Design of variation-tolerant integrated circuits

Jayakrishnan, Mini

2018

Jayakrishnan, M. (2018). Design of variation-tolerant integrated circuits. Doctoral thesis, Nanyang Technological University, Singapore.

<http://hdl.handle.net/10356/73872>

<https://doi.org/10.32657/10356/73872>

NANYANG
TECHNOLOGICAL
UNIVERSITY

DESIGN OF VARIATION-TOLERANT INTEGRATED CIRCUITS

MINI JAYAKRISHNAN

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University

in partial fulfillment of the requirement for the degree of

Doctor of Philosophy

2017

ACKNOWLEDGEMENTS

This thesis has been in making for the last four years or so. While I take immense pride and joy in being finally able to finish it, this task could not have been accomplished without the many contributions of the wonderful people that I have had the good fortune of being associated with, all these years. I want to take this opportunity to express my sincere and heartfelt gratitude to them.

First and foremost, I would like to thank my advisor, Prof. Kim Tae Hyung. His incredible creativity has been the source of a lot of ideas in this thesis. His capacity for hard work and single-minded focus is inspirational. In the last four years that I have known him, he has taught me by example, how to be a better researcher, writer, speaker and, perhaps more importantly, a better person. I have often turned to him for counsel during difficult times in this dissertation and his support has helped me sustain my focus. He is easily one of the best advisors to have.

Secondly, I would like to thank all thesis review panel members to take time from their busy schedules to be a member of my dissertation committee. In particular, I would like to thank Prof. Jong Ching Chuen, Prof. Goh Wang Ling and Prof. Gwee Bah Hwee to be part of my thesis dissertation committee. Your valuable suggestions and feedback have greatly strengthened the thesis.

I would like to thank Dr. Michael Bolt, Dr. Jose Pineda De Gyvez and Mr. Alan Chang for initiating the project in NXP and for the opportunity to work on various low power test chips. I want to express my gratitude to my colleagues at NXP, particularly Dongjun, who I have worked with very closely in the past four years. I have learned a lot from his vast experience and his creativity. I would also like to thank Shiyang and Rina for the numerous discussions about a broad range of topics, both technical and otherwise. I also thank Arjun, Juhui, Deepak, Kamal and my other colleagues at NXP from whom I have learned a lot about various aspects of chip design.

During my stay at NXP, I have met incredibly talented people and made great friends. They have been a very strong source of support for me and made my graduate career a memorable experience. In particular, I would like to mention Jose, Maurice, Peter, Juan, and countless others who have always been there to share ideas. Special thanks are due to the Beetle design team.

Finally, I would like to thank my family. I am forever indebted to my husband, Jay for his unconditional love and support. He has instilled in me an appreciation of hard work and he receives the most credit for whatever I have accomplished in my life thus far. With his constant encouragement and mentoring, he has made writing this thesis a very pleasant exercise. I also thank my wonderful children Dev and Krithin for their wholehearted support throughout my thesis. They were so understanding and independent which helped me to sail through all the hardships throughout this journey. I would like to express my gratitude to my parents-in-law who stays with me and takes good care of me and my family which helped me concentrate on my thesis all these years. I also thank my loving parents who always think for my good and without their blessings and prayers I couldn't have achieved this task.

ABSTRACT

The semiconductor industry is strategically focusing on automotive markets and significant investment is targeted to address these markets. Run time better-than-worst-case designs like Razor leads to massive timing errors upon breaching the critical operating point and have significant area overheads. As we scale to higher-reliability automotive and industrial markets we need alternative techniques that will allow full extraction of the power benefits without sacrificing reliability. In this work, we propose four resilient processor subsystems which incorporate slack balancing circuits to relax the timing margins of a processor pipeline based on the available slack. Together with logic under-design, the proposed methodology results in power and area savings compared to worst-case design techniques. The presence of slack balancing circuits ensures correct functionality of the design at worst operating conditions.

