived from [65], denoted as "SP PR", and our *4PR* with stronger and faster recovery, the parameter η is calibrated by the published data in [29, 50]. For different S, the time for stress and recovery are different and the final V_{th} are different. Based on the multiple cycle simulations, the $\alpha(S)$ is poly-fitted in Matlab.

4.4.3 V_{th} shift comparison

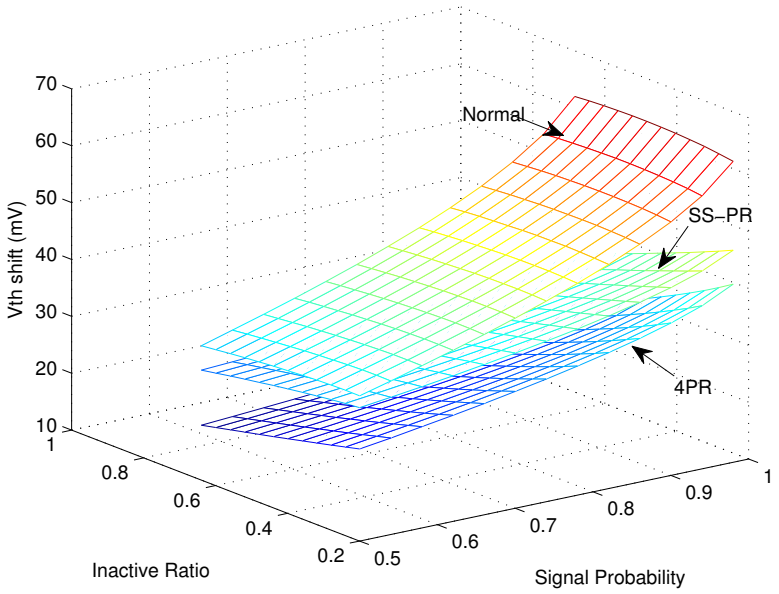
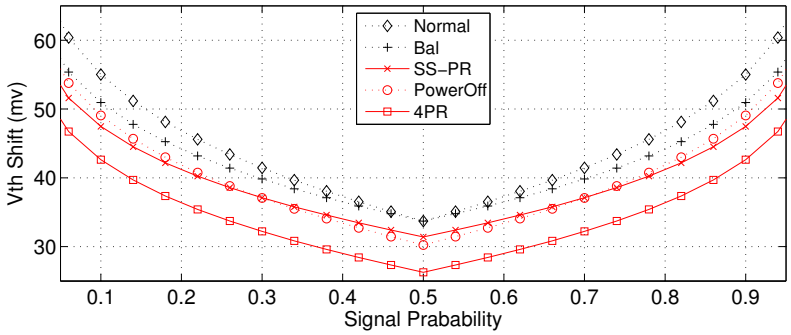


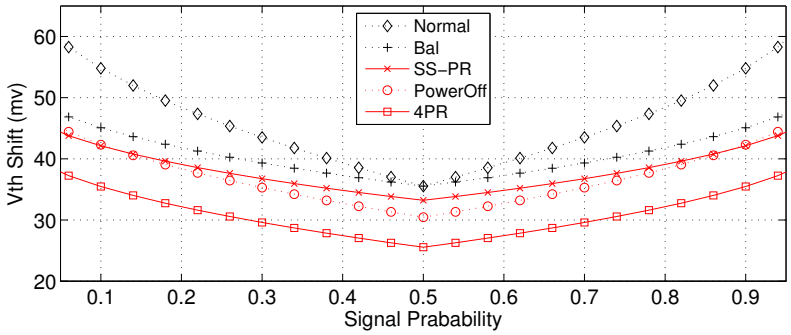
Figure 14: V_{th} shift for varying signal probability and inactive ratio

We first compare the effectiveness of the aforementioned schemes in suppressing V_{th} shifts: *Normal* mode, *Bal* mode in which the bits are flipped to balance the signal probability derived from [1], *SP PR* which is the single-pair proactive recovery derived from [65], *4PR-worst* and *4PR-best* are two extremes of the *4PR*. In addition, we also experimented for the "Power

Off" scheme which turns off the cache bank and all voltages are 0 such that all 4 gates are put in a weak-natural recovery mode (Normal recovery still has a bias in V_{gd}). We include this case because it is also a practical recovery scheme and natural to apply in reality.



(a) V_{th} shift for L2



(b) V_{th} shift for FUs in Pipeline

Figure 15: V_{th} degradation for L2 and FU in pipeline. The degradations are different due to different inactive ratios.

Fig. 14 shows the larger V_{th} shift (in 10 years) between two NMOS (similar trend for PMOS) in SRAM cell versus different signal probabilities and inactive ratios. Only *Normal*, *SS-PR* and *4PR* are shown. Since only the larger signal probability of two MOS in one cell is picked, it ranges between 0.5 and 1. The signal probability varies for different benchmarks. The inactive ratios of busy FUs in pipeline are extracted by our hardware, while that of L2 cache bank is fixed once how many cache banks in rotation are fixed. In Fig. 14, the trend for all schemes are similar except for *Normal*: more V_{th} shift with more stress signal probability and/or less inactive ratio. In *Normal*, V_{th} shift only depends on the signal probability, and

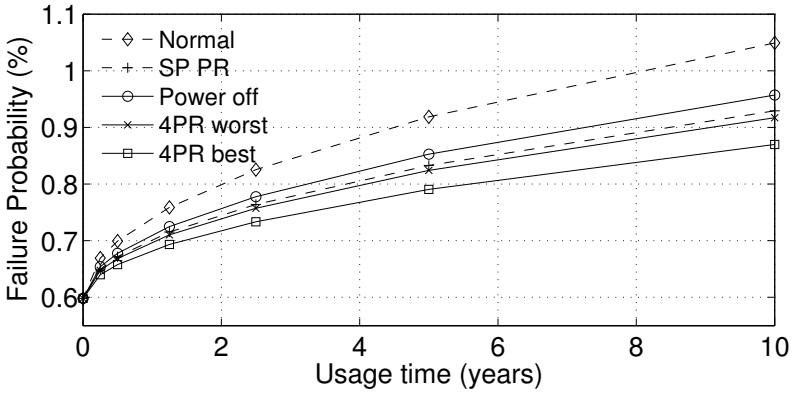
is irrelevant of inactive ratios. This is due to the fact that nothing is done to improve the reliability in inactive cycles. Comparing multiple recovery schemes, our *4PR* is strongest.

To better understand the comparison, Fig. 15(b) shows the V_{th} shifts under varying signal probability with fixed inactive ratio 24%, and Fig. 15(a) shows that with fixed inactive ratio 6.7% for the L2 cache configured as 16 normal with 1 spare bank. All curves are symmetric to signal probability of 0.5, because under signal probabilities of α and $1-\alpha$, the V_{th} shift of two pairs are symmetric. And the best result happens when the cell stores 0 and 1 in a 50%-50% split in time where stress is perfectly balanced to the two pairs. The largest V_{th} shift is always seen in the *Normal* mode, as no strong recovery mechanism is performed, and natural recovery alone is inadequate. Then comes the "SP PR", which are nearly equally effective with "SP PR" being slightly better for balanced stress and "Power Off" being slightly better for unbalanced stress. This is not surprising as both techniques intentionally recover internal MOS gates and "SP PR" uses strong recovery for 1 PMOS-NMOS pair but "Power Off" uses weaker recovery for all 4 MOS's. Finally, the *4PR* wins over the previous ones in all data-points even in the worst case since we use stronger recovery (than "Power Off") for all 4 MOS's and longer recovery time than SP PR. For L2 cache on average, 4PR reduce the V_{th} shift by 9.5mV and 4mV compared to *Normal* and "SP PR" in best case. For configurations of 64/32 banks + 1 spare bank, the reductions are 6.2mV/4.9mV and 3.3mV/2.6mV compared to *Normal* and "SP PR". Considering the improvement of lifetime, 1/17 area overhead is tolerable. For busy FUs in pipeline, the results are better since more inactive cycles are available for recovery on average.

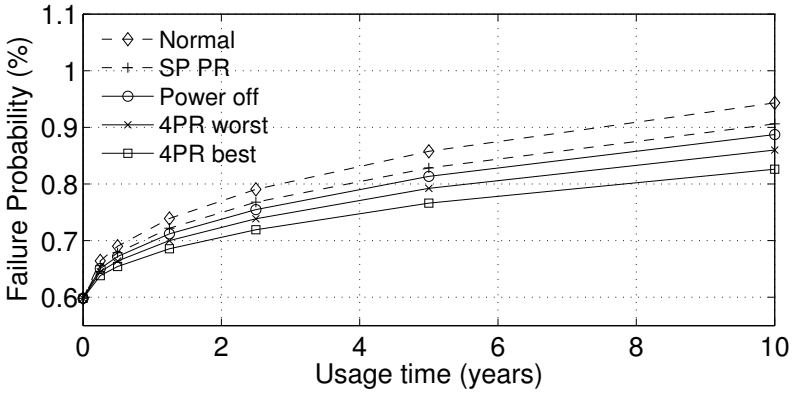
4.4.4 Cell failure probability analysis

In addition to BTI stress induced V_{th} shifts, process variations (PV) during fabrication also produces initial V_{th} shifts. The combined shifts will create cell failure statistically depending on whether they cancel or enhance each other. We performed Monte Carlo simulation to evaluate the probability of cell failure after a sustained amount of time (10 years) accounting for both PV and BTI induced V_{th} shifts.

We first used R [14] to generate initial V_{th} shifts for cells in all banks under normal distribution with $\sigma(V_{th}) = 30mV$ due to PV. These initial shifts may cause initial cell failures. Using the access time as the metric of measurement, we found this initial cell failure probability is 0.598%. Some of these defects can be mitigated by redundant rows in each cache banks or other failure protection mechanisms. Those that cannot be fixed will cause yield loss.



(a) Failure probability with most imbalanced stress



(b) Failure probability with perfectly balanced stress

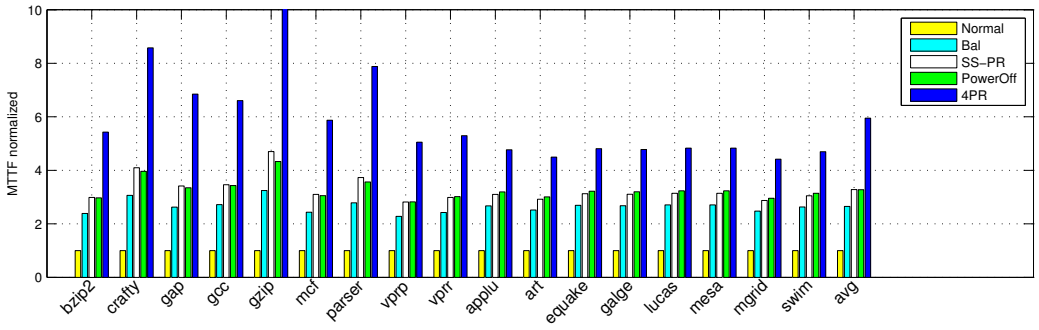
Figure 16: Failure probability analysis.

Next, we added V_{th} shifts introduced by BTI stress to the initial shifts, and ran Monte Carlo simulation to evaluate probability of failure over time. We assumed BTI induced V_{th} to be $50mV$ in the worst case after 10 years as we collected in Fig. 15. We experimented two extreme signal probabilities: (1) most imbalanced case that stresses only 1 pair of internal

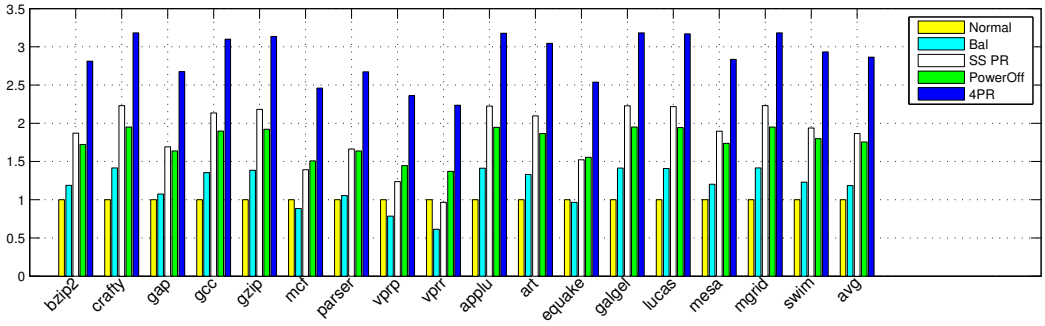
gates; and (2) perfectly balanced case that equally stresses both pairs of gates. The results were summarized in Fig. 16(a) for (1) and Fig. 16(b) for (2) respectively.

The *4PR* technique slows down the increase of the cell failure probability, since more V_{th} shift is recovered. The cell failure probability after 10 years under most imbalanced stress reduces from 1.05%/0.93% for Normal/SP PR to 0.87%/0.92% for *4PR*-best and *4PR*-worst. Under perfectly balanced stress condition, the cell failure probability is reduced from 0.94%/0.91% for Normal/SP PR to 0.83%/0.86% for *4PR*-best and *4PR*-worst.

4.4.5 MTTF improvement



(a) Busy FUs' MTTF improvement



(b) L2 Cache's MTTF improvement

Figure 17: MTTF Improvement

Then, we calculated the mean-time-to-failure (MTTF) using the V_{th} shifts in 17 SPEC2000 benchmarks. According to equation 2.2, the time when the V_{th} shift causes the access timing failure is calculated for each bit. Then, for the busy FUs in pipeline, the first bit failure

cause the function unit failure, since there is no correction circuit such as ECC. For L2 cache, the lifetime is determined when the 3^{rd} bit in a cache line failed because typically, there are failure protection mechanisms such as ECC for on-chip L2 cache lines. The results for busy FUs in pipeline and L2 cache are shown in Fig 17(a) and 17(b). Notice that there are no large MTTF variations among different benchmarks in L2 cache. There are 2 reasons for this: 1) cells that tend to fail sooner have extremely imbalanced signal probabilities close to either 0 or 1; 2) in all benchmarks the inactive ratios are same. In busy FUs, there are larger variation due to different inactive ratio exploited. Also, the MTTF improvement over the busy FUs are generally better since more inactive cycles are extracted.

Let's take L2 cache MTTF improvement as an example. The *Bal* only improves marginally (1.21x) over the *Normal*, due to the mild recovery strength and limited idle cycles for cells with extreme signal probabilities. The *SS PR* and *PowerOff* are neck and neck (1.8x to 1.9x), which is consistent with the V_{th} shift observations. Our *4PR* consistently outperforms those two techniques and achieves 2.85x MTTF improvement over "SS PR" and "PowerOff". These are considerable improvements compared with cell failure probabilities and V_{th} shift amount in Fig. 15(a). This is because V_{th} shift follows power law of time. In the long run, it take longer and longer time to achieve the same amount of V_{th} shift. Hence, a slight recovery of shift can increase the lifetime significantly. This is where a better recovery scheme becomes most effective. The MTTF improvements for busy FUs are even better, and our *4PR* achieves nearly 6x on average, compared to 3.4x of SS-PR and PowerOff.

4.4.6 IPC

Due to the transition delay to and from *4PR* in busy FUs in pipeline, the program performance degrades when insufficient ready entries are maintained in these FUs. Fig. 18 compares the IPCs with and without *4PR*. We observed on average 0.5% IPC slowdown when setting the threshold to be the triple of the pipeline issue width. The exception is a 3.7% slowdown for *lucas* in which case there is little recovery opportunity and putting entries into recovery state incurs extra overhead to switch back to ready state.

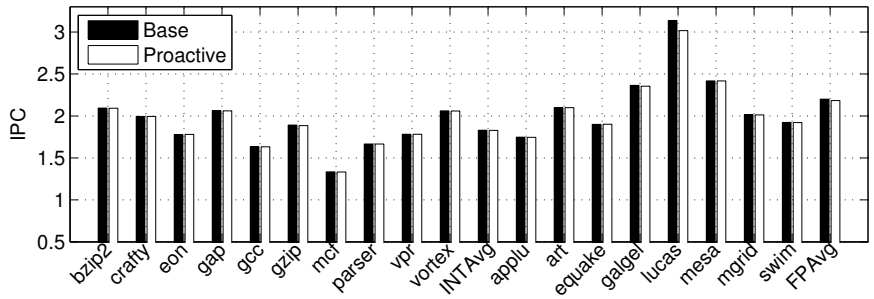


Figure 18: Impact of proactive recovery on IPC.

4.5 SUMMARY

This chapter elaborates the micro-architectural and circuit level design of applying *4PR* in L2 cache banks and busy function units to mitigate BTI stress. A spare cache bank is added to hold the data that are mapped from the bank under *4PR* and *4PR* is applied in a round-robin way. For PR, RS and ROB entries in Out-of-Order core, we extract entry-level idle cycles to apply *4PR*, with small area and performance overheads. Despite transition overhead of *4PR*, there are still abundant idle cycles, Compared to the baseline design with natural recovery, our scheme extends the FU lifetime by 6× and L2 Cache lifetime by 2.85×.

5.0 GENERIC FUNCTION UNIT

5.1 CHALLENGES OF FUNCTION UNIT RELIABILITY

As mentioned in Introduction, the SRAM cell based structures are protected from errors practically by many techniques due to their regular structure. However, irregular, non-memory core logic structures impose obstacles to detect and correct errors. Although core level and coarse-grained redundancy techniques are proposed for fault tolerance with high coverage, they are often over-designed and unnecessary, as a fault may only occur in one function unit (FU) and it is quite unlikely all FUs need replacement.

However fine-grained redundancy at FU level is also challenging: it is difficult to determine which FU is most vulnerable at design time. One solution in this direction [22, 23] attempts to reconfigure the pipelines of each physical core such that all non-faulty FUs from different cores form functional pipelines, to work around scattered broken FUs in different cores. Although significant reliability improvement is attainable, this design requires heavy re-engineering of the interconnection among all cores, as every pipeline stage must be able to reach the downstream stage in all other cores, which incurs non-trivial area, control and performance overhead since the inter-stage communication is global and expensive.

Hence this section discusses the Generic Function Unit (GFU) in Section 5.2, a promising framework for fine-grained FU level, low overhead and high fault coverage reliability improvement on FUs in execution stage. We discuss the how GFU is designed and integrated in the execution stage of microprocessor. By carefully studying the FUs of the execution stage at fine granularity, all units are partitioned into 3 groups:

- for the frequently used and logic sharable FUs, we design macro-based GFU/m to recover with low performance overhead;
- for memory structure dominated FUs, we refer to previous designs to mitigate aging effects;
- for all other FUs, STT-based fully reconfigurable unit is proposed. While it tends to have large overhead, we recover fine-grained FUs such that each particular FU has low usage and slowing down its execution tends to incur small performance overhead.

To further reduce the performance overhead, in Section 5.3 instruction steering is designed to dynamically determine the criticality of the instructions and whether to use the GFU or original FU. By steering the critical uses to fast FU, the performance overhead is minimized, with a controllable trade-off between reliability improvement and performance degradation. We simulated the generic functional unit (GFU) and instruction steering (IS) designs, and compared to other reliability enhancement designs. The results in Section 5.5 using methodology in Section 5.4 show that GFU extends the lifetime of execution stage in processor core by 2.2x while introducing less than 3% performance overhead.