Based on this, we propose the first subsystem to utilize positive slack available in the pipeline stages using Slack Balancing Flip-flops (SBFFs) and re-distributes it to the preceding error-prone critical paths. SBFF has a redundant latch to sample the late arriving data. We reshape the SBFF fan-in cone by downsizing the logic to achieve power and area savings of 12% and 8%, respectively, as compared to the worst-case design. Our second subsystem uses library based power optimization techniques together with SBFFs. Here we prune the SBFF standard cell library to filter out the power-hungry cells. In addition to this, we use Better than Worst-case (BTWC) sigma corner library to under-design the logic whose slack margins are relaxed by the SBFFs. This gave a power and area savings of 47% and 3%, respectively, in the execute pipeline stage. The third subsystem provides a Clock Stretching Flip-flop (CSFF) to remove the redundancy inside SBFFs for better power and area savings. The combination of CSFFs, multi-bit flip-flops (MBFFs), logic reshaping and BTWC sigma corner libraries resulted in a power and area savings of 32% and 16%, respectively, in the fetch pipeline stage and 69% and 15%, respectively, in the execute pipeline stage when compared to the traditional worst-case design. In the fourth subsystem, we use Data-dependent Clock Stretching Flip-flop (DDCSFF) in addition to simple CSFFs to replace the critical endpoints with sufficiently low activity rates. Here we use logic reshaping for power optimization together with CSFFs and DDCSFFs. Experiment results showed power and area savings of 70% and 9.5%, respectively, in the execute pipeline stage with respect to the worst-case design. For all the proposed subsystems, the critical logic power minimization method relaxes the slack margins of all the timing paths including the short paths converging into the SBFF. The timing budgeting and timing correction using opportunistic slack eliminates critical operating point behaviour, meta-stability issues and hold buffer overheads encountered in existing resilience techniques.

TABLE OF CONTENTS

ABSTRACT.....	ii
LIST OF FIGURES.....	vi
LIST OF TABLES	xi
Chapter 1. INTRODUCTION	13
1.1 Sources of Variations.....	16
1.2 Adaptive Design Approaches.....	17
1.3 The Proposed Opportunistic Under-design Approach.....	20
1.4 Main Contributions and Organisation of Thesis	23
Chapter 2. STATE OF THE ART	26
2.1 Circuit Level.....	26
2.1.1 Always correct	26
2.1.2 Fail and Correct.....	27
2.2 Architecture Level	33
2.3 Algorithmic Level.....	34
2.4 Summary and Discussion.....	36
Chapter 3. SLACK-AWARE DESIGN METHODOLOGY USING TIME BORROWING FLIP-FLOPS FOR VARIATION TOLERANT PIPELINED DIGITAL CIRCUITS	37
3.1 Design Motivation	37
3.2 Module Wise Slack Redistribution.....	38
3.3 Principle of Error Avoidance Flip-flop.....	40
3.4 Proposed EAFF Based Design Methodology	40
3.4.1 EAFF Replacement Strategies.....	41
3.4.2 Design Flow and Algorithm	41
3.5 Results and Analysis.....	46
3.5.1 Slack Improvement	46

3.5.2	Power and Area overheads	47
3.5.3	Better than Worst-case Design Opportunity	49
3.6	Conclusions	50
Chapter 4. A POWER EFFICIENT VARIATION TOLERANT PIPELINE DESIGN METHODOLOGY USING SLACK BALANCING FLIP-FLOPS AND LOGIC RESHAPING		51
4.1	Motivation	52
4.2	Proposed Variation Tolerant Pipeline Design	53
4.2.1	Slack Balancing Principle	53
4.2.2	Slack Balancing Flip-flop (SBFF)	55
4.2.3	Pipeline Design Flow using SBFF	56
4.2.4	Meta-stability and Hold Issues	57
4.3	Power Optimization Algorithm	61
4.4	Results and Analysis	63
4.5	Conclusion	64
Chapter 5. LIBRARY PRUNING AND SIGMA CORNER LIBRARIES FOR POWER EFFICIENT VARIATION TOLERANCE		65
5.1	Proposed Design Methodology	66
5.1.1	Slack Balancing	66
5.1.2	Design Flow	67
5.2	Power Optimization Techniques	68
5.2.1	Library Pruning	69
5.2.2	Sigma corner library cell swaps	69
5.3	Results and Analysis	71
5.4	Conclusions	72
Chapter 6. POWER AND AREA EFFICIENT CLOCK STRETCHING AND CRITICAL PATH RESHAPING FOR ERROR RESILIENCE		73
6.1	Motivation	73

6.2	Proposed Design Methodology	74
6.2.1	Slack Balancing	74
6.2.2	Clock Stretched flip-flop	75
6.2.3	Design Flow	78
6.3	Power Optimization Techniques	79
6.3.1	Power optimization by logic reshaping (SizeOpt)	79
6.3.2	Power optimization by library tuning (SigmaOpt)	80
6.4	Results and Analysis	82
6.4.1	Flip-flop level savings	82
6.4.2	Chip level savings	83
6.5	Conclusions	86
Chapter 7. DATA-DEPENDENT CLOCK STRETCHING FLIP-FLOP FOR POWER EFFICIENT VARIATION TOLERANCE		88
7.1	Motivation	88
7.2	Proposed Design Methodology	89
7.2.1	Slack Balancing	89
7.2.2	Design Flow	92
7.3	Power Optimization Techniques	92
7.4	Results and Analysis	93
7.5	Conclusions	93
Chapter 8. SUMMARY		95
8.1	Summary of Work-done	95
8.2	Future Work	98
Chapter 9. PUBLICATIONS		100
Chapter 10. BIBLIOGRAPHY		101

LIST OF FIGURES

Figure 1. Reliability and Power challenges.	9
Figure 2. Lost performance/energy due to increase in design margins.....	10
Figure 3. Chip life cycle.	10
Figure 4. Types of variations.	11
Figure 5. Spatial & Temporal variation.....	11
Figure 6. Threshold voltage and delay variations.	12
Figure 7. Speculative techniques (a) Razor I error detection with flip-flop as data path [18], (b) Razor II error detection with latch as data path [19], and (c) TIMBER error masking with flip-flop as data path [31]......	13
Figure 8. Different overheads involved in a speculative error resilience scheme.	15
Figure 9. Proposed Variation Tolerant Design (VTD)......	16
Figure 10. (a) Lost performance/energy due to increase in design margins and (b) Proposed approach converts lost performance/energy to power/area savings and keeps worst design margins with respect to the delayed clock edge.	17
Figure 11. Proposed opportunistic design.	18
Figure 12. Proposed Opportunistic under-designed (OUD) processor subsystems.	19
Figure 13. Canary flip-flop.	22
Figure 14. Canary TEA time circuit.	23
Figure 15. Tunable Replica Circuits.....	23
Figure 16. Common Master-slave sequential element topology.	23
Figure 17. Comparison between Normal flip-flops, Error Detection and Error Masking flip-flops.	24
Figure 18. Pipeline stage with (a) normal flip-flop (b) timing diagram and with (c) error detection sequential (EDS) (d) timing diagram.	25
Figure 19. Razor II.	25
Figure 20. Error detection and masking circuits (a) Razor I (b) TDTB (c) DSTB.....	26
Figure 21. Error masking soft edge flip-flop.	26
Figure 22. Triple Mode Redundancy.	27

Figure 23. TIMBER flip-flop.....	27
Figure 24. Architectural error correction using Razor FF.	29
Figure 25. Architectural voltage scaling.....	29
Figure 26. Slack histogram of processor pipeline (a) at worst-case, (b) with 20mV voltage scaling, and (c) with 40mV voltage scaling.	31
Figure 27. Cost of resilience and voltage scaling trade-off.	31
Figure 28. Slack balancing schemes: (a) Retiming (b) Soft edge flip-flop (c) Latch based data path (d) Useful skew (e) Clock skew scheduling and gate sizing and (f) Proposed Error avoidance flip-flop.	33
Figure 29. Critical slack and consecutive slack in a processor pipeline.	34
Figure 30. (a) EAFF architecture and (b) timing diagram.....	35
Figure 31. Simulation results for EAFF.	36
Figure 32. Time borrowing strategies: (a) without time borrowing, (b) single stage time borrowing, coarse grain or fine grain, and (c) multistage time borrowing.....	37
Figure 33. Proposed slack analysis and EAFF replacement flow.....	38
Figure 34. EAFF replacement algorithm.....	39
Figure 35. Clocking strategies for EAFF: (a) same clock delay used for all EAFFs, (b) different clock delays for different EAFF clusters, and (c) different clock delays with individual enable.	40
Figure 36. Design environment for EAFF based pipeline.....	40
Figure 37. Timing speculation window: (a) for Razor flip-flops, max delay is relaxed and min delay is deteriorated by $T_{ck}/2$, (b) for TIMBER, the max delay improvement is less than $T_{ck}/2$, the min delay is deteriorated by $T_{ck}/2$, and (c) for the proposed EAFF, max delay and min delay constraints in each stage is proportional to the consecutive stage slack.	40
Figure 38. Slack improvement with EAFF for processor modules, M1, M2, M3, M4, M7 and M8.....	42
Figure 39. Percentage of paths improved using the proposed time borrowing strategies.	43
Figure 40. (a) Chip power overhead for EAFF design and (b) Chip area overhead for EAFF design.	43
Figure 41. (a) Percentage EAFF replacement (b) Chip area overhead and (c) Chip power overhead for EAFF design.....	44