5.2 GFU: GENERIC FUNCTIONAL UNIT

5.2.1 The Overview of GFU Design

Our basic recovery strategy is to utilize *natural recovery* on BTI and avoiding stress on TDDB. Multiple wear-out mechanisms, such as TDDB and BTI, only occur when the logic circuit is powered on. They are also heavily dependent on the amplitude of the voltage and the degrees of temperature [43]. The stresses disappear when the circuit is powered off. In this mode, TDDB-introduced stress stops while there is natural recovery of BTI-introduced stress on all gates. Although powering off FU is an effective recovery approach, the process of powering on/off a FU takes hundreds to thousands cycles for voltage to stabilize. Thus, to prevent introducing high performance overhead, the recovery mode should not be switched frequently.

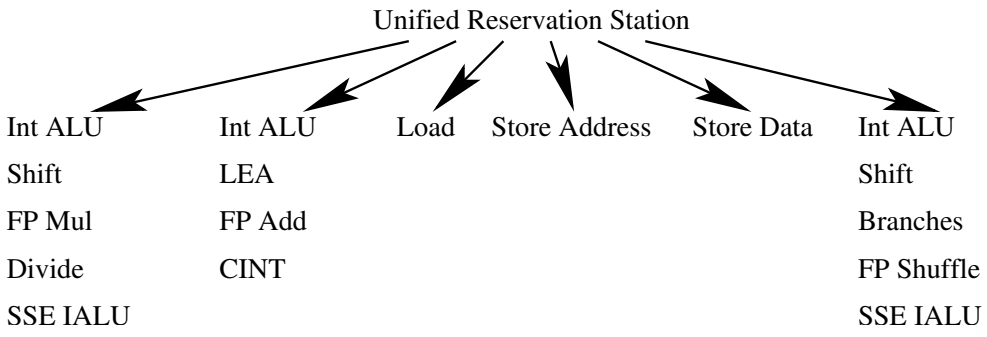


Figure 19: Intel Nehalem-like core execution units from [11]

In this chapter, we focus on improving the reliability of the function units in execution stage in modern processors. Such an execution engine, e.g., Intel Nehalem [64, 69], often has more than 30 functional units (FUs) (shown in Fig. 19 and some super-FUs are elaborated in Table 5) for executing the large and expanding instruction set. Studies [55] have shown that only a small number of these FUs are frequently exercised. Fig 20 shows the usage distribution of one Integer (*h*mmer at top) and one FP programs (*bwaves* at bottom) from SEPC2006 — `Int-ALU` and `Load/Store` FUs are most frequently used; `FP-ADD` and `FP-Mult` FUs may also be frequently used in FP programs.

Table 5: Detailed FUs in Super-FU block in Fig. 19 from [26]

ALU	ADD, SUB, Logical (XOR, AND, Compare) ...
CINT	IMUL, BitByte (Bit-scan, Bit-test, SET, TEST), Decimal adjust, String (move, compare, scan, repeat), program control, flag control, ...
SSI-I ALU	data transfer, packed arith (add/sub, mul, div, reciprocal, sqrt, max, min), compare, logical, conversion, average, byte mask, dot product, blending, horizontal search, text processing, ATA, ...

Based on this observation, we partition these FUs into three groups. The first group includes four types of FUs: `Int-ALU`, `FP-ADD`, `Int-MUL` and `FP-MUL`. `Int-ALU` has three

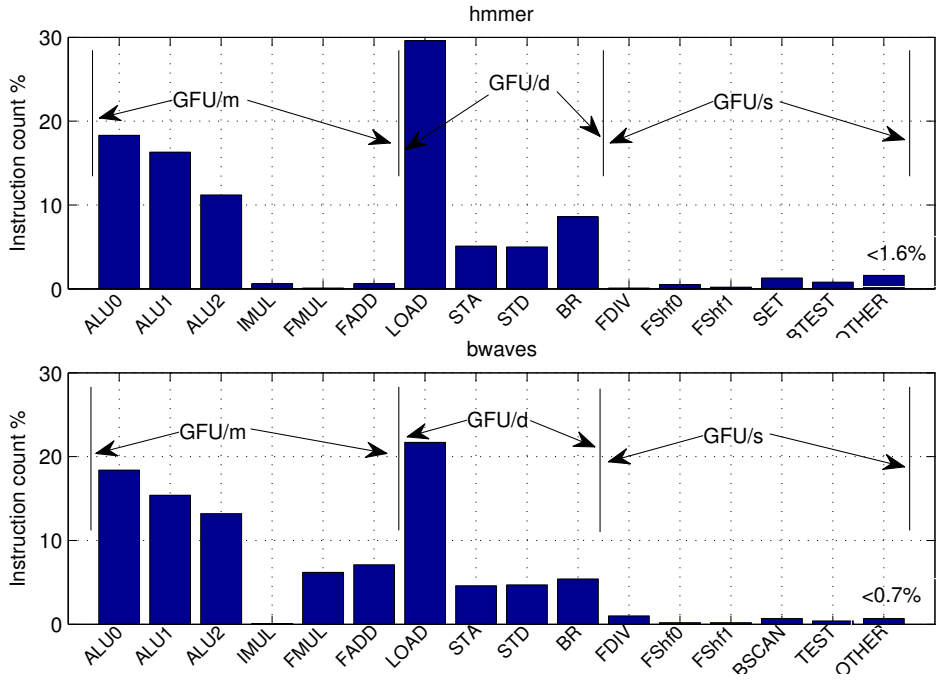


Figure 20: FU usage examples in 3 groups.

copies. Since these FUs are relatively frequently used, any recovery scheme that slows down their execution tends to incur observable performance degradation. We put them in one group because part of their ASIC implementation can be shared. The second group includes frequently-used but SRAM-buffer dominated FUs — `load/store`, `branch`, and `ROB`. Their interface logic reside in the execution unit. For these FUs, SRAM buffer dominates area overhead while accessing buffer dominates timing overhead. The third group includes all other FUs. We isolate and study them at fine-granularity such that most if not all FUs in this group are less frequently used, e.g., the most frequently used FU is `FP-Div` that implements FP division; it has less than 2% usage even in a FP program. In practice, it is also possible to aggregate several small FUs into a super-FU. For example, several complicated ALU operations such as complex masking, bit scan, bit testing, and check speculation require physically different circuits, but can be treated as one `CInt` FU [69] as shown in Table 5. Instead, we treat each physically isolated unit as a fine-grained FU in this thesis. As a result, `bit-scan` has a much low usage than `CInt`.

According to [55], we estimated the area breakdown of these groups is 20%, 5% and 75% respectively of the area of whole execution stage ($5.70mm^2$ in one 45nm Nehalem Core). Here we only consider the interface logic in the second group and ignore the buffer area.

Our generic functional unit (GFU) is designed, by taking advantage of usage and area characteristics of above three groups, to achieve the best trade-off among redundancy, hardware cost, performance impact, and reliability improvement.

- For FUs in the first group, we use **GFU/m**, a macro-based reconfigurable FU design that reuses large logic blocks of `Int-ALU`, `FP-ADD`, `Int-MUL` and `FP-MUL`. GFU/m helps to reduce area overhead and minimize performance degradation when adopting redundancy to improve core reliability.
- For FUs in the second group, we use **GFU/d**, a design that leverages replication-based designs in the literature [67] for the interface logic, and sparing lines or ECC for SRAM buffers [25]. For the area overhead, here we only consider the duplication of the interface logic.
- For FUs in the third group, we use **GFU/s**, a STT-based fully reconfigurable FU design that can be dynamically reconfigured to any FU in this group. GFU/s reduces area overhead by choosing STT over SRAM, gains flexibility by choosing LUT-based design, and incurs small performance overhead due to low usage of FUs in this group.

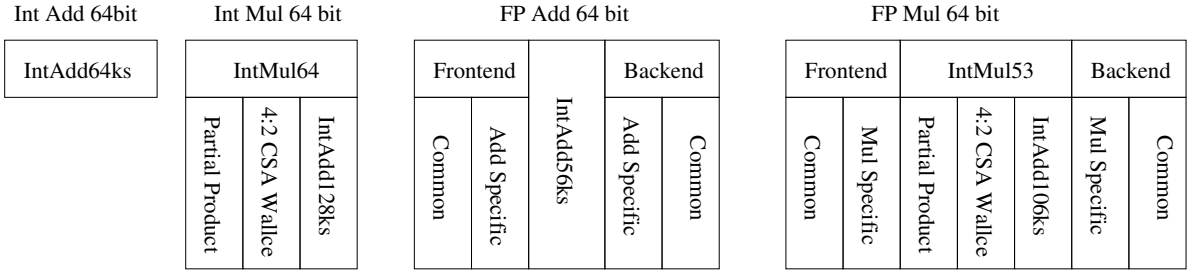
  Next, we discuss the details of each GFU strategy.

5.2.2 GFU/m: A Macro-based GFU

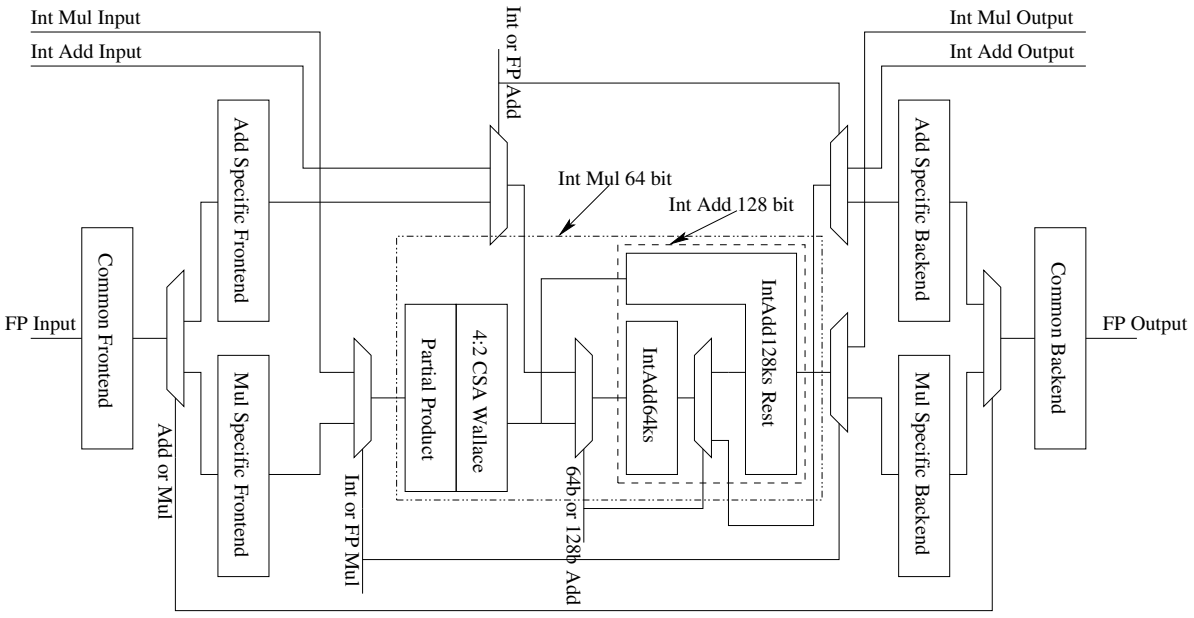
In this section we study the structure of four FUs to be covered by GFU/m: integer adder (Int-Add), integer multiplier (Int-Mul), floating point adder (FP-Add) and floating point multiplier (FP-Mul). Based on the analysis of redundant structures, we use GFU/m that can be configured to any of 4 FUs by reusing redundant logic blocks.

Fig. 21(a) shows the original structure of four FUs. The integer adder (Int-Add) is a widely used and fast 64-bit Kogge-Stone ("ks" in Fig. 21(a)) look-ahead adder that generates carry in 6 ($\log_2 64$) stages. The integer multiplier (Int-Mul) is comprised of "Partial Product", "4:2 CSA Wallace" (4:2 compressor based Wallace Tree Carry Save Adder (CSA))

and final 128 bit Kogge-Stone adder. "Partial Product" generates 64 partial products from multiplicand and multiplier. "4:2 CSA Wallace" computes the carry save sum of 64 partial products in 4 ($log_2(64/4)$) stages. Finally the 128 bit Kogge-Stone adder is used to calculate final product based on the carry save sum. Considering Int-Add and Int-Mul we observe one redundant block, since 64 bit Kogge-Stone adder can be formed by slicing part of 128 bit Kogge-Stone adder.



(a) Structure of original FUs.



(b) Configurable GFU/m structure.

Figure 21: (a) shows the building logic blocks of Int-Add, Int-Mul, FP-Int and FP-Mul. By connecting different blocks, the GFU/m design in (b) could be configured to any of 4 FUs by controlling the (de)multiplexers

When counting FP-Add and FP-Mul in Fig. 21(a), there are more redundant blocks. Both FP-Add and FP-Mul are comprised of front-end, mantissa calculation and back-end. The common front-end units include floating point format unpacking and de-normalized number detection. The remaining part of front-end is operation specific: for FP-Add, there are exponent difference calculation, alignment and swapping; for FP-Mul, there are exponent addiction. The mantissa calculation units are 56 bit integer adder for FP-Add and 53 bit integer multiplier for FP-Mul. In operation specific back-end, there are invert/complement for FP-Add and simple shifting for FP-Mul. And the common back-end units include normalize number, rounding, exception handling and FP format packing. Based on the structure analysis, the common front-end and back-end units could be shared by both FP-Add and FP-Mul. Also, instead of using separate 56 bit integer adder for FP-Add and 53 bit integer multiplier for FP-Mul in mantissa calculation, Int-Add and Int-Mul could be reused. Hence there are considerable redundant logic blocks when counting all four FUs.

Table 6: Control Signals and Added Overhead for Reconfiguring GFU/m.

FU	Add/Mul	Int/FP Mul	Int/FP Add	64b/128b Add	Overhead
Int-Add	NA	NA	Int	64b	4 (de)Mux
Int-Mul	NA	Int	NA	128b	4 (de)Mux
FP-Add	Add	NA	FP	64b	6 (de)Mux + 64b Add
FP-Mul	Mul	FP	NA	128b	6 (de)Mux + 64b Mul

The GFU/m design is motivated by removing the redundant logic blocks and configuring minimum set of logic blocks to form any of four FUs. Fig. 21(b) shows the (de)multiplexer-based configurable design: by properly setting the (de)multiplexers, any of four above FUs could be configured. Table 6 shows the control signals setting for different reconfigurations. For example, when “Int Add” and “64b Add” are selected, the “IntAdd64ks” is configured on the path between “Int Add Input” and “Int Add Output”, forming a 64 bit integer adder. When “FP Mul” and “128b Add” are selected, the front-end, 128bit integer multiplier and back-end form a 64 bit floating point adder.

By incorporating simple (de)multiplexers, the modification of GFU/m on original FU is minimal, while it still incurs overhead. The last column in Table 6 lists factors for added performance overhead. For Int-Add and Int-Mux, only 4 (de)multiplexers are added on the

path. For FP-Add and FP-Mul, besides added 6 (de) multiplexers, larger and potentially slower mantissa calculation units are used: 64b instead of 56b integer adder for FP-Add, and 64b instead of 53b integer multiplier for FP-Mul. However, the evaluation in Section 5.4.2 shows the performance overhead is minimal, as listed as extra cycles in Table 7 for four FUs. Comparing to STT-LUT based GFU/s (discussed later), GFU/m has small performance overhead since it's built on optimized ASIC design. For area overhead, GFU/m is smaller than summing up all individual FUs' area due to removing redundant logic blocks. The result in Section 5.4.2 shows GFU/m is 65% of area of all FUs.

Table 7: The extra latency of GFU/m.

FU	Int-ALU	Int-Mult	FP-Add	FP-Mult
ASIC Latency	1	4	6	8
Extra Latency	+1	+2	+2	+2

5.2.3 GFU/d: A Replication based GFU

GFU/d is a combination of previous proposed recovery schemes. The load/store queue, reorder buffer, and branch predictor are comprised by SRAM-entries and supporting logic. For SRAM-entries, GFU/d adopts designs such as sparing entries [25] or ECC to improve reliability. By reconfiguring the interface logic, the sparing entries are used to replace the faulty entry. Reorder buffers are covered by the simple redundancy and reconfiguration. For the supporting logic, different schemes are used for different structures. In load/store queue logic, there are comparators in the CAM structure that supports simultaneous associative searches to honor memory dependencies and memory consistency models [53]. Similar to fully associative cache, extra entries with companion comparators could be added as backup with acceptable area overhead. For branch predictors, supporting logic such as priority encoders and comparators use structural duplication [67] with 100% area overhead for the logic. The supporting logic duplication does not incur significant area overhead since it takes a small area of the whole structure.

5.2.4 GFU/s: A STT-based Reconfigurable GFU

To recover FUs in the third group, GFU/s utilizes LUT (lookup table) based reconfigurable circuit to replicate a FU on demand. While LUT-reconfiguration is widely adopted in previous FPGA design, the major difference between GFU/s and FPGA-design is that GFU/s is implemented at fine-grained FU level and targets at redirecting the execution of one instruction to the reconfiguration unit. The FPGA design, in contrast, often targets at redirecting a sequence of instructions for performance improvement. As a result, GFU/s incurs performance degradation (although small) but greatly improves reliability.

The motivation of adopting STT-based rather than SRAM-based LUT comes from the area and power advantages that STT provides. Given that STT cell ($25F^2$) is significantly smaller than SRAM cell ($140F^2$), a STT-based LUT is about 25% of SRAM-based LUT [80]. In addition, STT is CMOS compatible such that STT-based LUT can be seamlessly integrated on chip. For the disadvantages, STT write consumes much high power than that of SRAM. However, since reconfiguration of GFU/m happens much less frequently, write power of STT/s during reconfiguration is not a concern.

Replacing one FU using STT-based LUT still incurs large area overhead. Recent study [79] shows that implementing an out-of-order core using SRAM-based LUT takes 17× area of original core in ASIC implementation, and different structures have varying area ratio. Although the area ratio of some function units is between only 4.7× to 7× with supporting DSP and/or memory, we estimate replacing one FU using SRAM-based LUT takes 12× of the original FU size. Then we evaluate the area overhead of STT-based LUT design. Both SRAM-based and STT-based LUT design have two parts, logic (22%) and routing resource (78%). Using STT to replace SRAM cell reduce the logic area by 48% [54]. Routing resource (78%) is comprised of memory (35%) and interconnects+buffers+mux (43%) [44]. Using STT to replace memory (35%) in routing resource reduces 26.25% of total FPGA area. Considering STT replacing the memory both in logic and routing resource, the area of STT-based LUT design is 36.81% smaller than SRAM-based LUT design (12× of ASIC design). In summary, we estimate up to 8× area overhead for GFU/s to replace one FU.