Figure 42. Slack distribution of processor using BTWC voltage corner for: (c) $V_{dd}=0.97$ V (d) $V_{dd}=0.98$ V.	45
Figure 43. Slack distribution of processor after using BTWC voltage corner for (a) $V_{dd}=0.95$ V (b) $V_{dd}=0.96$ V.	45
Figure 44. (a) critical stage and (b) consecutive stage slack distribution of processor for BTWC voltage corner for $V_{dd}=0.98$ V.	46
Figure 45. Slack distribution of processor for different voltage corners.	46
Figure 46. Design margin Vs error rate for different error resilience techniques.	47
Figure 47. Critical path slacks (pipe_1) and consecutive stage slacks (pipe_2) of the processor.	48
Figure 48. A circuit example and timing graph to explain slack balancing.	50
Figure 49. Speculative power-aware flip-flop: (a) slack margin at sign off, (b) slack margin relaxed with timing speculation flip-flop, and (c) slack margin traded off with power.	50
Figure 50. Performance-aware optimizations: (a) slack margin at sign off, (b) slack margin relaxed with clock skew scheduling and gate sizing (c) slack margin traded off with frequency.	50
Figure 51. Proposed SBFF: (a) slack margin at sign off, (b) slack margin relaxed with SBFF, and (c) slack margin used to downsize logic.	50
Figure 52. (a) Proposed SBFF and (b) timing diagram.	52
Figure 53. Data activity monitoring in SBFF: (a) Intermittent triggering of activity monitor with low data activity, (b) continuous triggering of activity monitor with high data activity.	52
Figure 54. Design flow using SBFF.	53
Figure 55. (a) Speculative flip-flop pipeline, (b) Data path meta-stability, and (c) Error path meta-stability.	54
Figure 56. (a) SBFF pipeline, (b) Data path timing.	54
Figure 57. Cell Downsizing (a) Area Vs Downsizing, (b) Delay Vs Downsizing.	57
Figure 58. SBFF replacement and size based optimization algorithm.	57
Figure 59. Slack improvement in different modules.	58
Figure 60. SizeOpt power optimization technique (a) Power savings, (b) Area savings with respect to worst-case design.	59
Figure 61. Area Vs Speed trade-off for different process conditions.	61

Figure 62. Power comparison between master-slave flip-flop (MSFF) and error tolerant slack balancing flip-flop (SBFF) standard cells.....	61
Figure 63. Critical path slacks and consecutive path slacks in the execute (EXE) pipeline stage module M1 of a processor pipeline.	62
Figure 64. A processor pipeline with SBFF and BTWC sigma corner logic.	62
Figure 65. (a) SBFF architecture and (b) timing diagram.	63
Figure 66. Proposed design flow with BTWC sigma corner logic and SBFF.....	64
Figure 67: (a) Power comparison between MSFF and SBFF standard cells and (b) Area comparison between MSFF and SBFF standard cells.....	64
Figure 68. (a) Delay comparison between MSFF and SBFF standard cells and (b) Setup time comparison between MSFF and SBFF standard cells.	65
Figure 69. (a) Relaxed design margins with sigma corner libraries and (b) Delay reduction with sigma scaling and voltage scaling.....	66
Figure 70. SBFF replacement and sigma based optimization algorithm.....	66
Figure 71. SigmaOpt and Library Pruning power optimization techniques (a) Power savings, (b) Area savings with respect to worst-case design.....	68
Figure 72. (a) Critical path slacks (pipe_1) (b) retimed pipe_1 slack and (c) consecutive stage slacks (pipe_2) of the processor.	69
Figure 73. A processor pipeline with CSFF and under designed logic.....	70
Figure 74. Redundant nodes in typical error resilient flip-flops.....	71
Figure 75. (a) <i>CSFF</i> [45] architecture (b) timing diagram and (c) Multi-bit flip-flop structure.	72
Figure 76. Multibit flip-flop.....	73
Figure 77. Multi-bit flip-flop synthesis flow.	73
Figure 78. Proposed design flow with CSFF.....	74
Figure 79. Pseudo code for slack analysis and reshaping.....	75
Figure 80. (a) Area Vs Speed trade-off for different process conditions and (b) Delay reduction with sigma scaling and voltage scaling.....	76
Figure 81. Pseudo code for sigma corner based optimization.	77
Figure 82. Area reduction compared to TIMBER flip-flop.....	78
Figure 83. Path relaxation in different modules of a processor pipeline.....	79

Figure 84. (a) Power comparisons and (b) Area comparisons between WC baseline, <i>TIMBER</i> [31] and the proposed schemes.	81
Figure 85. (a) Power comparisons and (b) Area comparisons between WC baseline, <i>TIMBER</i> [31] and the combined optimizations.	81
Figure 86. Chip layout.....	82
Figure 87. (a) Flip-flop activity factor for fast_fft (b) Flip-flop activity factor for simple fft, fft32, fft64 and fft128.	84
Figure 88. Slack balancing using CSFF and DDCSFF.	85
Figure 89. (a) <i>CSFF</i> [45] architecture (b) timing diagram.	86
Figure 90. (a) <i>Data-dependent CSFF</i> architecture (b) timing diagram.....	87
Figure 91. Design flow for slack and data analysis and reshaping.	87
Figure 92. Pseudo code for slack and data analysis and reshaping.....	88
Figure 93. Proposed power optimization technique: (a) Power savings, (b) Area savings with respect to worst-case design.	90
Figure 94. Chip optimizations at design time and runtime.....	91
Figure 95. Slack histogram vs. voltage scaling.....	94
Figure 96. Consecutive stage slack for voltage scaling.....	94
Figure 97. Frequency scaling opportunity.	95

LIST OF TABLES

Table 1. Summary of error resilience techniques.....	32
Table 2. Various error detection schemes comparison.....	32
Table 3. Various error correction schemes comparison.	33
Table 4. Various algorithms comparison.....	34
Table 5. Coarse grain TB map shows number of critical paths before and after EAFF insertion.	38
Table 6. Summary of error resilience techniques.....	60
Table 7. Power and Area savings of the proposed design methodology.....	64
Table 8. SBFF flavors used in the proposed design methodology.....	68
Table 9. Power savings of the proposed design methodology.....	72
Table 10. Area savings of the proposed design methodology.	72
Table 11. Comparison of error resilience techniques.....	74
Table 12. Max power saving for quad bit MBFF.	78
Table 13. Power delay savings of Proposed circuit	82
Table 13. Power delay savings of Proposed circuit	83
Table 14. Power savings of the proposed design methodology.....	83
Table 15. Area savings of the proposed design methodology	83
Table 14. Power savings of the proposed design methodology.....	83
Table 15. Area savings of the proposed design methodology	83
Table 16. Power savings of the combined optimizations	84
Table 17. Area savings of the combined optimizations	84
Table 18. Chip specifications.....	86
Table 19. Power savings of the proposed optimization.	94
Table 20. Area savings of the proposed optimization.....	94
Table 21. Power and Area comparisons of the proposed optimizations.	95

Table 22. Power and Area comparisons of the proposed optimizations.	95
Table 23. Comparison summary of the proposed optimizations with state of art.....	97

CHAPTER 1. INTRODUCTION

The GHz race has been taken over by the nano-watt race. It all started with electrons and it looks like we are going back to electrons. If we look at the growth of semiconductor devices from the point contact transistor to the CMOS, we are now striving towards nano and molecular devices and technologies. This evolution was a result of both market pull and technology push driven by Moore's Law. The traditional device is working on its performance edge. At the same time, other contemporary technologies are evolving fast. Nevertheless, I believe I am lucky to be born in an era of semiconductors and I have seen the evolution, saturation, and trade-offs of it. Through this research, I am part of the community trying to contribute to one of the greatest inventions of mankind. Semiconductor devices were big in the beginning because of which everyone tried to reduce the size and enhance performance. Power consumption issues were not on the top of our mind. With new market demands and green energy revolution, the super big devices with high power consumption have shrunk into portable/wearable ultra-low power nanodevices as shown in Figure 1. The need for high performance, high density, and low power drove the market towards scaling which is now being pushed to the limits and the returns are diminishing. Until now everything was in harmony. With scaling, we could push more devices in one chip thus reducing the wire lengths, component sizes, and supply voltage which resulted in lower power. With scaling, we achieved our area, performance, cost and power targets to a greater extent. But market demands are never ending, so we have to bear with the side effects of scaling and try to make the devices reliable at the same time meeting power, performance and area targets.

Figure 1. Reliability and Power challenges.

The need for high performance led to aggressive scaling of transistors. With device scaling, the supply voltage also scaled and we could get power, performance and area gains to some

Figure 2. Lost performance/energy due to increase in design margins.

extent. Sizing reduces capacitance, and thus give a big energy saving. But with reduced supply voltage, gate delay increased and that caused power performance trade-off. With the advent of low power and ultra-low power devices in the market, the focus of designers was shifted to low power designs. This led to multi Vdd approaches, where timing critical paths get high voltage and others operate at a low voltage. Clock gating emerged as a technique to reduce dynamic power by reducing activity factor. But as we started to scale down further, the lower threshold voltage resulted in high leakage power. The leakage power consumption is increasing at a faster rate and is at par with dynamic power for modern VLSI circuits. Low threshold voltage leads to less delay but higher leakage, and high threshold voltage contribute to low leakage but high delay. This resulted in multi-threshold and variable threshold CMOS. Low power designs are shifting from subthreshold to near-threshold to maximize the gains. We need to design and optimize each gate individually according to power and performance requirements at a fine grain level.