However, compared to the total area of all FUs (at least 25+ and growing) covered by GFU/s, the GFU/s area is acceptable. One potential issue is that if the largest FU is considerably larger than rest FUs, covering it by up to 8× GFU/s incurs intolerable overhead. We set a limit on the GFU/s size to be as large as 35% of total area of all FUs covered by GFU/s. If the GFU/s implementation of some FUs exceed this limit, they are not capable of being covered by GFU/s and we resort to using software emulation to implement the instruction at a larger performance overhead. Implementing all FUs covered by GFU/s and evaluating their area are not likely practical workload, and we would design some sample FUs based on which statistical analysis is performed to evaluate GFU/s coverage.

We estimate the latency of GFU/s is about 2× of original latency (in ASIC design flow) of each individual FU, based on the performance evaluation of various circuit benchmarks implemented in both SRAM-based LUT FPGA and ASIC design flows in [39]. Considering the design space for SRAM-based LUT FPGA is non-trivial, [39] evaluated two representative options: LUT-only design, and the hybrid design built on LUT, memory and DSP. The critical path delay of SRAM-based LUT FPGA is 3.2× (LUT-only) or 2.1× (hybrid) of the delay of corresponding ASIC implementation. Since hybrid design is widely used in current FPGA, we also adopt it GFU/s design. The latency of GFU/s is expected to be smaller than SRAM-based LUT design, due to its reduced size and shorter interconnection. Hence 2× is our conservatively estimation.

5.2.5 Integrating GFU in the Execution Stage

To dynamically replace a FU and redirect instructions using that FU to GFU, we need to integrate GFU in the data-path within the execution stage. As shown in Fig 22, the FUs of our baseline Intel Nehalem-like core architecture [64, 69] are clustered to use six ports of the unified reservation stations (URS). Some FUs such as `Int-ALU` has three copies and is distributed to use port 0, 1, and 5 respectively.

A naive integration of GFU in the data-path is to add another port to URS, which tends to incur unpredictable complexity and cost. Instead, our design introduces a reroute logic that consists of de-multiplexers and a reconfiguration control circuit as shown in the dash-

line box in Fig. 22. When an instruction using the being recovered FU is sent from URS to certain FU cluster via the port, its opcode is compared in parallel by XNOR circuit to control the de-multiplexers for determining whether the instruction is rerouted to GFU, or sent to original FU. For example if ALU0 is detected vulnerable by online self-test and *Conf GFU* decides recovery on ALU0 is needed, *conf0* is set to the opcode that matches that of the instructions running on ALU0. In this way, all instructions that was scheduled to run on ALU0 are rerouted to GFU for execution. If none of FUs in Port0 need recovery, *conf0* is set to a code that does not match any instruction running on FUs in Cluster 0. Due to the space limitation, only configuration signal *conf0* controlling Port 0 is shown in Fig. 22, while the *Conf GFU* generates control signals for other ports. Although the de-multiplexer is added on the critical path, its simple structure adds negligible latency overhead after circuit optimizations such as gate resizing.

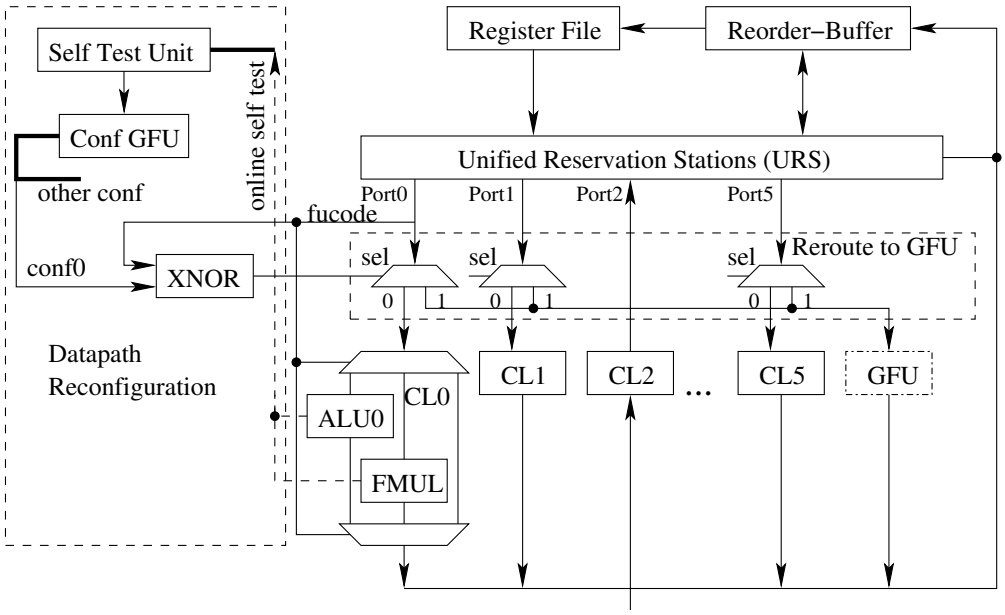


Figure 22: Integrating GFU (rightmost block in dash-dot line box) in the execution stage by incorporating de-multiplexer based reroute logic (in right dash-line box) and configuration logic (in left dash-line box)

5.3 RELIABILITY-AWARE SCHEDULING

For GFU/m and GFU/s, only one FU in each group can be recovered at a time. Therefore we prefer to choose the FU by recovering which we can achieve the largest chip lifetime improvement. On the other hand, naively choosing one FU (e.g., choosing `FP-Mult` when running FP applications) might incur large performance degradation. Therefore, a reliability-aware FU choosing and instruction scheduling scheme is required to find the best trade-off between reliability and performance.

In the thesis, a greedy two-step approach is developed to improve hardware reliability while minimizing performance degradation.

5.3.1 Step 1: Determine the FU to Recover

In the first step, we conduct online monitoring that collects the following information.

- An online self test circuit is integrated to each FU to detect leakage current (due to TDDB effect) and/or timing margin (due to BTI effect). For the former, [16] proposed to track TDDB stress using a simple I_{ddq} testing circuit. For the latter, previous studies have proposed the adoption of a series of delay buffers to capture the timing margin left in each FU [42, 7].

  Note that the timing circuit is only added to the FUs whose BTI stress may fail the FU. For a simple logic whose slowdown shall not affect cycle time, there is no need to track its BTI effect. On the other hand, the testing circuit for TDDB is required per FU.
- Several sampling counters are added into the instruction scheduling unit such that they can record the usage information in the past interval.

Based on the collected run-time information, we develop a simple heuristic to choose the FU that has the largest value as follows.

$$MAX_{\forall FUi}\left(\frac{M_i}{f\left(\frac{D_i+d_i}{D_i} \times U_i\right)}\right) \tag{5.1}$$

where M_i indicates the timing and leakage current margin left for FU_i; U_i indicates the usage in percentage for the past sampling interval; D_i and d_i are the original latency of FU_i and

the extra delay due to adopting GFU. Function f(x) normalizes the usage to balance the weight of reliability and performance degradation. Different performance degradation values are normalized closer if we weight more on reliability improvement.

When a functional unit $\texttt{FU}_x$ is chosen in the first step, we can reconfigure GFU/m (or GFU/s) to recover $\texttt{FU}_x$, redirect instructions that need $\texttt{FU}_x$ to GFU/m (or GFU/s) accordingly, and then power off the original circuit of $\texttt{FU}_x$ in the core. Since reconfiguration GFU/s can take tens of milliseconds or longer, it should not be done frequently. In our design, after choosing one FU, GFU/m (or GFU/s) will be reconfigured to be a copy of that FU for at least one hour.

When choosing a FU to recover, we only focus on all original FU, excluding GFU/m and GFU/s. Shutting down GFU/m and GFU/s for recovery is same as not adding GFU/m and GFU/s at all. Since GFU is added to improve reliability of original FUs, we assume we always use it.

5.3.2 Step 2: Minimize Performance Degradation

After having determined the appropriate FU to recover, we focus on minimizing performance degradation in the second step. Step 1 actually estimates the performance degradation but it is done very conservatively. In particular, it assumed that each instruction redirection degrades the overall performance the same. This is often too conservative as we will elaborate.

5.3.2.1 Handling GFU/s based Recovery Due to different usage frequency and latency overhead, GFU/m and GFU/s are utilized differently in recovering FU. For GFU/s-based recovery, the unit under recovery is powered off for natural recovery and requesting instructions are redirected to GFU/s. Although GFU/s unit almost doubles latency, due to low usage of the replaced FU, the performance impact is small to negligible. The reason that FU in the third group has low usage is based on our usage-aware grouping that puts low usage FUs into this group, and also our FU isolation that recognizes FU at fine granularity.

In extreme cases that a FU in this group has extremely high usage, e.g., an attacking/buggy program uses a vulnerable FU `FP-Add` or `FP-Mul` intensively, there are two op-

tions: 1) leaving it to the OS to reschedule it to another core, or 2) staying but using the instruction steering design as described next.

5.3.2.2 Handling GFU/m based Recovery

For the FUs in the GFU/m group (Section 5.2.2), they have relatively high usage. Even though we use fast GFU/m to recover and there may be multiple copies, we still have observable performance degradation. Such performance degradation might be a big issue when there are tight constraints on performance degradation, such as in real-time computing. In this Section we discuss how to further reduce performance overhead by incorporating slower GFU/m, depending on whether the FU has multiple copies or not. If the FU is unique [1] and its use is quite high in some phases, we use *Switch to Original FU* to tradeoff some recovery opportunity for less performance loss. If the FU has more than one copies, we use *Instruction Steering* technique to identify critical instructions and steer them to original FUs, only leaving uncritical instructions running on slower GFU/m to reduce performance overhead.

Switch to Original FU

Switch to Original FU handles the core case that FU under recovery is frequently used and considerate performance loss is observed. *Switch to Original FU* determines whether to switch back from slower GFU/m to fast original FU under recovery so that performance loss is avoided at the cost of compromised reliability improvement. Although the GFU/m configuration is fast (controlling (de)multiplexers, powering on/off original FU and stabilizing the voltage takes up to hundreds of cycles. Hence we limit epochs (minimum switching interval) in coarse level (seconds) to prevent frequent switching between original FU and GFU/m. If in the past epoch, the sampled number of critical uses issued to GFU/m is above a threshold, then it powers on the FU being recovered for the next epoch, and use it for all instructions. When the FU being recovered is powered on, it loses the natural recovery for one entire epoch. Therefore, choosing the proper threshold illustrates a trade-off between performance degradation and reliability improvement. However, we expect unique FU configuration is less likely for GFU/m group, since it is logical to use multiple copies

[1] FP-Mul and FP-Add in Intel Nehelem are unique. However in [20], the Intel Nehelem core has 2 FP-Add/Mul. Also, in IBM Power 7 there are 2 FP FUs.

for frequently used FU (which falls into GFU/m group) when there are more transistors in future technology generations. [20] use two FP-Add/Mul FUs in the Intel Nehalem core modeling and IBM Power 7 processor has two FP FUs.

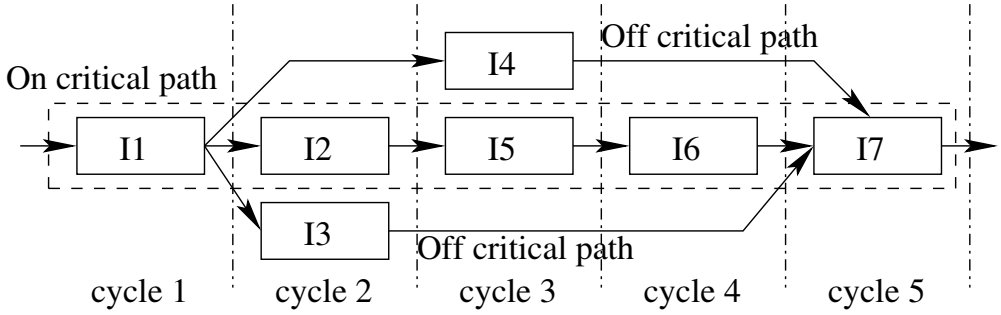


Figure 23: A simple instruction dependency graph. Both `I1` and `I4` use a specific FU. However, `I1` is the critical use while `I4` is not.

Instruction steering (IS)

*I*nstruction steering (IS) is developed based on the observation that not all instructions executed on a specific FU are of the same impact on slowing down performance. If the instruction is not on critical path, running it on a slower FU has no performance overhead. Fig 23 illustrates a simple instruction dependency graph. Since instruction `I4` does not reside on the critical execution path, executing it on a slower FU would not generate performance degradation. On the other hand, instructions `I1, I2, I5, I6, I7` are critical, i.e., slowing down their execution may slow down the program. By identifying the critical instructions and steering them to original FUs, the performance overhead of using slower GFU/m is further reduced. Note that *Instruction Steering* is only feasible when there are multiple copies of FU, which is probable for FUs in GFU/m.

Instruction Steering relies on lightweight and accurate criticality analysis at run-time. We find the concepts of instruction slack and criticality proposed in [13] useful in illustrating how the slower function unit affects the program performance. The results are promising: doubling the execution latency (FU running at half speed) or delaying 5 cycles of greater than 75% instructions on average for most applications do not affect the performance in execution stage due to sufficient slacks. However, it is also observed that there is no instruction type

which nearly always has slack and it's necessary to predict criticality. Hence [13] proposes the history-based predictor to answer the following question "for a dynamic instruction i, does it have n cycles of slack (where n is the extra latency introduced by GFU/m)?". In this section we propose a light-weight heuristic predictor to support *Instruction Steering*.

Lightweight critical-use detector.

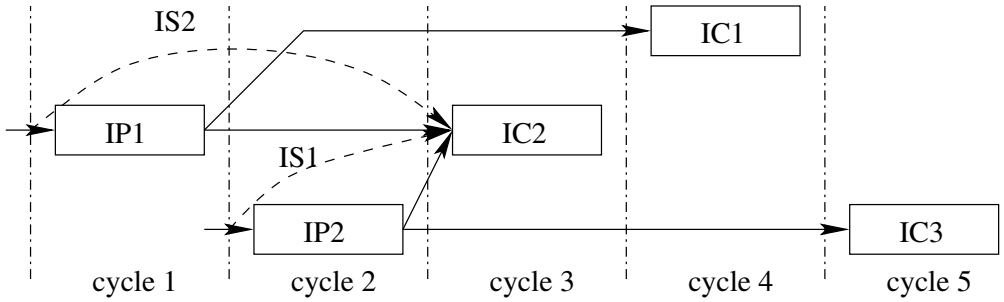


Figure 24: Issue slack is the time slack from the cycle when producer instruction finishes to the cycle when its first consumer starts execution: the issue slack of IP1 (producer 1) is 2 (to IC2, consumer 2); and the issue slack of IP2 (producer 2) is 1 (to IC2, consumer 2)

An intuitive perspective is used to extract slacks other than the expensive delay-and-observe approach in [13] and a simple heuristic is proposed to detect critical instructions with small slacks. We introduce two concepts, issue slack (IS) and commit slack (CS). The IS of one instruction is the time slack from the cycle when the instruction finishes execution to the cycle when its first consumer instruction starts execution. In Fig 24, the producer instruction IP1 has two consumers IC2 (first) and IC1 (second), and the IS of IP1 is 2 (from cycle 1 to cycle 3). Similarly, the IS of producer instruction IP2 is 1. We classify instructions to 2 groups: IS1 instruction with 1 cycle issue slack, and IS2+ with more than 1 cycle issue slacks. If IS2+ instructions (like IP1) run on slower GFU (+1 cycle), its consumers aren't delayed since its IS is larger than 1 which could tolerate the extra cycle induced by slower GFU. On the other hand, if IS1 instructions (like IP2) run on slower GFU (+1 cycle), its first consumer (IC2 for IP2) is delayed for one cycle. Hence, delaying IS1 (like IP2) instructions delay its consumers and may affect overall performance. Although delaying IS2+ (like IP1) instructions does not affect consumers, it may indirectly blocks commit queues.

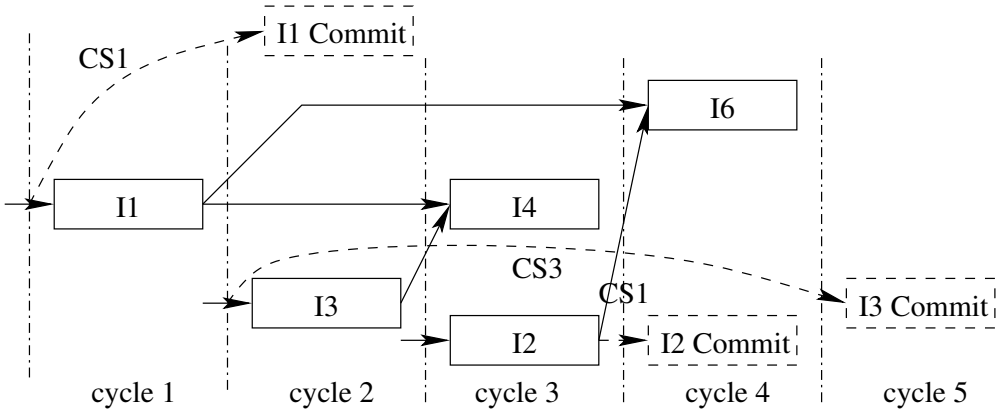


Figure 25: Commit slack is the time slack from the cycle when instruction is ready to commit to the cycle when its actually commits: I1 commits 1 cycle after it's ready, and its CS is 1; I3 commits 3 cycles after it's ready, and its CS is 3.

Similarly, the CS is defined as the time slack from the cycle when the instruction is ready to commit and placed in commit queue to the cycle when the instruction is actually committed. In Fig 25, I3 has three cycle commit slack (CS2+) as it is ready to commit at cycle 2 and actually commits at cycle 5. Similarly I1 has one cycle slack (CS1). Due to in-order commit, I3 cannot commit early as another instruction I2 (ahead of I3 in ROB) is slow. Such an instruction tends to block more instructions (not just I2) such that I3 at issuing time would see more instructions in the ROB. On the other hand, I1 does not wait for anyone such that I1 at issuing time should see less instructions in the ROB. Slowing I1 may quickly accumulate entries in ROB and slows the execution. However, slowing I2 tends to have local impact such as slowing down I4 but it cannot commit anyway.