If we look at the evolution of integrated circuits, the traditional chips used were designed to handle worst-case variations. In other words, they were overdesigned, so that we could use it reliably in silicon. The resulting power, performance and area overheads were not a big issue until now. But, with scaling, the nature of variations has changed from static to temporal, as well as the spatial outreach of variations has moved into Within-Die, rather than Die-to-Die. Thus, the predictability of variations has reduced significantly over the years with tremendous scaling. In addition to this, the spread of variations has also increased through years, with leakage spread being larger than frequency spread. With stringent power budget requirements driven by portables and wearables, the focus of IC designer has shifted significantly from performance to power targets. Moreover, variations in advanced technology nodes at

Figure 1. Chip life cycle.

subthreshold as well as near-threshold voltages tends to affect the yield of chips to a great extent.

Cost-effective variation tolerance is hard to achieve in nanometer scale technology nodes [1], [2]. Manufacturing process limitations in advanced nodes cause random dopant fluctuations and line edge roughness [3] - [5] which creates the need for additional process margins. High device density in nanoscale chips leads to an increase in voltage and temperature

Figure 2. Types of variations.

Figure 3. Spatial & Temporal variation.

variations [6] which widens the design margins further. As a result, the chip life cycle has shrunk tremendously as shown in Figure 1. Moreover, the magnitude of delay variations in the near-threshold regime is exponential with respect to supply voltage. The impacts of variations continue to increase with process technology scaling, which leads to pessimistic delay margins in processor pipelines. This stretches the timing guard bands in traditional designs as illustrated in Figure 2. The worst-case design margins result in overdesign and wastage of chip resources. Therefore, design techniques for improving the chip yield without compromising performance and energy efficiency are highly demanded.

1.1 Sources of Variations

Different types of variations can occur in an IC chip as shown in Figure 4. Spatial process variations are static in nature. Such variations can be Within Die and Die-to-Die. There can be Wafer-to-Wafer variations where the operating corners of two wafers will be different. Spatial variations include global lithographic variations resulting from different device dimensions from chip to chip. It can be local intrinsic material variations resulting from random dopant fluctuations (RDF) and Line Edge Roughness (LER). Variation in device dimensions like L , W , and t_{ox} results in parametric V_{th} variation which affects the power and performance. Temporal variations include environmental and aging-related variations. The environmental variations are temperature and voltage dependent. Temperature variations are a result of load characteristics & switching activity changes in neighbouring blocks. Voltage variations are a result of the transient fluctuation in switching activity, Ldi/dt drop, and steady state current IR drop. Aging is caused by physical changes due to movement of charges & breaking of atomic bonds. Typical aging-related variations are Negative/Positive Bias Temperature Instability (NBTI/PBTI), Hot Carrier Injection (HCI), Time Dependent Dielectric Breakdown (TDDB) and Electro-migration. TDDB is caused by creation and joining of defects in dielectric due to the thin gate. HCI is a result of the defect in gate stack by high energized carriers causing shift in threshold-secondary emission. BTI is due to capturing of holes/electrons from the inverted channel by broken SiH bonds delay. Figure 5 illustrates the graphic description of spatial and temporal variation.

Figure 4. Threshold voltage and delay variations.

Device parameters like delay and leakage are impacted due to the process, voltage and temperature variations. Process variations affect device dimensions and threshold voltage. Voltage and temperature fluctuations affect threshold voltage and mobility. Temperature variations reduce semiconductor band gap which increases device leakage exponentially. Process variations affect threshold voltage which in turn affects leakage. Variations impact product in the form of reliability, yield and power budget. Reliability issues result in hard functional failures and soft timing failures during operation. Yield is affected due to the failure to meet chip specifications during the manufacturing phase. Power budget is not met due to the failure to meet leakage power constraints which affects battery life and battery size. The demand for low power and high performance has led to aggressive voltage scaling and

technology scaling. As a result, the sigma spread of variations has increased significantly, with leakage spread being larger than delay spread.

Figure 6 shows the threshold voltage variation and the resulting delay variation which worsen with technology scaling. The delay variation double with voltage scaling and Within-Die variations are exponential with logic depth. The predictability of variations has become significantly challenging in the nanometer scale with Random Dopant Fluctuations (RDF) and aging of devices. We need to tackle the multi-dimensional trade-offs for an efficient variation tolerant design.