Classifying instructions to IS1, IS2+, CS1 and CS2+ provides useful clues to detecting critical instructions: CS2+ and IS2+ instructions are off critical path; either IS1 or CS1 instructions are candidates on critical path. Our simulation results on replacing ALU2 by GFU/m show that IS1 instructions take up 10% to 15% of, and CS1 instructions take up 10% to 25% (INT benchmarks) and 1% to 15% (FP benchmarks) of all instructions on one ALU. This confirms large portion of instructions are off critical path [13]. Further study

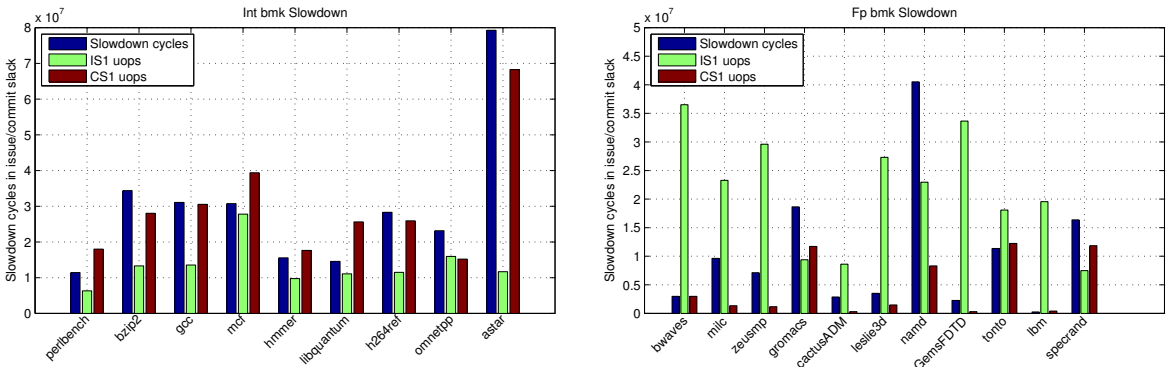


Figure 26: Comparisons of slowdown cycles when ALU2 is replaced by GFU/m with 1 extra cycle latency, and CS1 and IS1 instructions numbers show strong correlation of slowdown cycles and number of CS1 instructions in both INT and FP benchmarks.

shows that CS1 instructions are more likely to be critical than IS1 instructions. In Fig. 26, the raw number of CS1 and IS1 instructions in baseline processor core are compared to extra cycles when ALU2 is replaced by GFU with one extra cycle latency in INT benchmarks. CS1 instructions are generally more than IS1 instructions (especially in perlbench, gcc, h264ref, astar), and it shows a strong correlation with CS1 instructions and slowdown cycles. The results are similar in FP benchmarks: although there are much more IS1 instructions than CS1 instructions, the slowdown cycles are generally closer to number of CS1 instructions. The fact CS1 instructions are more critical than IS1 instructions are intuitive: 1) CS1 instructions directly blocks the in-order commit queue if slowed down; 2) consumers of IS1 instructions may not be critical instructions. However, portion of IS1 instructions are also critical when their consumers (for example, load, and other long latency instructions) are critical, as observed in [13].

A simple heuristic for detecting CS1 instructions is used based on the observation that the distance to reorder buffer (ROB) tail (where commit happens) strongly correlates with whether the instruction is CS1 or not: 90% CS1 instructions are within 13 entries range from ROB tail in almost all INT and FP benchmarks. Fig 27 and Fig 28 show the accumulated instruction count versus distance to ROB tail for CS1 (square symbol) and CS2+ (circle

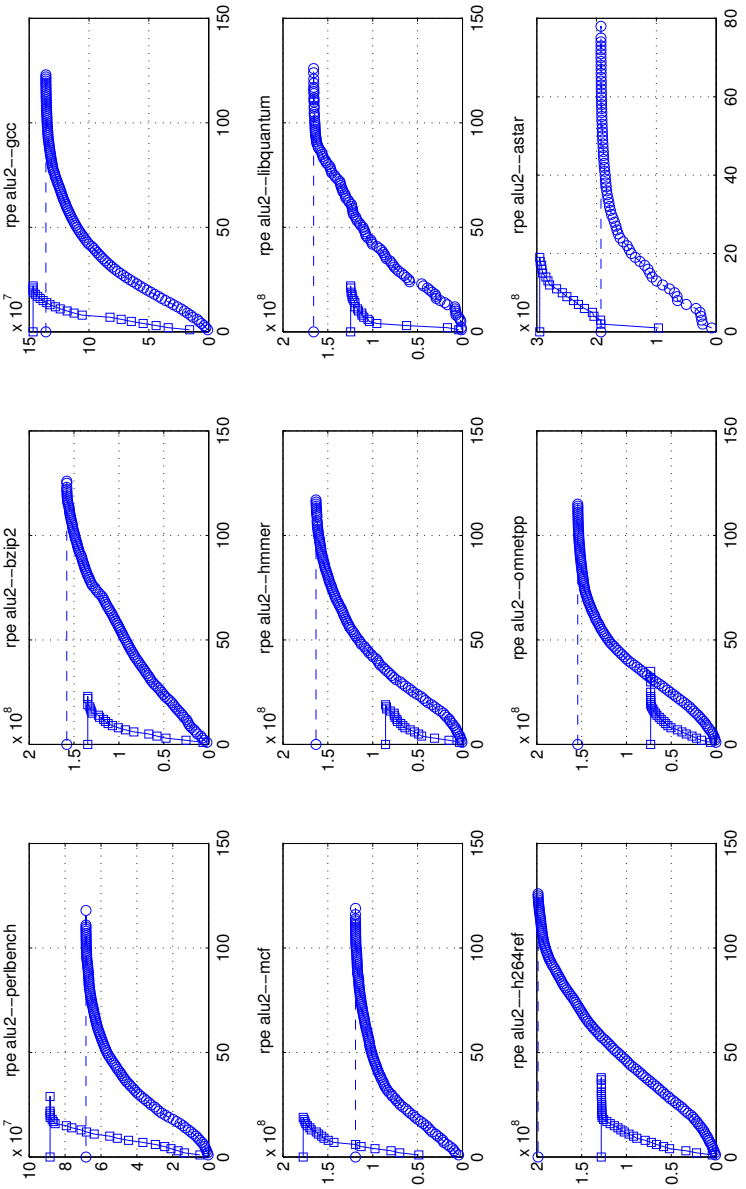


Figure 27: CS1/CS2+ instruction count (accumulated count in y axis) and distance to ROB tail (x axis) in ALU2 with integer benchmarks show most CS1 instructions (in square symbol) are generally close to ROB tail while the CS2+ instructions have variable distance to ROB tail.

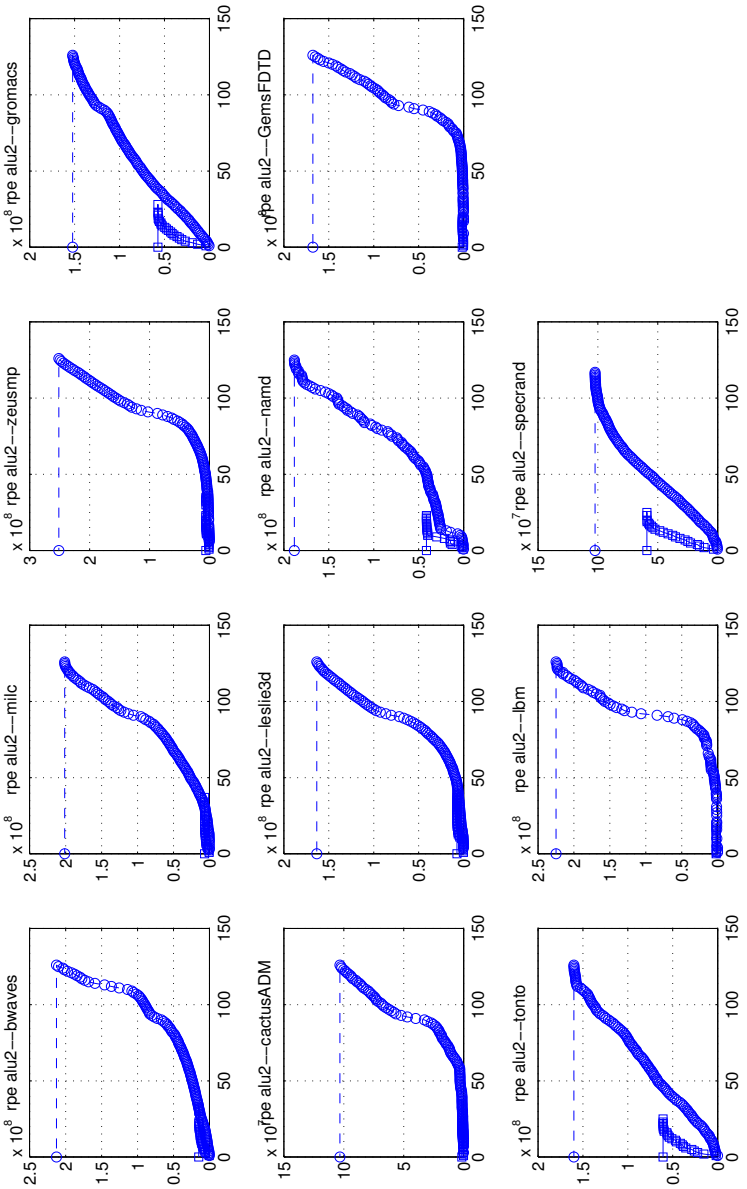


Figure 28: CS1/CS2+ instruction count (accumulated count in y axis) and distance to ROB tail (x axis) in ALU2 with floating point benchmarks show CS1 instructions (in square symbol) are less and generally close to ROB tail while CS2+ instructions have variable distance to ROB tail.

symbol) in ALU2 with SPEC2006 integer and floating point benchmarks. In integer benchmarks, most CS1 instructions are close to ROB tail. We set the threshold of distance to ROB tail such that 90% CS1 instructions are included. Although the threshold varies in different benchmarks, we conservatively set the threshold to be 13. One drawback of the simple heuristic is mispredicting some CS2+ instructions to be CS1 instructions and causes unnecessary instruction steering. This misprediction is a minor problem since 1) in integer benchmarks the misprediction rate is relatively low and 2) in floating point benchmarks although the misprediction rate is high, the absolute count of mispredicted CS1 instruction is small. This heuristic is also intuitive: the further the instruction is from ROB head when issued, the less likely it is actually committed soon after ready to commit. The distance to ROB tail could be simply tracked in existing issue logic and is easy to implement. Comparing to the more generalized slack predictor approach[13], it does not need additional hardware and thus has lower hardware overhead. The results show that scheduling detected CS1 instructions to original FUs reduce the performance overhead, as shown shortly in Section 5.5.3.

- Whether there are some more effective heuristics in detecting CS1 instructions. For example, instead of using the number of instructions between the current instruction and re-order buffer queue tail, a potentially more accurate metric would be accumulated execution latency of the instructions between the current instruction and the re-order buffer queue tail.
- Whether some IS1 instructions are also critical. We would utilize the history-based predictor to help identify the critical IS1 instructions. If a large portion of IS1 instructions are critical in some benchmarks, we would explore heuristics to detect them. For example, if its consumer is critical or it has many consumers, IS1 instruction may be critical.
- Considering both IS1 and CS1 instructions may be critical, we would hybrid heuristics to detect both critical IS1 and CS1 instructions.
- Compare our heuristic based critical instruction predictor to the history-based predictor in terms of accuracy and implementation overhead. We would also use history-based predictor to predict IS1 and CS1 instructions and evaluate performance when IS1 and CS1 instructions are detected and saved for original FU.

- Besides evaluating the heuristic based predictor on ALU, more evaluations are need on other FUs, such as FP-Add and FP-Mul. For FP-Add and FP-Mul that have extra 2 cycles latency, we would also consider CS2 and IS2 instructions in addiction to CS1 and IS1 instructions.

5.3.3 Discussion on OS/Compiler reliability scheduler.

Different from using hardware scheduler to determine when and how to engage GFU to replace vulnerable FUs, an alternative is using OS or compiler level techniques to offload heavy activity on vulnerable FUs. For example, OS may put processes with more idle cycles (usually due to frequent cache misses) on vulnerable cores. Also compiler may schedule less add instructions on vulnerable adders.

The OS reliability scheduler is a good complementary approach to hardware scheduler. This is very useful when different processes exercise different FUs, such as mixing INT and FP processes. However, the compiler approach is less practical since optimized compilations processors' vulnerable FUs are varied and changing overtime

5.4 EXPERIMENTAL METHODOLOGY

5.4.1 Architecture Modeling

An Intel Nehalem-like out-of-order core is modeled with parameters in Table 8. Our simulation were performed using a cycle accurate simulator PTLsim [56]. The execution stage area in one core is $5.72mm^2$ (2.54mm x 2.25mm), estimated based on the floor-plan in [36] that has Instruction Decode, Instruction Fetch & L1 I-Cache, L1 D-Cache, TLB, Branch Prediction, L2 D-Cache, Out-of-Order Scheduling & Retirement, Memory Ordering & Execution. Execution stage is the largest component in core except for L3 cache. It is also dominated by combinational logic. SPEC2006 CPU benchmark suite (9 integer and 11 floating point benchmarks) is used to evaluate the performance overhead of our GFU design. The warm up phase is skipped and 1 billion instructions are simulated to compute the IPC.

Table 8: Simulation configuration.

Out-of-order Execution
4-wide fetch/commit, 6-wide issue, 128 ROB, 32 FQ, 48 IQ, 48LQ, 32SQ, 96 Int-PRF, 64 FP-PRF
Memory Hierarchy
L1 I/D-Cache: 32KB, 4/8 way, 64B line size, 3-cycle L2 Cache: 256KB, 8 way, 64B line size, 10-cycle L3 Cache: 8MB, 16 way, 64B line size, 30-cycle
Benchmarks
INT: perlbench, bzip2, gcc, mcf, hmmer, libquantum, h264f, omnetpp, astar FP: bwaves, milc, zeusmp, gromac, cactuasADM, leslie3d, namd, GemsFDTD, tonto, lbm, specrand

5.4.2 GFU/m Circuit Modeling

Four individual FUs (Int-Add, Int-Mul, FP-Add and FP-Mul) and GFU/m are modeled in VHDL files according to the design in Section 5.2.2. Structural style design is used in implementing the 64 bit Int-Add and Int-Mul (double precision). In FP-Add and FP-Mul, the front-end and back-end units are implemented from *fpu_double* design from opencores [52]. Since the essential mantissa calculation in *fpu_double* uses simple behavioral design that generate inferior net-list, we replace it with structural design of 56 bit Kogge-Stone integer adder for FP-Add and 53 bit integer multiplier for FP-Mul. The GFU/m design follows Fig. 21(b) that is comprised of 64 bit Kogge-Stone integer adder, 64 bit integer multiplier, front-end and back-end unit and (de)multiplexers. All VHDL design files are synthesized in Design Compiler [27] and mapped to 45nm FreePDK technology library [15] for area and delay evaluation that is shown in Section 5.5.1. The layout and its evaluation on area, delay and power is scheduled for future work.

5.4.3 Reliability Modeling

Two sources, manufacturing time Process Variation (PV) and in-field stress contribute to wear-out that is represented as diminishing guard-band. The guard-band is the margin

between the specified metric and real metric (for example, delay, or frequency). Guard-band is widely used in manufacturing such that during the warranty period the chip's real metric degrades while still within the margin to the specified metric. Due to PV, the guard-band of each function unit is different. Under the same stress, the function unit with less guard-band is more vulnerable. We used VARIUS [62] to generate voltage threshold (V_{th}) and effective length (L_{eff}) samples and then modeled the critical path delay each function unit. The guard-band variations were also calculated accordingly. The in-field aging mechanisms, BTI and TDDB, follow the models presented earlier in Chapter 2.

5.5 EXPERIMENTAL RESULTS

5.5.1 Hardware Overhead

Monitoring BTI and TDDB introduced stress at run-time is required to determine the candidate unit under recovery: the most vulnerable FUs. The timing monitor circuit (for BTI) consists of a series of 12 buffers and latches per FU, and share the pulse generation, delay calibration and encoder circuit among all FUs [16]. The I_{ddq} test circuit (for TDDB) requires three transistors per FU. In total, the hardware overhead for testing TDDB and BTI stress is below 1% of total area of the execution stage.

Table 9: GFU/m Hardware Area/Delay Evaluation

FU	Int-Add	Int-Mul	FP-Add	FP-Mul	GFU/m with (de)mux
Delay(ns)	1.4	4.32	7.21	10.98	+0.04/+0.06/+0.22/+0.5
cycle	1	4	6	8	+1/+1/+2/+2(conservative)
area(μm^2)	2796	88185	33133	106925	122139

The delay, cycle number and area of GFU/m design are listed in Table 9, collected from Design Compiler reports. The cycle time is set at 1.4ns such that Int-Add finishes in one cycle [2]. The delay overhead of GFU/m when configured to Int-Add or Int-Mul is 0.06ns and 0.09ns. This is considered within 1 cycle, since configured (de)multiplexers behave as

[2]Int-Add in commercial processors are faster than 1.4ns due to heavy optimizations.

added resistance on the conducting path and the effect on delay is negligible. The delay overhead of GFU/m when configured to FP-Add and FP-Mul is 0.24ns and 0.53ns, due to the over-design of mantissa calculation by reusing Int-Add and Int-Mul. Although the delay overhead is still within 1 cycle, we conservatively assume it's within 2 cycles with extra wiring overhead. For the area overhead, the GFU/m is 52.9% of total area of all 4 FUs due to removing redundant logic blocks. The (de)multiplexers takes 1.82% of GFU/m area.

For GFU/d, the area overhead is 100%. However, GFU/d only serves as the interface to Load Store Queue and it only takes 5% of total execution unit area. There is negligible latency and power overhead caused by multiplexers while the function unit itself is identical to the original one.

For GFU/s, as discussed in Section 5.2.4, due to the uncertainty of individual function units' area, the limit of 35% area overhead for GFU/s is imposed. Since 35% area of GFU/s may not cover some large FU implementation and incur non-perfect coverage, coverage analysis is performed based on the implementation and evaluation on 3 sample function units: integer divider, CRC and AES generator for DIV, CRS and AES instructions. The 3 function units are chosen due to their complexity and large area to implement thus provide an upper bound estimation of GFU/s implementation.

The integer divider design is an implementation of one 53 bit Radix-4 SRT divider (widely used in commercial processors) generated by tool divgen [47] that is to be used in double precision floating point divider. The Radix-4 SRT design generates 2 bits of quotient every cycle and the latency is 27 cycles. The CRC hardware function unit calculates the Cyclic Redundancy Code (CRC) to check errors in data transmitted over unreliable medium such as network. The CRC generator design [46]'s core component is the Galois Field Multiplier that is to be mapped to a netlist of XORs up to 10 levels. The 128 bit key AES cipher [71] calculates the encrypted code using the 128 bit key to protect sensitive data. The core components are key expanding unit and 3 permutation units. The operation takes 12 cycles to finish. All three FUs are implemented in ASIC and FPGA work flows in to compare the area and latency. The ASIC work flow is same as that in Section 5.4.2. The FPGA work flow uses the Xilinx ISE 10.1 with target Virtex-5 chip.

Table 10: Evaluation of DIV, CRC and AES

Design	ASIC	FPGA (SRAM-LUT)	STT-LUT
DIV	$6605um^2$	$29723um^2$(666 LUT)	$20806um^2$)
CRC	$8212um^2$	$29467um^2$(531 LUT)	$20627um^2$
AES	$56417um^2$	$264550um^2$(694 LUT)	$185185um^2$
DIV	27 cycles	80 cycles	57 cycles
CRC	10 cycles	35 cycles	24 cycles
AES	12 cycles	28 cycles	22 cycles
DIV	1.39mW/0.46uW	7.48mW/2mW	2.41mW/179uW
CRC	803uW/82uW	5.67mW/0.8mW	1.53mW/112uW
AES	1.654mW/273uW	10.52mW/2.7mW	5.32mW/234uW

The area, power and latency are summarized in Table 10. The results for the STT-LUT is based on the estimation in Section 5.2.4.