1.2 Adaptive Design Approaches

Several techniques have been proposed in the literature to achieve cost-effective error tolerance. Static variability is dealt with operating point tuning [7], [8], and [9]. Post-silicon tuning and adaptive techniques help to combat static and dynamic variations and recover the target design margins [10], [11]. Sensors and monitors are used to generate a hardware signature of variations that is then used in higher abstraction layers to tune the operating point [12], [13] and [14]. However, sensors may not be able to respond fast enough to dynamic variations. This limits the amount of design margin improvement and prevents the utilization of sensor-based error tolerance compared to the worst-case based design techniques. In addition, sensors need to be calibrated to determine the safe region of operation with the additional cost of post-silicon testing. Error prediction techniques like Canary [15] and TEA time [16] predicts the timing failures due to variations. Critical path replica [17] is a popular non-intrusive technique where variations are detected by monitoring the duplicated critical paths thus removing short path timing issues. However, they fail to detect the actual variations due to the mismatches between the critical paths and the replica paths. Additional margin should be incorporated in the design to address the failures coming from the mismatches.

Performance variations, whether static or dynamic, are manifested as timing errors in the critical paths of a system. Many in-situ error detection techniques as shown in Figure 7 are available to detect these timing errors with the help of special flip-flops as error detection elements. In-situ error resilience techniques compare the input data with a delayed version to

Figure 5. Speculative techniques (a) Razor I error detection with flip-flop as data path [18], (b) Razor II error detection with latch as data path [19], and (c) TIMBER error masking with flip-flop as data path [31].

decide whether a timing error has happened. The operating point of the silicon is then tuned dynamically to regain the design margins until the error rate exceeds the threshold limit. Here the error detection window decides the design margin improvement. Max delay constraint of the flip-flop is relaxed by the error detection window and the min delay constraint is tightened by the same amount to prevent short paths from corrupting the data and the error signal. This max delay min delay trade-off is a major design challenge when using in-situ error resilience. They suffer from meta-stability issues since the data is allowed to change too close to the clock edge. They also have huge error correction overheads. Razor I [18] uses a flip-flop in the data path and a latch in shadow path to generate timing error signal. It suffers from meta-stability issues since the data is allowed to change too close to the clock edge. Meta-stability detector, short path buffer and MUX overheads make this technique power hungry. In Razor II [19] the meta-stability issue is avoided using a latch based data path. But latch makes the data path transparent to glitches and spurious transitions. Also, there is significant hold buffer and duty cycle control overheads. Double sampling with time borrowing (DSTB) [20] is similar to Razor I, with flip-flop as the data path. Transition detection with time borrowing (TDTB) is similar to Razor II with a latch as the data path. Another approach Bubble razor [21] breaks the dependency between speculation window and min delay using a two-phase latch implementation but has area and power penalties. In-situ techniques perform the error correction at the architecture level using instruction replay or counter-flow pipelining. This adds a latency of few clock cycles for error recovery and makes the error path design constraints very stringent. Also, they suffer from sharp design margin deterioration with increasing error rate due to the error correction latency to recover from the critical wall of slack behaviour. All these techniques need to periodically tune the processor using the actual error rates. Further, the high instruction per cycle (IPC) penalty for error recovery during heavy workloads makes the whole scheme non-scalable. Meta-stability recovery also consumes ~2 to 3 clock cycles. Several other methods are introduced in the literature which can be used for error detection [22] – [29].

The aforementioned flip-flops detect local dynamic variations but has huge latency overheads due to the architecture level error correction. Soft edge flip-flops [30] and TIMBER [31] helps to mask errors by borrowing time from the next pipeline stages. Compared to the error detection methods, these flip-flops do not have latency for error masking, which makes them good candidates for performance-driven designs. Soft edge flip-flops can reclaim only small design margins since the internal clock control circuit limits the degree of softness. TIMBER uses discrete time borrowing coupled with error propagation logic to detect multistage timing errors. The design margin reduction is limited because the time borrowing window is divided into smaller intervals to deal with errors in multiple stages. TIMBER based pipelines need tight design margins in the last pipeline stage to absorb the time borrowed which makes it difficult to under-design the whole chip. Also, the system frequency needs to be halved to recover from multistage errors beyond three stages. Once the design has error resilience elements in the critical path, better-than-worst-case (BTWC) design techniques [32] can be used to relax the circuit operating point to give power and area savings. BTWC techniques tune the operating point of the chip to the typical corner together with error resilience elements in the critical path to detect and correct the dynamic errors.