Although the area of GFU/s covered units are not random in a certain design (for example in Nehalem), due to limited access to accurate units area number, it's assumed the units area is randomly distributed in different designs, considering varying trade-off decisions made in architecture and RTL level. For example, some design implements complex and large AES unit (in recent IBM Power7 micro-processor), while another design may use simple and small units to carry multiple micro-instructions to implement complex function. As a proof of concept, two representative distributions are shown in Fig.: 1)optimistic (area is small in most units), and 2) pessimistic (area is large in most units).

5.5.2 GFU-introduced Performance Degradation

Then the performance degradation is studied when slower GFU replaces original FU that is under recovery. Fig. 29 shows normalized IPC of measured benchmarks when `FP-Add`, `FP-Mult`, or `FP-Div` is replaced by GFU/m and GFU/s individually. When replacing FP-

Add or FP-Mul, GFU/m is 2 cycles slower than original FU. The results are labeled "fadd +2" and "fmul+2". When replacing FP-Div, GFU/s is 20 cycles slower than original FU and its results are labeled "fdiv+20". Fig. 30 shows the normalized IPC when `ALU-2` (in Port 5) or `ALU-0` (in Port 0) is replaced by GFU/m individually. Since GFU/m is 1 cycle slower than `ALU-2` and `ALU-0`, the results are labeled "alu2+1" and "alu0+1". In these experiments, we did not perform *instruction steering*. That is, the slowdown indicates the raw performance impact due to fine-grained adoption of slow GFU.

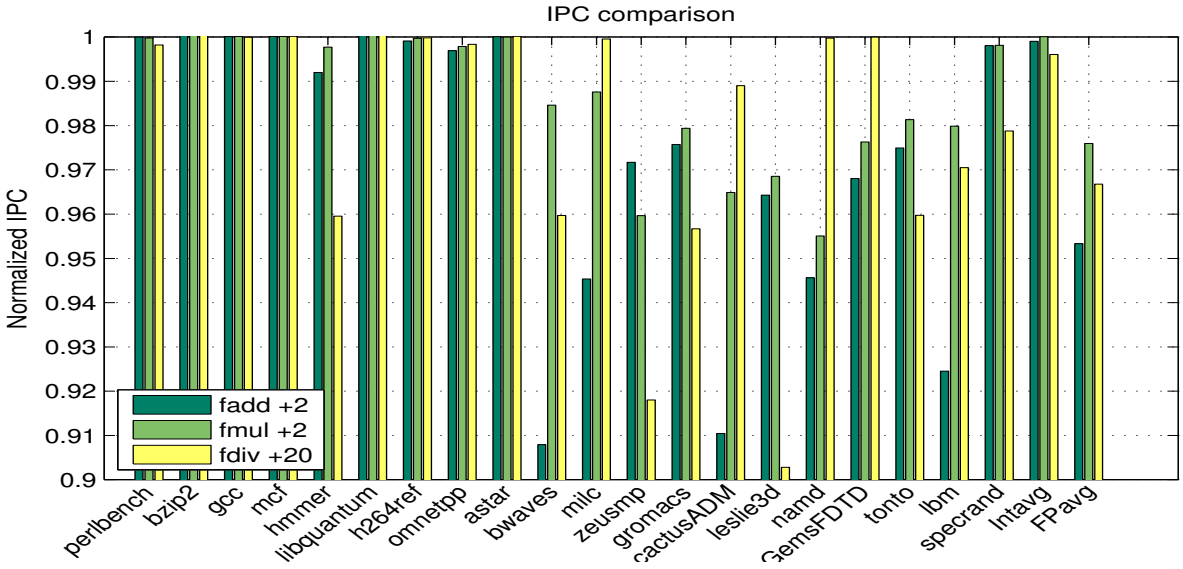


Figure 29: Normalized IPC when FP Add/Mul/Div gets replaced respectively.

From Fig. 29, there are 2 interesting observations:

- Most INT benchmarks are immune to floating point FU latency overhead, even the FP-Div in GFU/s is 20 cycles slower. The only exception is "hmmer".
- All FP benchmarks are affected by floating point FU latency overhead. Most benchmarks are sensitive to FP Adder, FP Multiplier and FP Divider in descending order. Some exceptions are "zeusmp", "leslie3d". In some extreme cases, the IPC loss is as high as 9%, such as FP Adder on "bwaves" and "cactusADM", FP Divider on "leslie3d". On average, the FP Add/Mul/Div slowdown cause total IPC loss of 4.6/2.3/3.2%.

The above observations are closely related to the FU usage. In INT benchmarks where floating point FUs are lightly used, replacing them with slower GFU causes negligible per-

formance loss. All FU usage statistics are collected in PTLsim simulator and that of two representative benchmarks from INT (hmmer) and FP (bwaves) groups are shown in Fig. 20. Floating point FUs in most INT benchmarks except for "hmmer" are almost never used but heavily used in "cactusADM". In "hmmer" floating point FU slowdown is explained by minor FU usage.

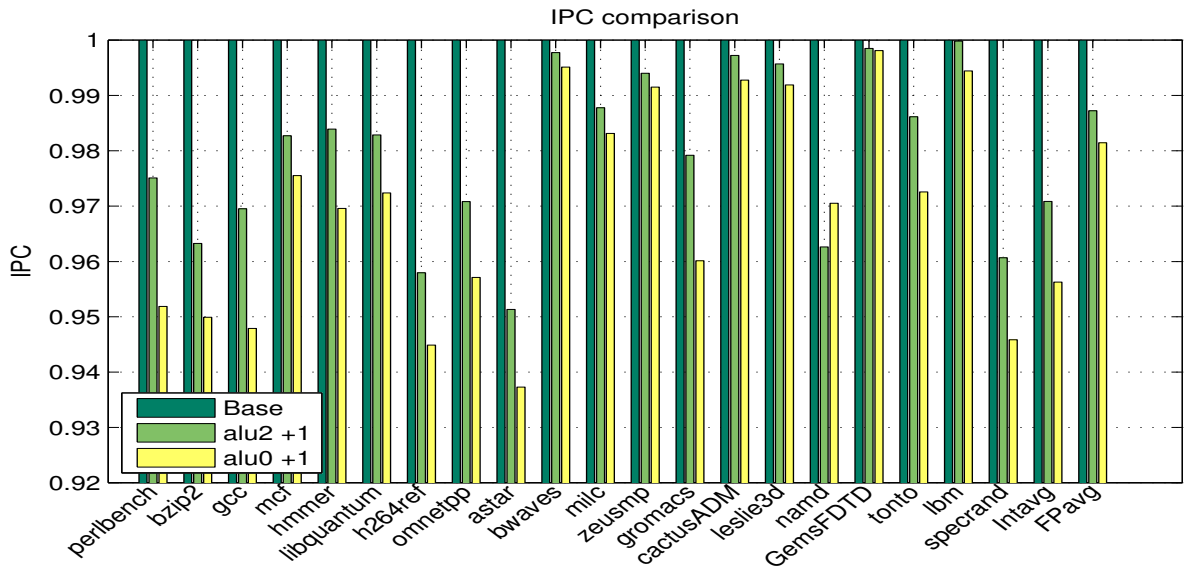


Figure 30: Normalized IPC when ALU2/0 gets replaced respectively.

Next, we study replacing ALU-0 and ALU-2 that are frequently used FUs in all benchmarks, with slower GFU. In Fig. 30, on average the IPC loss for ALU-2 in INT benchmarks is 2.9% and that in FP benchmarks is 1.3%. We also get some observations from Fig. 30:

- When GFU is used to replace ALU-0, the performance overhead is slightly higher than the case for ALU-2. This is explained by the fact that ALU0 is generally used slightly more than ALU2 [3], shown in Fig 20.
- The variation of performance overhead in INT benchmarks is large. Benchmarks "mcf", "libquantum" and "hmmer" are less sensitive to ALU performance loss. Further study shows they have higher L1 miss rate such that memory accesses take larger portion of total execution time. As a result, the performance loss is partially hidden in long

[3]Caused by slightly biased ALU usage in default scheduler in PTLsim simulator

latency memory access and their IPC is less sensitive to ALU latency overhead. On the contrary, "astar" and "h264f" are CPU intensive benchmarks and ALU operations take larger portion of critical path. Slowing down ALU in those benchmarks incurs larger performance overhead

- Most FP benchmarks are less sensitive to ALU0/2 latency overhead, such as "bwaves", "cactusADM", "GemsFDTD", etc. This is due to that in FP benchmarks the critical path is comprised of many long latency floating point instructions, by which the latency overhead of ALU is hidden. The effect of long latency floating point instructions is similar to that of memory accesses in INT benchmarks.

The aforementioned study shows that IPC loss is acceptable if replacing original FU with GFU/m. Similarly the IPC overhead is also acceptable when GFU/s is used. As shown in Fig. 29, when FP-Div is replaced by GFU/s that is 20 cycles slower than original FUs, the performance loss is just 0.04% for INT benchmarks and 3.2% for FP benchmarks.

From brief study we learned that IPC loss is sensitive to FU usage and the existence of long latency instructions on other FUs. The long latency instructions increase program execution time and reduces the effective FU usage time. Thus, by replacing lightly used FU with our GFU, the performance overhead could be well controlled. The FUs' usage in processor core are different in different benchmarks and even in difference phases of one benchmark. Our study shows obvious difference in INT and FP benchmarks: floating point FUs in INT benchmarks are lightly used and integer FUs such as ALUs are used less in FP benchmarks than in INT benchmarks. If there is no dedicated GFU/m to recover, we choose one FU to recover such that performance overhead is minimized by monitoring its activity.

Besides using a slower 3rd ALU for replacement, another option to recovery third ALU is to shut it down. We evaluate the performance impact of disabling one of the 3 ALUs in processor pipeline. The results are shown in Fig 31. In each group of the bars, the first bar shows the base case where all 3 ALUs are available, and the second and third bars show the normalized IPC when ALU0 or ALU1 is slower. The forth bar shows the average normalized IPC when one ALU is disabled and only 2 ALUs are available. For integer benchmarks, the IPC loss on average is almost 9%. For some applications with higher ALU usage such as "astar" and "h264ref" the IPC loss is 14% and 12%. For floating point benchmarks, the IPC

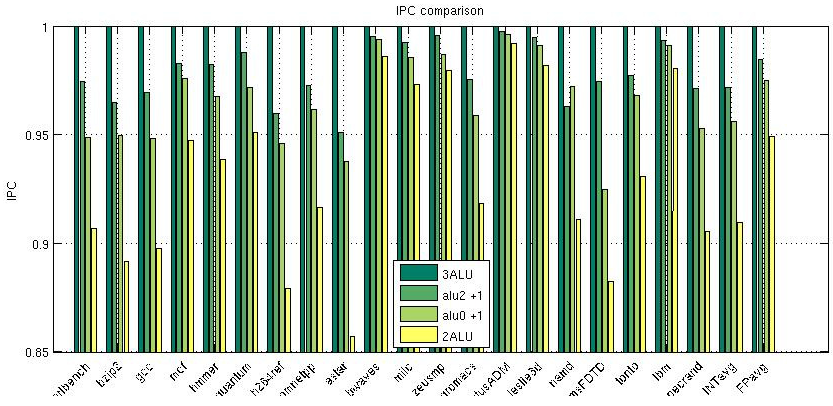


Figure 31: 2 ALUs vs 3 ALUs

loss is mildly at 5% on average. However, for some extreme cases the IPC loss is approaching 10% (namd, GemsFDTD, gromacs). The results show disabling one ALU causes significant IPC loss in general cases and is considered less effective than our design.

5.5.3 Instruction Steering

The instruction steering is based on CS1 instruction prediction. The predicted non-CS1 instruction are scheduled to GFU, while the predicted CS1 instructions are scheduled to original FU. However, this simple instruction steering could not recover all performance overhead, due to non-perfect CS1 prediction rate and minor IS1 impact on performance. Fig. 32 shows result of applying CS1-prediction based instruction steering. In ideal case the performance overhead is reduced from 2.9% to 1.4% in INT benchmarks and from 1.25% to 0.85% in FP benchmarks. For ideal cases we assume when we decide to steer predicted CS1 instructions from GFU/m to original FU, the original FU is always available. In reality this assumption does not always hold. If no original FU is available, the predicted CS1 have to run on slower GRU. The third bar "alu2+1 cs1 Real" show the results in reality is very close to the ideal case. Hence our CS1-prediction based instruction steering is very simple and effective in further reducing performance overhead when replacing heavily used intrinsic redundant FUs with our GFU.

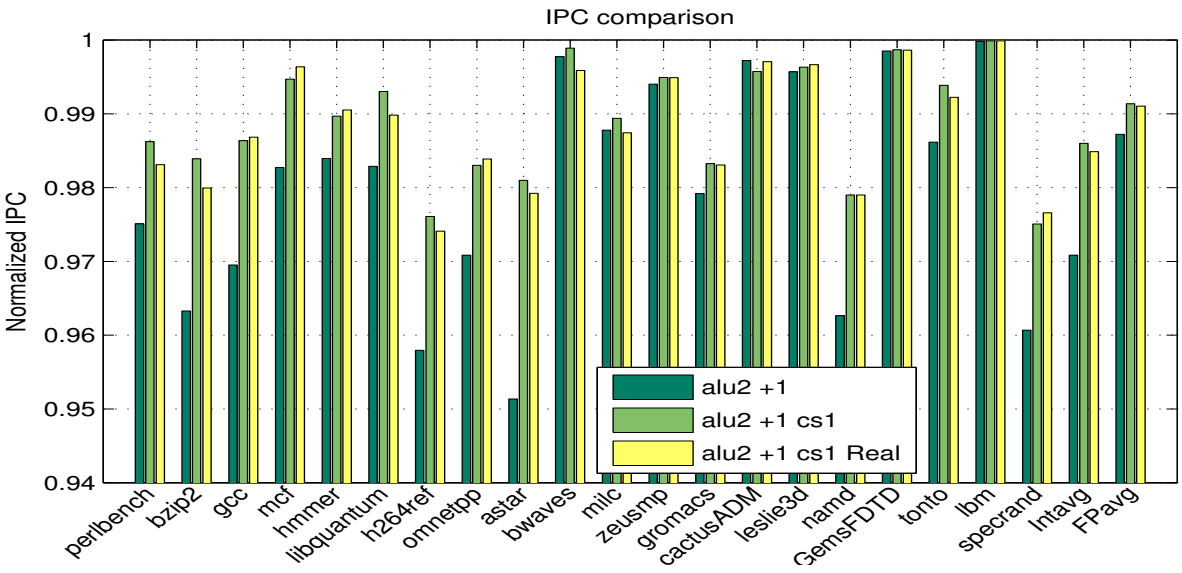


Figure 32: Performance overhead reduction of CS1-prediction based instruction steering in ideal case, alu2+1 cs1 reduce performance overhead from 2.9% to 1.4% (INT) and 1.25% to 0.85%; in reality, alu2+1 cs1 Real approaches the ideal limit.

5.5.4 Lifetime Improvement

To evaluate the lifetime improvement, we implemented and compared the following schemes. The vulnerability of FU is determined by process variations (PV) at manufacturing time, and the dynamic stress due to BTI and TDDB.

- `rep1` — We reserve GFU/m (or GFU/s) as a spare unit before the first FU in its group fails.
- `rep2` — We configure GFU/m (or GFU/s) to replace the most vulnerable FU in its group at the beginning and replace the first appeared failed FU thereafter.
- `rr` — We reconfigure GFU/m (or GFU/s) to replace all FUs in its group in a round-robin fashion.
- `adap` — We reconfigure GFU/m (or GFU/s) to replace the most vulnerable FU in its group based on online self test.

- `adap+repl` — Similar to `adap`, but we reconfigure GFU/m (or GFU/s) to replace the first failed FU in its group.
- `dup` — Each FU has a duplicated copy. This is using GFU/d for all FUs.

To statistically evaluate the lifetime impact, we generated the floor-plan of 29 FUs, one added GFU/s, and one GFU/m using VARIUS [62]. Due to process variation, some FUs are slower and more vulnerable to aging induced failure in some samples. We generated and evaluated 400 execution stage samples to reduce random odds. In addition, the BTI and TDDB induced aging effects were calculated according to models in Chapter 2. Different schemes of using GFU have different net stress and recovery/idle time, hence the aging progress are different.

In the following discussion, we use GFU/m and GFU/s to recover the first and third group of FUs. Assuming the lifetimes of FUs in the first and third FU groups are sorted in ascending order as $t_1, t_2, \cdots$ (with their names denoted as $FU_1, FU_2, \cdots$). Hence the first failed FU stops working at time t_1. We also denote the lifetime of GFU as t_r.

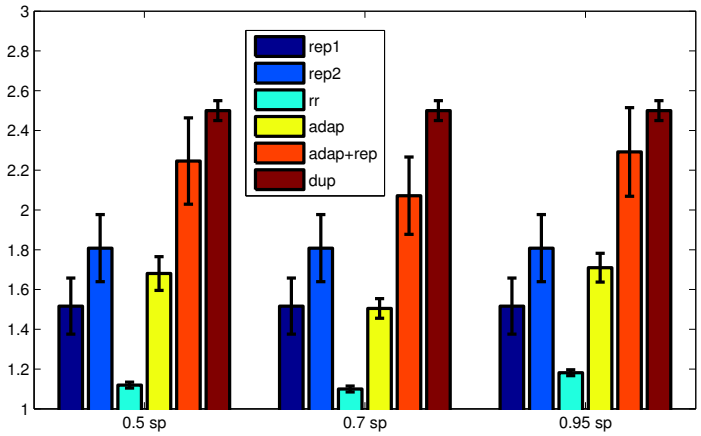


Figure 33: MTTF improvement

Fig 33 shows the normalized lifetime of six schemes over the baseline design. Since the BTI-induced degradation depends on signal probability inside the corresponding FU, x-axis draws the scenarios with three possible (uniform) signal probability — 0.5, 0.7 and 0.95. Since we consider both NBTI and PBTI and pick the worst degradation of the two, the effective signal probability is always larger than 0.5.

`rep1` replaces FU_1 with GFU at t_1. After t_1, GFU replaces the failed FU to form a correct core. The core may fail at t_1+t_r if GFU fails early, or at t_2 if FU_2 fails early. The lifetime of `rep1` is therefore $\min(t_2, t_1 + t_r)$.

`rep2` replaces FU_1 at the beginning such that the first failure appears at either t_2 or t_r. If at t_2, then GFU is configured to be FU_2. The next failures appears at min(t_1+t_2, t_r, t_3). If the first failures appears at t_r, then the next failure appear at $\min(t_2, t_1 + t_r)$. Although `rep2` statistically extends lifetime more than `rep1` (as shown in Fig 33), it incurs more performance degradation from the beginning.

`rr` replaces all FUs in round-robin way from the beginning. For original FUs, the net recovery/idle time increases and net stress time decreases as GFU replace them. Assuming GFU does not fail and the lifetime of all original FUs are extended from $t_1, t_2, \cdots$ to $t'_1, t'_2, \cdots$ (sorted in ascending order). Hence the lifetime is extended from t_1 to t'_1.

`adap` replaces more vulnerable FUs with GFU in more epochs, and less vulnerable FUs in less epochs. The correlation between recovery time and vulnerability resulting in balanced final lifetime of several most vulnerable FUs. Fig 33 shows that it achieves better lifetime than `rr`.

Ideally, `adap` extends the lifetime to t_x where $t_r = \sum (t_x - t_i)$ for all $t_i < t_x$ $(i >= 1)$. That is, GFU extends the failing time of several most vulnerable FUs to about the same time. By the time when the first failed FU appear, the next one is very close to fail (assuming $t_1 + t_r > t_2$). However, in practice, this is affected by the resolution of the online margin detector. GFU may be scheduled to recover a non-the-most-vulnerable FU due to imperfect margin detector. While the margin difference is small, the lifetime difference is actually large. This is due to the power law of BTI and TDDB degradations: when FU is used more, it takes much longer time to degrade for same degree. We thus adopted `adap+repl` that further extends the lifetime by replacing the first failed FU with GFU. Fig 33 showed that on average, it achieves more 40% lifetime improvement.

`dup` prepares a replicated copy for each FU in the core. Since it is impossible to know which unit is the most vulnerable unit at design time, a safe bet is full redundancy with an area overhead 100%.

The error bars in Fig 33 show the range of 400 samples. As we can see, `dup` has small error bar, i.e., it has good ability to tolerate PV and usage differences. `adap+repl` has relatively larger error bar than `adap`, which indicates the variation comes from replacing the failed FU. That is, the worst FU varies significantly in different samples. On average, `adap+repl` improves MTTF to 2.2x on average. This improvement is not only much better than simply using replacement FU or round-robin usage, but also is close to more area-expensive `dup`.

5.6 SUMMARY

In this chapter, GFU is designed to significantly improve microprocessor reliability with modest area overhead and small and controllable performance degradation. By exploiting the area and usage characteristics of different FUs in modern processor cores, we categorized three groups of FUs and developed different recovery strategies. GFU/m and GFU/s have the flexibility to be reconfigured to recover any FU is each group. By recovering at fine-granularity, GFU/s incurs small to negligible performance overhead. When using GFU/m to recover frequently used unit, instruction steering is designed to further reduce performance overhead. The results from simulations show that the scheme help to reduce the area overhead but greatly improve the hardware reliability of processor cores.

5.7 RELATED WORK

This section summarize the related work in processor core logic reliability and reconfigurable function unit.

5.7.1 Processor Core Logic Reliability

Past works on improving core reliability can be grouped into three categories: aging recovery, handling soft errors and handling hard faults.

To combat core logic aging, [32] uses adaptive voltage control. There are several papers on how to proactively recover core logic. In [1]: idle cycles are utilized to recover NBTI induced degradations by balancing signal probabilities on gates, which is be quite effective. [16] studies NBTI stress with process variation. With process variation, some FUs are inherently weaker, slower and vulnerable to NBTI stress. By using the vulnerable FU less and the robust FU more, the vulnerability difference initially introduced by PV is evened out in field use and the lifetime is greatly improved. However, previous recovery techniques only work for NBTI induced degradation, the solution is still needed for other degradation mechanisms such as TDDB, HCI, etc.

Soft errors in pipelines are tolerable by check-pointing and re-execution: in [75], the check-pointing fabric adds roughly 20 to 50% to the digital-logic area; [12] uses shadow latch to detect the timing errors in execution path, and re-execute the stage with error after the pipeline is stalled. Although expensive, these techniques work well.

To handle hard errors, disabling faulty cores is common practice to maintain acceptable yield. Considering throwing away faulty cores is too expensive, [55] and [3] try to salvage and use faulty cores: [55] observes some faulty function units are not frequently used and it's safe to run applications that do not utilize the faulty FU on the core; [3] observes a faulty core is likely to be an animator core in sufficient long period and provides useful hints to accelerate the execution of normal cores.

Different from core level, the FU level redundancy has smaller area overhead. FU level redundancy in [66] and structural duplication in [67] have been proposed to maintain core functionality. Both use separate micro-architectural spares, such as ALUs to replace the faulty one. However, the fault coverage by inherent function unit redundancy is limited. In [55], only 26% area of execution units are covered by inherent redundancy. Without extra redundancy, inherent redundancy is only applicable to defects in about 10% of core area.

To achieve high fault coverage, simply using spares in core or FU level may incur unnecessary high overhead. In FU level, using spare FU for each FU is prohibitively expensive. It is observed that the cases when all FUs are faulty is extremely unlikely, and it not necessary nor cost effective to use spares for all FUs. However, it is also hard to determine which FU is most vulnerable post fabrication or stressed most in field use. Hence, flexibility is

desirable for using redundant FU to improve reliability. A reasonable alternative is to use reconfigurable FU to replace the faulty FU.

5.7.2 Reconfigurable Function Unit

Previously, reconfigurable fabric are used mainly for performance improvement, power saving and other purposes and are implemented in Field Programmable Gate Arrays (FPGAs). [59], OneChip [78], Garp [24] are among the earliest works to use reconfigurable FPGA as coprocessor to accelerate CPU performance. In FPGA-based co-processor, a sequence of instructions, instead of single instruction, is mapped to hardware and run concurrently to boost performance. Small kernels in Garp [24] may increase the performance by 10X. Later works such as OneChip [78], Chimaera [81], Tartan [48] improve performance of FPGA-based reconfigurable FU. In [57], with board level CPU and FPGA hybrid platform, multiple cache architecture is proposed to accelerate the FPGA HPC platform. The compiler CHiMPS automatically allocates and optimizes the on-chip memory as the cache of FPGA. Besides performance improvement, in [21] the traditional LUT storage, SRAM cells, are replaced by emerging Spin Torque Transistors (STT), and using STT-based LUT designs replace multiple components in processor pipeline to reduce power. Similarly in [73], conservation core is proposed to save energy. Another interesting technique of using FPGA based design is FlexCore [9], which uses reconfigurable FPGA fabric for monitoring the execution of core. None of the above techniques focus on reliability improvement by utilizing a reconfigurable FU.

The reason of not using FPGA-based configurable FU to replace ASIC designed FU is the large performance gap. Although FPGA-based co-processors boost performance by exploiting parallelism, when used as an FU in execution stage, the parallelism is lost and the custom ASIC design FU is still a must. The gap between the FPGA and ASIC design is still huge [39, 79]: FPGA design is 3X in critical path delay, 17X to 27X in area and 12X power hungry than ASIC design.

A faster yet less flexible design is combining customized ASIC circuits and reconfigurable fabric with multiplexers. In recent high-end commercial FPGAs, other than LUTs, Macro

circuits such as small adders, shifters and even multipliers (DSP48E in Xilinx FPGAs) are also basic building blocks of reconfigurable fabric. In [82] and [83], the compound circuits are are configurable by setting multiplexers to choose from and build different FUs. In the DSP benchmarks, 1/3 more area is used and 37% more delay is introduced. Although the Macro based reconfigurable design is studied in context of embedded processor [83], similar idea could be used in high performance processors: Macro-based RFU is fast enough to replace customized ASIC FU for improving reliability.

6.0 BIT FLIPPING IN TDDB RELIABILITY IMPROVEMENT

Previous chapters study the BTI and TDDB problems in SRAM-based structures and function units. The fundamental idea is to create and/or exploit recovery opportunities that has been shown to be effective in improving microprocessor reliability. However, there are other critical structures with little idle cycles for recovery, such as scheduler in pipeline, DTLB, ITLB and L1 cache since they are used or accessed very frequently. These critical FUs are vulnerable under stress, especially TDDB.

TDDB degradation highly depends on temperature and voltage as described in Eqn. 2.5. It is very sensitive to voltage: experiments in [61] show TDDB lifetime increase to 100x when gate voltage is reduced by $0.1V$ in the range of $1.2V$ to $1.4V$. Hence, TDDB lifetime improvement is a side product of traditional voltage reducing techniques for power saving. For less critical or less frequently used FUs such as last level cache, reducing voltage is an effective way to tolerate TDDB stress. However, for critical and busy FUs where reducing voltage incur intolerable performance loss, TDDB is still an serious reliability problem. In this chapter, the study shows that 1) reducing voltage can be used in many FUs to tolerate TDDB, including but not limited to function units in execution stage, lower level cache hierarchy; 2) some FUs (scheduler, DTLB, ITLB and L1 cache) are vulnerable to TDDB and hard to tolerate TDDB where reducing voltage is not feasible due to potential large performance overhead.

As an alternative to reducing voltage, Section 2.2.1 shows significant TDDB reliability improvement is achieved with increased bit flipping frequency. Inspired by the frequency dependent TDDB stress, a simple yet effective bit flipping circuit is designed in Section 6.2 for scheduler design. To achieve desirable reliability improvement, the bit flipping circuit is utilized at target high frequency. Results in Section 6.3 show scheduler, DTLB and L1/L2

cache have sufficient updated activities upon which bit flipping circuit could be used to increase bit flipping frequency. The evaluations of performance, area overhead and reliability improvement are discussed in Section 6.5.

However, we also find one structure, ITLB, does not have natural high update activities to utilize the bit flipping circuit for increasing bit flipping frequency, as shown in Section 6.3.2. Hence we need to design proactive bit flipping scheme for ITLB. One essential requirement for proactive bit flipping scheme is its low delay overhead, since ITLB is used in almost every cycle to translate instruction's virtual address to physical address. In Section 6.4 we design a promising local write circuit (LWC) to implement proactive bit flipping. Although LWC has negligible delay overhead, its per-entry area overhead is non-negligible. Luckily, the number of entries in ITLB is usually not large and the added area by LWC is amortized when compared to comparators for fully associative searching.

In this chapter, we first briefly study the matrix-based scheduler in Section 6.1. And we use the entries in scheduler as an example to show how bit flipping circuit is designed in Section 6.2. Then in Section 6.3 we present results showing many structures have sufficient natural update activity to utilize the bit flipping circuit. For the ITLB whose natural update activity fails to meet the target, in Section 6.4 we design a light overhead local write circuit for proactive bit flipping. Finally the results are presented in Section 6.5.

6.1 MATRIX-BASED SCHEDULER

In this section we introduce the scheduler structure that is not covered in previous chapters. Scheduler is an large and important unit in Out-of-Order pipelines. In Intel Nehalem microprocessors, the area of scheduler is 80% of all function units execution stage combined. The scheduler is responsible for tracking the operands dependencies, waking up ready instructions and picking proper instruction for execution stage when resource conflicts occur. To keep the pipeline busy, it is a busy and hot by itself. The scenario is even worse considering dynamic circuit structure is often used that incurs more dynamic power and higher temperature, making scheduler an extremely vulnerable unit under TDDB degradation.

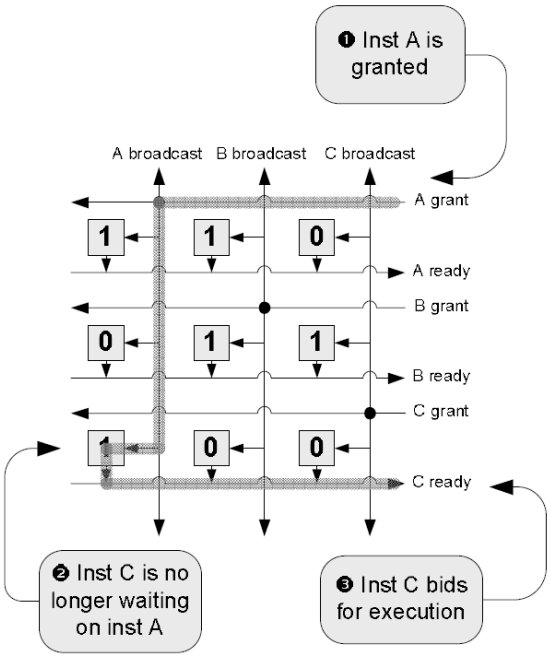


Figure 34: Wakeup Matrix in [63]

Although traditionally comparator based schedulers are used, in this section we focus on the matrix-based scheduler first proposed in [18] and then improved in [63]. Comparator based schedulers suffer high power and delay due to a large number comparators are used to compare the broadcast and stored tags. The matrix-based scheduler, as shown in Fig. 34, replaces large comparators by matrix which stores the dependency and age information for wake-up and select, resulting smaller, faster and lower power hardware. The advanced matrix scheduler design is found in recent IBM z196 mainframe processor (dependency and age matrix schedulers are used).

The dependency matrix for wake-up is used as an example to show how matrix scheduler works. Fig. 34 shows the operation of dependency matrix. The storage cell in row i and column j stores the flag whether instruction i depends on instruction j, i.e., instruction j (producer) generates at least one result to be consumed by instruction i (consumer). In step (1), instruction A is granted for execution and the dependency of instruction C on A is cleared in step (2). When all bits in row C are cleared, ready signal is generated and sent to

aging matrix for grant signal. The details of similar age matrix for picker is also documented with detail in [63].

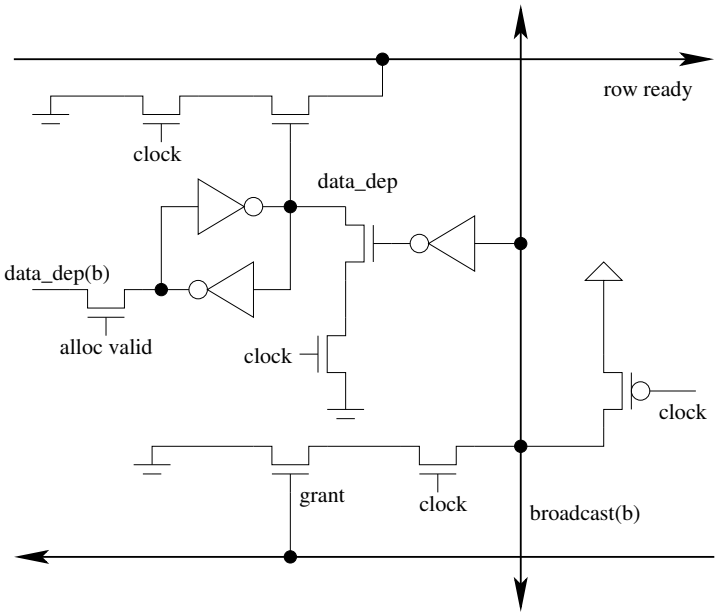


Figure 35: Wakeup Matrix Circuit in [63]

Fig.35 shows the dynamic circuit implementation for cells in dependency matrix. For dynamic circuit in every cycle there are two phases, precharge and evaluation, controlled by the clock in Fig.35. The dependency information is initialized by writing $\overline{data\_dep}$ into the cross coupled inverter (similar to SRAM) when *alloc_valid* signal is on in the dispatch stage. If *data_dep* is "1" (instruction in row i depends on instruction in column j), in the evaluation stage the *row_ready* signal for instruction in row i is drained to "0". If there is any dependency unresolved for instruction in row i, the *row_ready* is evaluated at "0" in evaluation stage. Hence the *row_ready* signal is the NOR function of all dependency bits. The dependency bit is cleared by a grant signal generated by the producer at column j, when the producer is in execution and would generate the result soon. The grant signal drives the $\overline{broadcast}$ signal in column j to ground in evaluation and all the dependency bits in column j are cleared. In matrix scheduler, the dependency information is passed from producer to consumer by grant, broadcast and row ready signals. Although the matrix scheduler design has advantages over comparator based scheduler in terms of complexity and area, its power

is still large due to the dynamic circuit operations in every cycle, making TDDB a serious problem.

One note for the scheduler is that the BTI stress is a less severe problem than TDDB. Although the result in Section 6.3.1 shows severe imbalanced signal probability in matrix, the degraded cross-coupled inverters (storage cell) have little impact on the overall operations. This is due to the storage cell no longer drives the bit line by itself as does in cache. Instead in scheduler it controls the gate terminal of one transistor, which is not affected by the potential access failure or read failure. Since BTI marginally improves write failure, we don't treat BTI as an important problem in scheduler.

6.2 BIT FLIPPING CIRCUIT

In Chapter 2, TDDB stress shows high dependence on flipping frequency and there is the opportunity to improve TDDB reliability by increase the bit flipping frequency, as an alternative to dynamic voltage/temperature management. To proactively increase the bit flipping frequency, bit flipping circuit is designed in this section. The matrix based scheduler is used as an example and the design is also applicable to other entry-based structures.

To support bit flipping, one multiplexer is added to control the which node to control the gate as dependency bit, as shown in Fig. 36. In original design only *data_dep* controls the gate for draining row ready signal. With adding a pair of multiplexers both *data_dep* and $\overline{data\_dep}$ can be used to control the gate, decided by the flip signal for all bits in one row: when flip is 1, *data_dep* is used as normal case; otherwise, $\overline{data\_dep}$ is used. Note that one demultiplexer is also added to determine which note to discharge when broadcast signal comes. By changing the flip signal, even the effective *data_dep* seen by the control gate remains unchanged for a long time, the physical *data_dep* and $\overline{data\_dep}$ experience bit flipping at desired frequency.

The flipping circuits are applied similarly to other structures, such as branch predictor, cache, TLB and etc, by adding multiplexers/demultiplexers in read/write path such that either normal or inverted data are stored and accessed. In a similar way, we toggle the flip

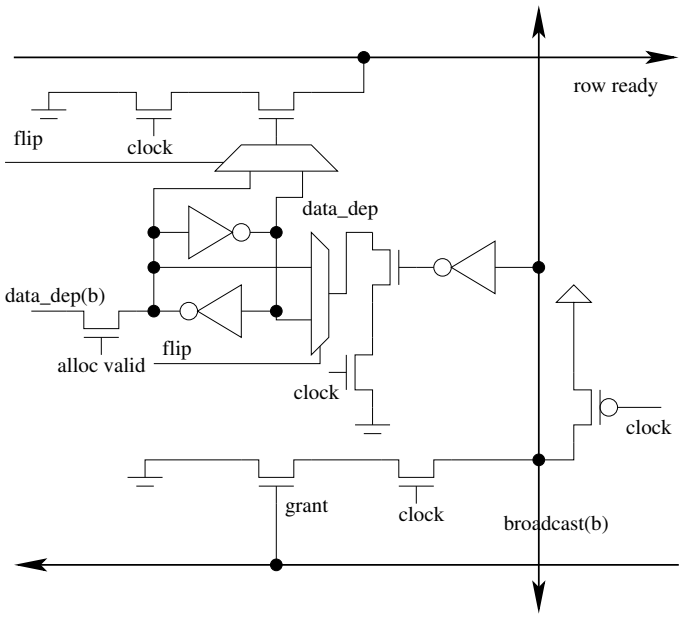


Figure 36: Bit Flipping Support in Wakeup Matrix Circuit

signal when there is a update to the entry. For cache and TLB, the update activities include read miss, write hit/miss. For branch predictor, the update activities include new branch decision and target address update on mis-prediction. However, the frequency of transition between normal and flipped data depends on the frequency of update activity to the entry. We would study the update activity in multiple structures shortly.

6.3 BIT FLIPPING FREQUENCY IN MULTIPLE UNITS

This section studies the achievable bit flip frequency in matrix scheduler, caches and TLBs. The bit flipping is performed at every entry update, so the bit flipping frequency is closely related to entry update frequency: for read access the bit flipping frequency is same as the entry update frequency; for write access the bit flipping frequency of unchanged bits is same as the entry update frequency. 1KHz as set as the target for effectively using bit flipping technique to improve TDDB reliability. The simulation setting is same as in Table 8.

6.3.1 Matrix Scheduler

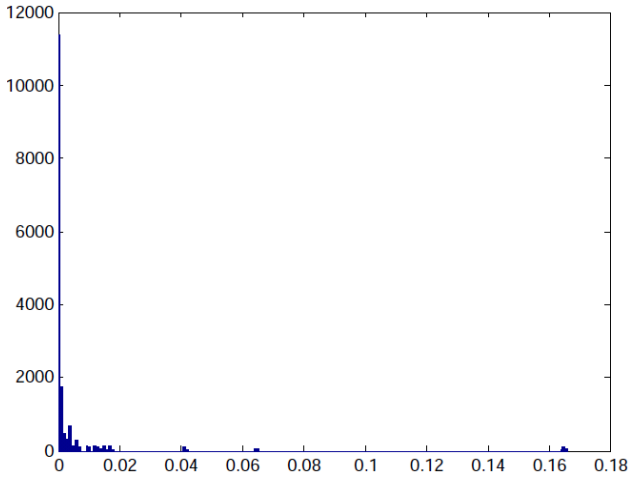


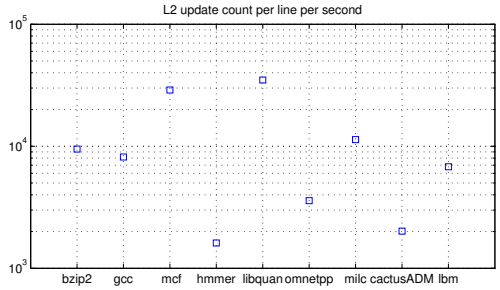
Figure 37: Signal Probability in Wakeup Matrix

Fig. 37 shows the histogram of signal probability of dependency bits in wakeup matrix. The signal probability has strong bias toward 0: dependency bits are 0 in 98% cycles. Although the entries in wakeup matrix are updated busily since every dispatched instruction allocates one entry, the natural bit flipping occurances are extremely rare. This is due to that the number of true dependency of one instruction (at most 3 in the simulator) is much smaller than issue queue size. Except for the dependency bits, all other bits keep unchanged at 0 during the access. This is the typical example that the natural entry update failed to generate frequent bit flipping despite of frequent entry update and Stale dependency bits pose a serious threat to matrix scheduler reliability under TDDB stress. Applying the bit flipping circuit in the design greatly increases the flipping frequency of most bits to entry update frequency, resulting improved reliability under TDDB stress.

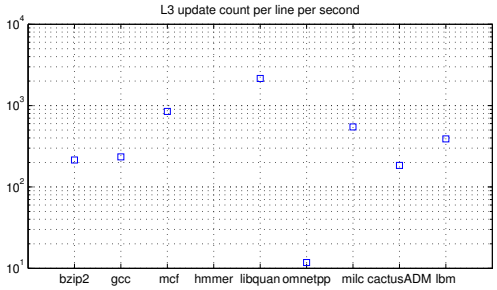
6.3.2 Cache and TLB

The update frequency data are also extracted from architecture simulations. Fig. 38(a) and Fig. 38(b) show the average update count per line in one second for L2 and L3 caches. The update frequency L2 is above targeted 1KHz and bit flipping circuit is used to translate it

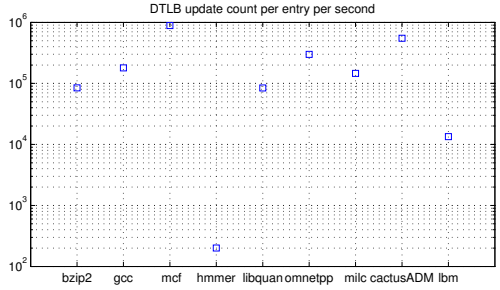
into most bits' flipping frequency. However, the update frequency for L3 of most benchmarks are under 1KHz. This is not a big issue since reducing L3 voltage is a practical approach to be applied in conjunction of bit flipping without introducing noticeable performance overhead.



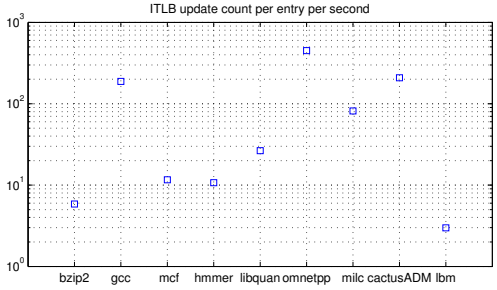
(a) L2 cache update count



(b) L3 cache update count



(c) DTLB update count



(d) ITLB update count

Figure 38: Update Count for Multiple Structures

Fig. 38(c) and Fig. 38(d) show the average update count per entry in one second for DTLB and ITLB. The update frequency for DTLB in most benchmarks is above 1KHz and bit flipping is applied. However, for ITLB, the update frequency is in the range of 3 to 500. To meet the 1KHz target, the proactive bit flipping design is introduced.

6.4 LOCAL WRITE CIRCUIT FOR ITLB

Since in ITLB the natural update activities are not frequent enough to improve the bit flip frequency, a proactive bit flipping technique is designed when there is no natural update

activity. One easy and direct method is to read the data out from the entry, apply inversion operation and then write the inverted data back to the entry. Although simple and effective, this method incurs performance overhead that may become non-negligible when the target bit flipping frequency is high. Then we design the low overhead Local Write Circuit (LWC) to implement fast and energy efficient proactive data flipping.

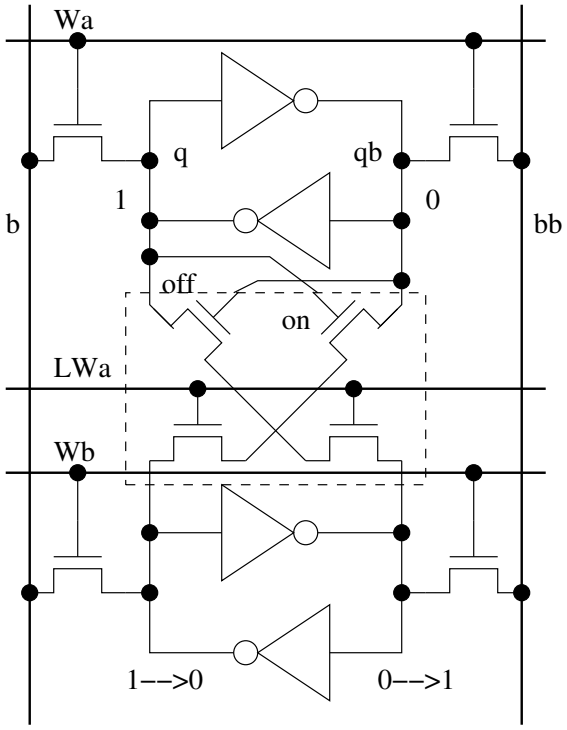


Figure 39: Local Write Circuit

Fig. 39 shows our LWC design that enables SRAM cell (top) to write its inverted data to its adjacent cell (bottom). For each 6T SRAM cell, we add 4 NMOS gates in the dash-line box. Two NMOS gates are controlled by complementary nodes to route the data to the their complementary nodes. And another two NMOS gates controlled by LWa determine whether the local write operation is enabled. For example, if the top SRAM cell stores "1" ($q = 1, qb = 0$), when local write enable signal LWa is asserted, the qb node of top SRAM cell and q node of bottom SRAM cell are connected and then compete (if $q = 1, qb = 0$ in bottom SRAM cell). The result is q of bottom SRAM cell is drained to ground by qb of top SRAM cell, the qb of bottom SRAM cell is driven to "1" and "0" is written. Two factors

determine qb of top SRAM cell drive q of bottom SRAM cell: 1) SRAM cells are designed to favor larger NMOS for easily draining nodes to ground, and 2) NMOS gate conducts good "0"s, not good "1"s. Hence the LWC guarantees deterministic operation of writing inverted data to adjacent cells. The circuit is named "Local Write" since the cell is written by a local cell, not by global bit lines via enabling access transistors. Since there is no bit line precharging, the local write operation both fast ($< 500ps$) and power efficient (4% of global write).

However, one drawback of LWC is the area overhead. For each 6T SRAM cell, we add 4 NMOS gates. The results show 35% area overhead on the SRAM array. However, considering the supporting logic such as decoders and comparators in ITLB, the area overhead is amortized. Also careful layout would optimize area overhead. With the support of LWC, light weight proactive bit flipping is possible.

6.5 RESULTS

6.5.1 Analysis of Local Write Circuit

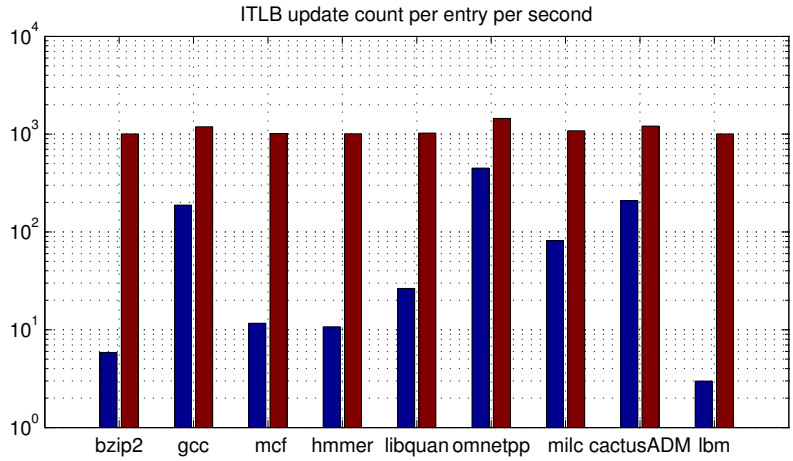


Figure 40: Update Activity of ITLB with Local Write Circuit

In the simulation a 2GHz processor is used with 196 entries fully associative ITLB. It's observed that a line is updated much less than 1KHz on average in most benchmarks. Hence the local write circuit is integrated in ITLB to proactively increase the flipping activity. A simple design is set the local write frequency at the targeted 1KHz, no matter what the natural update frequency is. In our settings this requirement translates into 5.1μs (10.2K cycles) interval for two local write operations. The update activity on each line is increased by 1000 times per second, as shown in Fig. 40.

With increased flipping frequency, the ITLB lifetime is increased. Fig. 41 shows the ITLB lifetime increase. The gcc and omnetpp applications have limited improvement due to their natural update activity is close to 1KHz. For other applications with very limited natural update activity such as bzip2 and lbm, the improvement is significant.

6.5.2 LWC Overhead Evaluation

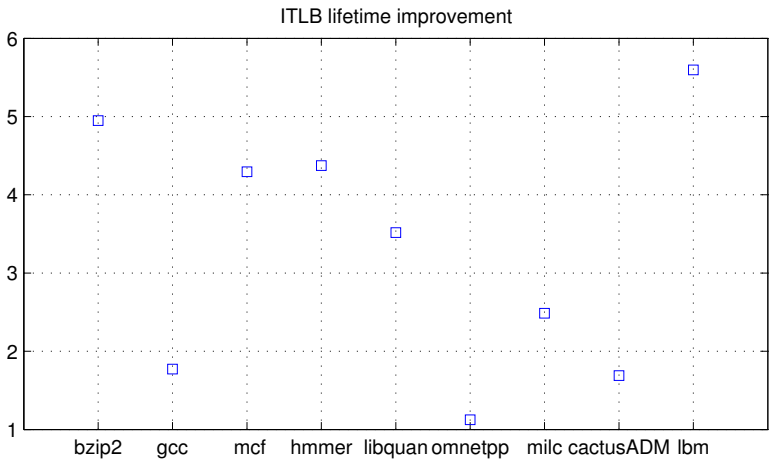


Figure 41: ITLB MTTF with Local Write Circuit

The power, latency and area of local write circuit design in ITLB is evaluated. The ITLB has 192 entries and is fully associative. The processor cycle time is 0.5ns in 2GHz clock. The target flipping frequency of ITLB entries is set to 100KHz in context that ITLB is vulnerable under high temperature.

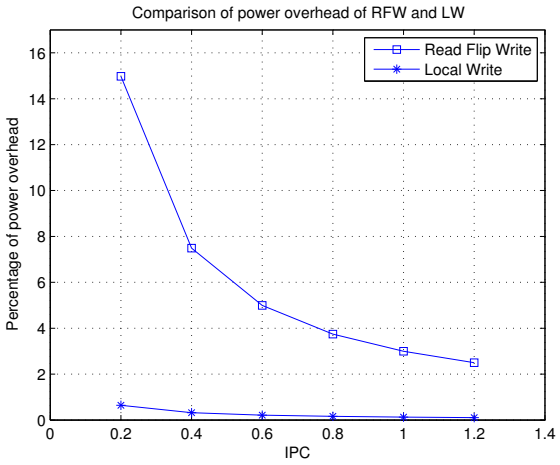


Figure 42: Comparison of RFW and LW Power in ITLB

Both Read Flip Write (RFW) and Local Write (LW) circuits proactively flip the bits in ITLB cache lines. However the LC is more power efficient than RFW since the driving internal node with small capacitance consumes less power than driving bit lines with significantly larger load in RFW. Fig 42 shows the comparison of power of the two circuits in ITLB. It's observed that the percentage of power overhead (in y axis in Fig. 42) depends on IPC. The power consumer of ITLB is smaller with low IPC applications since less instructions are fetched after ITLB is accessed. In the simulations the applications have varying IPC ranging from 0.2 to 1.4 (in x axis in Fig. 42). Across the IPC range the RFW consumes 2.3% to 14.9% and LW consumes 0.1% to 0.64% total power.

The latency overhead of RFW is 2/3 cycles, depending on whether flipping is merged into read/write stage. The latency overhead of LW is 1 cycles. During RFW or LW is engaged, the ITLB blocks all accesses to it and cause potential performance loss. In the worst case, each RFW or LW blocks some accesses. However targeting at 100KHz proactive bit flipping frequency with 256 entries takes a negligible portion of execution cycles of a 2GHz processors. In the machine configuration the worst case performance overhead of RFW and LW circuits are 1.28% and 2.56%.

The area overhead of LW is larger than RFW. RFW requires one extra line and flipping circuit composed of one level XOR gates. LW requires 30% more area in the data lines. However the data lines only takes less than 20% of total ITLB area that is dominated by complex comparators. Hence the area overhead of LW is 6% on ITLB and is negligible considering the whole chip area.

6.5.3 MTTF improvement for Scheduler and L2 Cache

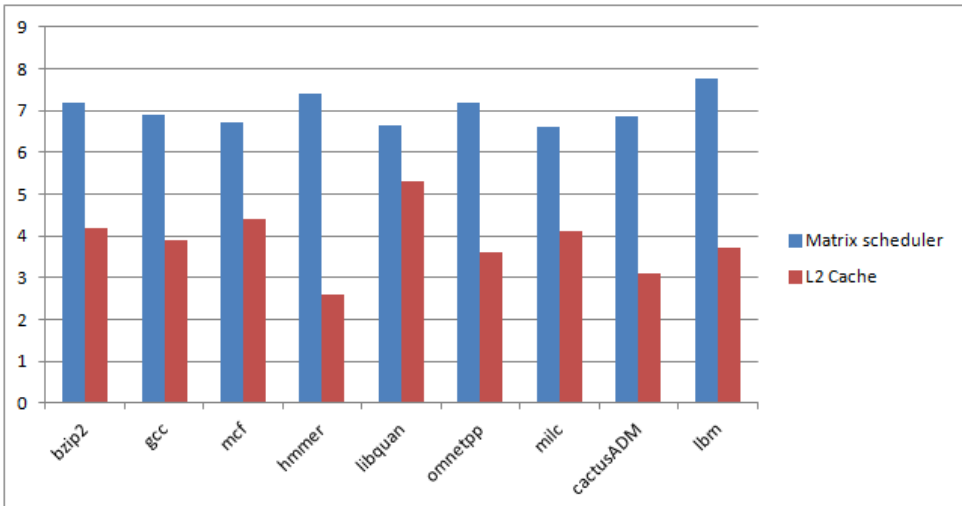


Figure 43: MTTF Improvement for Scheduler and L2 Cache

The reliability of matrix scheduler and L2 cache under TDDB stress is improved by applying bit flipping circuit during proper entry update activity. Fig 43 shows the MTTF improvement for scheduler and L2 cache. The matrix scheduler has larger improvement due to higher bit flipping frequency. The simple bit flipping circuit is a simple and effective design to improve multiple units' reliability under TDDB stress.

6.6 SUMMARY

This chapter discuss the TDDB reliability issues in memory based structures. Although TDDB could be effectively mitigated by reducing voltage or shutting down, some FUs that

are always on critical path such as scheduler need alternative to improve TDDB reliability. The bit flipping circuit is inspired by the interesting characteristic of frequency dependent TDDB stress. With the results we learned there are sufficient update frequency to utilize bit flipping circuit to increase the bit flip frequency. Ongoing work is being performed to further study the designs and evaluations of TDDB reliability improvement.

7.0 CONCLUSION

As discussed in Introduction, the circuit reliability is as important as traditional key objectives: timing, power and area in complex and large scale processor design. Investing little effort in reliability aware design leads to compromised yield and rising cost in manufacture phase and shortened life time in service phase. It is also an increasingly challenging problem when technology advances into near future 20nm and 14nm since small nodes are more vulnerable to manufacturing defects or in-filed disturbances. In spite of massive research interests in reliability aware design, this thesis focuses on aging induced hard errors, especially the parametric hard errors caused by BTI and functional errors caused by TDDB.

Chapter 2 gives a deep dive into BTI and TDDB mechanisms. For BTI, the faster and stronger proactive recovery mode lays down the foundation for *4PR* discussed in Chapter 3. For TDDB, the signal probability dependent stress paves the way to the design of bit flipping circuit in Chapter 6. Fully studying the physics properties of the aging mechanisms enables follow up research in circuit level and architectural level to exploit the unique characteristics.

The innovative *4PR* mode and supporting circuits are designed in Chapter 3 to mitigate both PBTI and NBTI stress on SRAM cells. The advantage of the *4PR* mode are multiple fold: 1) it works for both NBTI and PBTI on all 4 internal gates; 2) it works for access failure as well as read-flip failure; 3) the design is light weight in that it doesn't require extra power rails as in other designs and it's easily incorporated in existing power gating logic; 4) the reliability improvement in terms of failure probability and MTTF is significant; 5) reduce leakage power as a beneficial side effect. The *4PR* is also important since SRAM cell based data array takes a significant if not dominating portion of die area including caches, queues in out-of-order pipeline, branch predictor queues and store-load queues.

Chapter 4 continues the discussion of *4PR* mode and studies its application in multiple structures of processor. Although the SRAM cells are little difference, *4PR* application and supporting circuit are different on L2 entries and core pipeline entries due to different access pattern and latency requirement. For the L2 cache banks, redundant cache bank is added exclusively for proactive recovery rotation. For the critical and busy function units in microprocessors, idle time of busy function units are sufficient for recovery thus exploited at per-buffer-entry level. In the evaluated L2 cache, the technique effectively slows down the cell failure probability increase, and achieves 2.85/2.12× (best/worst case) lifetime improvement over normal designs. For the function units in pipeline, the scheme achieves on average 6/3.45× MTTF (Mean Time To Failure) improvement at the cost of <1% IPC degradation and <1% area overhead.

Although *4PR* effectively mitigates the BTI stress in SRAM based structures that take a major portion of die area, the combinational logic in function units also need recovery but it's not as easy due to irregular structure and lacking redundancy. That's the motivation to design GFU framework to tolerate both BTI and TDDB in this thesis. As a reconfigurable FU, GFU replaces the function of vulnerable FU that is powered off for both BTI and TDDB recovery. Note that the *4PR* is not practical due to irregular structure of combinational logic in function unit. Since duty cycles and design complexity vary, three configurations, GFU/m, GFU/d and GFU/s are designed to replace 3 groups of FUs. Since the GFU is reconfigurable but slower than the original target FUs, it is carefully designed to minimize the performance loss when it is in-use. Schemes to avoid using the GFU on performance critical paths of a program execution are used to cover corner cases. The evaluation results show that we the lifetime of execution stage in processor core is extended by 2.2x and improve yield from 93.9% to 98.16 or 95.69% while introducing less than 4.5% performance overhead.

The last piece in this thesis is solving the TDDB stress in memory based structures in Chapter 6. The big difference of BTI and TDDB is the BTI causes parametric failure while the TDDB causes function failure. Previous technique uses voltage scaling on many less busy units to mitigate TDDB stress, however structures on critical path that are accessed every cycle require other solutions. Scheduler, ITLB and L1 cache are identified to be such structures. Based on the frequency dependent TDDB stress, the bit flipping circuit

is implemented to increase the bit flipping frequency as an essential solution to mitigate TDDB stress. Results show in many units the entry update activity is sufficiently frequent to utilize the bit flipping circuit. However, one exception is ITLB in which the natural bit flipping frequency is extremely slow. The local write circuit is designed to enable proactive bit flipping, although the area overhead is not non-negligible. Evaluations show promising lifetime improvement in scheduler, cache and ITLB.

This thesis thoroughly studies the both the parametric and functional hard errors caused by aging mechanisms of BTI and TDDB in micro-processors with up to 28nm technology (High-k metal gate and bulk). By exploiting the unique and beneficial characteristics of BTI and TDDB, the *4PR* and bit flipping are designed to mitigate the stress on memory based structures that take major portion of the processor die area. The combinational logic in multiple function units are also covered by GFU design that provides a reconfigurable redundant function unit to replace original unit when under recovery. The performance/area overhead are controlled well through careful design. Overall the reliability improvement is significant and the overhead is negligible in most cases. As the technology advances to 20nm and 14nm, more aging mechanisms are emerging and reliability is an evolving problem to researchers to solve.

BIBLIOGRAPHY

[1] J. Abella, X. Vera, and A. Gonzalez. Penelope: The NBTI-Aware processor. In *MICRO*, pages 85–96, 2007.

[2] N. Aggarwal, P. Ranganathan, N. P. Jouppi, and J. E. Smith. Configurable isolation: building high availability systems with commodity multi-core processors. In *ISCA*, pages 470–481, 2007.

[3] A. Ansari, S. Feng, S. Gupta, and S. Mahlke. Necromancer: enhancing system throughput by animating dead cores. In *ISCA*, pages 473–484, 2010.

[4] A. Bansal, R. Rao, J. Kim, S. Zafar, J. Stathis, and C. Chuang. Impact of NBTI and PBTI in SRAM bit-cells: Relative sensitivities and guidelines for application-specific target stability/performance. In *Reliability Physics Symposium*, pages 745–749, 2009.

[5] A. Bansal, R. Rao, J. Kim, S. Zafar, J. H. Stathis, and C. Chuang. Impacts of NBTI and PBTI on SRAM static/dynamic noise margins and cell failure probability. *Microelectronics Reliability*, 49(6):642–649, June 2009.

[6] D. Bernick, B. Bruckert, P. D. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen. NonStop advanced architecture. In *DSN*, pages 12–21, July 2005.

[7] J. Blome, S. Feng, S. Gupta, and S. Mahlke. Self-calibrating online wearout detection. In *MICRO*, pages 109–122, 2007.

[8] Y. Cao. Predictive technology model. Website. http://www.eas.asu.edu/~ptm.

[9] D. Y. Deng, D. Lo, G. Malysa, S. Schneider, and G. E. Suh. Flexible and efficient Instruction-Grained Run-Time monitoring using On-Chip reconfigurable fabric. In *MICRO*, pages 137–148, 2010.

[10] R. Desikan, D. Burger, S. W. Keckler, and T. Austin. Sim-alpha: a validated, execution-driven alpha 21264 simulator. In *Technical report TR-01-23*, 2001.

[11] M. Dixon, P. Hammarlund, S. Jourdan, and R. Singhal. The next-generation intel core microarchitecture. *Intel Technology Journal*, 14(3):8–27, 2010.

[12] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A Low-Power pipeline based on Circuit-Level timing speculation. In *MICRO*, pages 7–16, 2003.

[13] B. Fields, R. Bodik, and M. D. Hill. Slack: maximizing performance under technological constraints. In *ISCA*, page 4758, 2002.

[14] R. foundation. The r project for statistical computing. Website. http://www.r-project.org.

[15] FreePDK-45nm. A variation-aware design kit for 45nm. Website. http://avatar.ecen.okstate.edu/projects/scells/OSUFreePDK45/indexframe.html.

[16] X. Fu, T. Li, and J. Fortes. NBTI tolerant microarchitecture design in the presence of process variation. In *MICRO*, pages 399–410, 2008.

[17] X. Garros, L. Brunet, M. Rafik, J. Coignus, G. Reimbold, E. Vincent, A. Bravaix, and F. Boulanger. PBTI mechanisms in la containing hf-based oxides assessed by very fast IV measurements. In *Electron Devices Meeting (IEDM), 2010 IEEE International*, pages 4.6.1–4.6.4. IEEE, Dec. 2010.

[18] M. Goshima, K. Nishino, T. Kitamura, Y. Nakashima, S. Tomita, and S.-i. Mori. A high-speed dynamic instruction scheduling scheme for superscalar processors. In *Proceedings of the 34th annual ACM/IEEE international symposium on Microarchitecture*, MICRO 34, page 225236, Washington, DC, USA, 2001. IEEE Computer Society.

[19] T. Grasser, W. Gos, V. Sverdlov, and B. Kaczer. The universality of NBTI relaxation and its implications for modeling and characterization. In *Reliability physics symposium, 2007. proceedings. 45th annual. ieee international*, pages 268–280, 2007.

[20] E. Gunadi and M. H. Lipasti. CRIB: consolidated rename, issue, and bypass. In *ISCA*, page 2332, 2011.

[21] X. Guo, E. Ipek, and T. Soyata. Resistive computation: avoiding the power wall with low-leakage, STT-MRAM based computing. In *ISCA*, pages 371–382, 2010.

[22] S. Gupta, S. Feng, A. Ansari, B. Jason, and S. Mahlke. The StageNet fabric for constructing resilient multicore systems. In *MICRO*, pages 141–151, 2008.

[23] S. Gupta, S. Feng, A. Ansari, and S. Mahlke. Erasing core boundaries for robust and configurable performance. In *MICRO*, pages 325–336, 2010.

[24] J. R. Hauser and J. Wawrzynek. Garp: a MIPS processor with a reconfigurable coprocessor. In *FPGA*, pages 12–21, Apr. 1997.

[25] M. Horiguchi. Redundancy techniques for high-density DRAMs. In *, Second Annual IEEE International Conference on Innovative Systems in Silicon, 1997. Proceedings*, pages 22–29. IEEE, Oct. 1997.

[26] I. Inc. Intel 64 and ia-32 architectures software developer's manual. 325462-040US, October 2011.

[27] S. Inc. Design compiler: Early rtl exploration accelerates design schedules. Website. www.synopsys.com.

[28] D. P. Ioannou, S. Mittl, and G. L. Rosa. Positive bias temperature instability effects in nmosfets with hfo2/tin gate stacks. *IEEE TDMR*, 9(2), June 2009.

[29] B. Kaczer, T. Grasser, P. Roussel, J. Martin-Martinez, R. O'Connor, B. O'Sullivan, and G. Groeseneken. Ubiquitous relaxation in bti stressing–new evaluation and insights. In *Proc. of 46th IRPS*, pages 20–27, Phoenix, 2008. IEEE.

[30] K. Kang, S. Gangwal, S. P. Park, and K. Roy. NBTI induced performance degradation in logic and memory circuits: how effectively can we approach a reliability solution? In *ASPDAC*, pages 726–731, Seoul, Korea, 2008.

[31] K. Kang, H. Kufluoglu, K. Roy, and M. A. Alam. Impact of Negative-Bias temperature instability in nanoscale SRAM array: Modeling and analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(10):1770–1781, 2007.

[32] U. R. Karpuzcu, B. Greskamp, and J. Torrellas. The BubbleWrap many-core: Popping cores for sequential acceleration. In *MICRO*, pages 447–458, Dec. 2009.

[33] D. Khalil, M. Khellah, N. Kim, Y. Ismail, T. Karnik, and V. De. Accurate estimation of SRAM dynamic stability. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(12):1639–1647, 2008.

[34] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. Hoe. Multi-bit error tolerant caches using Two-Dimensional error coding. In *MICRO*, pages 197–209, 2007.

[35] S. Kosonocky, T. Burd, K. Kasprak, R. Schultz, and R. Stephany. Design in scaled technologies: 32nm and beyond. In *Symposium on VLSI Technology Digest of Technical Papers*, 2012.

[36] R. Kumar and G. Hinton. A family of 45nm IA processors. In *ISSCC*, pages 58–59, 2009.

[37] S. Kumar, K. Kim, and S. Sapatnekar. Impact of NBTI on SRAM read stability and design for reliability. In *ISQED*, pages 6 pp.–218, 2006.

[38] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar. An analytical model for negative bias temperature instability. In *ICCAD'06*, pages 493–496. ACM, 2006.

[39] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. In *FPGA*, pages 21–30, 2006.

[40] H. Labs. Cacti. Website. http://www.hpl.hp.com/research/cacti/.

[41] K. T. Lee, J. Nam, M. Jin, K. Bae, J. Park, L. Hwang, J. Kim, H. Kim, and J. Park. Frequency dependent TDDB behaviors and its reliability qualification in 32nm high-k/metal gate CMOSFETs. In *Reliability Physics Symposium (IRPS), 2011 IEEE International*, pages 2A.3.1–2A.3.5. IEEE, Apr. 2011.

[42] C. R. Lefurgy, A. J. Drake, M. S. Floyd, M. S. Allen-Ware, B. Brock, J. A. Tierno, and J. B. Carter. Active management of timing guardband to save energy in POWER7. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 '11, page 111, 2011.

[43] X. Li, J. Qin, and J. B. Bernstein. Compact modeling of MOSFET wearout mechanisms for Circuit-Reliability simulation. *IEEE Transactions on Device and Materials Reliability*, 8(1):98–121, Mar. 2008.

[44] M. Lin, A. El Gamal, Y. C. Lu, and S. Wong. Performance benefits of monolithically stacked 3-D FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):216–229, Feb. 2007.

[45] S. Mahapatra. Mechanism of negative bias temperature instability in cmos devices: Degradation recovery and impact of nitrogen. In *Proc. of Int'l Electron Devices Meeting*, pages 105–108, 2004.

[46] J. Maria and N. Serrano. Gbps crc generator in 0.35um cmos technology implemented with standard cells. Technical Report Version 0.6, Lund Institute of Technology, 2002.

[47] R. Michard, A. Tisserand, and N. V. Arenaire. divgen user's and reference manual. Technical Report Version 0.11, CNRS-ENSL-INRIA-UCBL, 2005.

[48] M. Mishra, T. J. Callahan, T. Chelcea, G. Venkataramani, S. C. Goldstein, and M. Budiu. Tartan: evaluating spatial computation for whole program execution. In *ASPLOS*, pages 163–174, 2006.

[49] R. Mishra, D. Ioannou, S. Mitra, and R. Gauthier. Effect of Floating-Body and stress bias on NBTI and HCI on 65-nm SOI pMOSFETs. *Electron Device Letters, IEEE*, 29(3):262–264, 2008.

[50] J. Mitard, X. Garros, L. Nguyen, C. Leroux, G. Ghibaudo, F. Martin, and G. Reimbold. Large-Scale time characterization and analysis of PBTI in HFO2/Metal gate stacks. In *Reliability Physics Symposium Proceedings, 2006. 44th Annual., IEEE International*, pages 174–178, 2006.

[51] S. Mukhopadhyay, H. Mahmoodi, and K. Roy. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(12):1859–1880, 2005.

[52] opencores.org. The no.1 community within open source hardware ip-cores. Website. www.opencores.org.

[53] I. Park, C. L. Ooi, and T. N. Vijaykumar. Reducing design complexity of the Load/Store queue. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, pages 411–422, Washington, DC, USA, 2003. IEEE Computer Society.

[54] S. Paul, S. Mukhopadhyay, and S. Bhunia. A circuit and architecture codesign approach for a hybrid cmos-sttram nonvolatile fpga. *Nanotechnology, IEEE Transactions on*, 10(3):385 –394, May 2011.

[55] M. D. Powell, A. Biswas, S. Gupta, and S. S. Mukherjee. Architectural core salvaging in a multi-core processor for hard-error tolerance. In *ISCA*, pages 93–104, 2009.

[56] PTLsim. X86-64 cycle accurate processor simulation design infrastructure. Website. www.ptlsim.org.

[57] A. Putnam, S. Eggers, D. Bennett, E. Dellinger, J. Mason, H. Styles, P. Sundararajan, and R. Wittig. Performance and power of cache-based reconfigurable computing. In *ISCA*, pages 395–405, 2009.

[58] S. Ramey, C. Prasad, M. Agostinelli, S. Pae, S. Walstra, S. Gupta, and J. Hicks. Frequency and recovery effects in high-k bti degradation. In *Proc. of 47th IRPS*, pages 1023–1027, Montreal, 2009.

[59] R. Razdan and M. D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *MICRO*, pages 172–180, 1994.

[60] R. Riedlinger, R. Bhatia, L. Biro, B. Bowhill, E. Fetzer, P. Gronowski, and T. Grutkowski. A 32nm 3.1 billion transistor 12-wide-issue itanium processor for mission-critical servers. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2011 IEEE International*, pages 84 –86, Feb. 2011.

[61] S. Sahhaf, R. Degraeve, P. J. Roussel, B. Kaczer, T. Kauerauf, and G. Groeseneken. A new TDDB reliability prediction methodology accounting for multiple SBD and wear out. *IEEE Transactions on Electron Devices*, 56(7):1424–1432, July 2009.

[62] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: a model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, Feb. 2008.

[63] P. G. Sassone, J. Rupley,II, E. Brekelbaum, G. H. Loh, and B. Black. Matrix scheduler reloaded. *SIGARCH Comput. Archit. News*, 35(2):335346, June 2007.

[64] S. Sawant, U. Desai, G. Shamanna, L. Sharma, M. Ranade, A. Agarwal, S. Dakshinamurthy, and R. Narayanan. A 32nm Westmere-EX xeon enterprise processor. In *ISSCC*, pages 74–75, 2011.

[65] J. Shin, V. Zyuban, P. Bose, and T. M. Pinkston. A proactive wearout recovery apporoach for exploiting microarchitectural redundancy to extend cache sram lifetime. In *Int'l Symp. on Computer Architecture*, pages 353–362, 2008.

[66] P. Shivakumar, S. W. Keckler, C. R. Moore, and D. Burger. Exploiting microarchitectural redundancy for defect tolerance. In *ICCD*, pages 481–488, Oct. 2003.

[67] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *ISCA*, pages 520–531, 2005.

[68] J. Suh, M. Annavaram, and M. Dubois. Macau: A markov model for reliability evaluations of caches under single-bit and multi-bit upsets. In *Proc of HPCA 2012*, 2012.

[69] M. E. Thomadakis. The architecture of the nehalem processor and nehalem-ep smp platforms. Research report, Texas A&M University, 2011.

[70] A. Tiwari and J. Torrellas. Facelift: Hiding and slowing down aging in multicores. In *MICRO*, pages 129–140, 2008.

[71] R. Usselmann. Advanced encryption standard rijndael ip core. Technical Report Version 1.1, ASICS.WS, 2002.

[72] R. Vattikonda, W. Wang, and Y. Cao. Modeling and minimization of pmos nbti effect for robust nanometer design. In *Proc. of Design Automation Conf.*, pages 1047–1952, 2006.

[73] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: reducing the energy of mature computations. In *ASPLOS*, pages 205–218, 2010.

[74] Y. Wang, U. Bhattacharya, F. Hamzaoglu, P. Kolar, Y. Ng, L. Wei, Y. Zhang, K. Zhang, and M. Bohr. A 4.0 ghz 291mb voltage-scalable sram design in 32nm high-k metal-gate cmos with integrated power management. In *Proc of ISSCC 2009*. ISSCC, IEEE, 2009.

[75] J. Warnock, Y. Chan, et al. A 5.2GHz microprocessor chip for the IBM zEnterprise system. In *ISSCC*, pages 70–72, Feb. 2011.

[76] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-l. Lu. Reducing cache power with low-cost, multi-bit error-correcting codes. In *Proc of ISCA*, page 8393, 2010.

[77] C. Wilkerson, H. Gao, A. R. Alameldeen, Z. Chishti, M. Khellah, and S. Lu. Trading off cache capacity for reliability to enable low voltage operation. In *ISCA*, pages 203–214, 2008.

[78] R. D. Wittig and P. Chow. OneChip: an FPGA processor with reconfigurable logic. In *FPGA*, pages 126–135, Apr. 1996.

[79] H. Wong, V. Betz, and J. Rose. Comparing FPGA vs. custom cmos and the impact on processor microarchitecture. In *FPGA*, page 514, 2011.

[80] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 34–45, 2009.

[81] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee. CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit. In *ISCA*, pages 225–235, 2000.

[82] S. Yehia, N. Clark, S. Mahlke, and K. Flautner. Exploring the design space of LUT-based transparent accelerators. In *CASES*, pages 11–21, 2005.

[83] S. Yehia, S. Girbal, H. Berry, and O. Temam. Reconciling specialization and flexibility through compound circuits. In *HPCA*, pages 277–288, Feb. 2009.

[84] D. H. Yoon and M. Erez. Memory mapped ECC: low-cost error protection for last level caches. In *ISCA*, pages 116–127, 2009.

[85] S. Zafar. Statistical mechanics based model for negative bias temperature instability induced degradation. *Applied Physics*, 97(10), May 2005.

[86] S. Zafar, Y. Kim, V. Narayanan, C. Cabral, V. Paruchuri, B. Doris, J. Stathis, A. Callegari, and M. Chudzik. A comparative study of NBTI and PBTI (Charge trapping) in SiO2/HfO2 stacks with FUSI, TiN, re gates. In *VLSI Technology, Digest of Technical Papers.*, pages 23–25, 2006.