Processor pipelines have the critical wall of slack behaviour. This leads to a rapid surge in critical paths with dynamic operating point tuning which makes the cost of resilience even higher. Razor techniques deal with high error rates by reducing the system frequency. TIMBER shares a fixed design margin among multiple pipeline stages. Also, the system frequency needs to be halved to recover from multistage errors beyond three stages. So, we cannot guarantee the savings at high error rates. This may counter effect the very cause of introducing such error detection elements and affects the energy efficiency and throughput of the design. The existing error correction circuits use redundant data sample to check for timing errors. This redundancy creates more switching nodes thus increasing the power consumption and the area. Also, error management, meta-stability and hold buffer overheads are significant.

Figure 8 summarises the overheads involved in a speculative error resilience scheme.

Design margins are also recovered by redistributing slack to the critical pipeline stages using combinatorial and sequential optimizations. Sequential optimizations like retiming help to minimize clock period, achieve low power and maximize tolerance to variations. However, there are restrictions on the space of possible retiming and may result in unnecessary area

Figure 6. Different overheads involved in a speculative error resilience scheme.

overheads. Useful skew and clock skew scheduling combined with gate sizing help to compensate variations and speeds up the processor pipelines. *EVAL (Environment for Variation-Afflicted Logic)* speeds up the timing of critical paths through *Adaptive Body Bias (ABB)* as well as *Adaptive Supply Voltage (ASV)* [33] but has significant area overheads. Another work *Blue Shift* uses *On-demand Selective Biasing (OSB)* and *Path Constrained Tuning (PCT)* [34] to achieve performance gains at the cost of significant power overheads. Design optimizations worsen the critical operating point behaviour of the design [35] which limits the effectiveness of voltage scaling. Power-aware slack distribution (*SlackOptimizer*) [36], [37], [38] use cell sizing together with Razor voltage scaling to distribute slack evenly in a power and cost- efficient manner. Sequential optimizations like retiming and clock scheduling [39] - [43] are also part of traditional design flows. Retiming repositions the flip-flops so that they balance the slack between the pipeline stages more effectively. Conventional retiming procedures can lead to an increase in the number of registers leading to power and

area overheads. Clock skew scheduling adjusts the clock delay of flip-flops at selected locations where there is sufficient slack. Moreover, the existing optimizations based on retiming [22], skew scheduling [23] and gate sizing [25] does not fully reclaim the available slack to be traded off with power/performance. They also limit the amount of time borrowed due to hold violations.

1.3 The Proposed Opportunistic Under-design Approach

Figure 7. Proposed Variation Tolerant Design (VTD).

Traditional IC's follow Worst-case Design (WCD) to address pessimistic variations by employing conservative guard bands in operating voltage and frequency. However, providing enough guard bands becomes unacceptably expensive in deeply scaled process technology. In addition, the nature of variations has changed from static to temporal and die-to-die to within-die which requires additional guard bands. We need optimal solutions to tackle reliability requirements with a focus on the cost of reliability in terms of area and power. The traditional design methods are suboptimal in this regard and more focus should go into opportunistic design availing the underutilized resources inside the IC chip for better energy efficiency. With variation tolerant design (VTD), we aim to improve the chip reliability. The proposed Variation Tolerant Design gives better power, area, and performance for the same reliability margins as the Worst-Case Design (WCD). Also for the same power, area, and performance, the design reliability can be improved further compared to the WCD as shown in Figure 9.

In the current semiconductor market, PPA benefits play a big role in the success of the product. There are several rooms for optimization in a typical chip design. In a semiconductor design, only a subset of timing paths may be critical. Even among those critical paths, some may be active some others may not. So, when we design and optimize, we should be able to prioritize to dedicate more resources to the real critical paths and less focus of the non-critical ones and the critical inactive ones. On the wafer level and chip level, not all regions may face the worst variations which result in unnecessary wastage of chip resources. These factors forced designers to focus on better than worst-case (BTWC) design techniques. Such methods focus on operating the chip at typical operating corner i.e. use a less pessimistic process corner at a lower voltage and at a higher frequency. They need special in-situ error handling circuits

Figure 9. Proposed opportunistic design.

available slack. The logic under-design delays the short paths as well and thus eliminates hold buffer overheads. The redundancy inside SBFF helps to determine the data activity rate of the input signal. Our slack analysis on a processor pipeline demonstrates that the proposed method improves the slack of 85% of the critical paths by 11% of the clock period. This slack improvement enabled us to incorporate opportunistic under-design approaches in the processor pipeline.

1.4 Main Contributions and Organisation of Thesis

Figure 10. Proposed Opportunistic under-designed (OUD) processor subsystems.

We aim for an opportunistic design-time approach to aim for a low power resilience technique rather than using unpredictable and power-hungry runtime resilience. Infrastructure requirements for runtime error resilience far outweigh the benefits. Besides runtime techniques has meta-stability issues. Since the system is not pre-characterized for the dynamic operating corner, meta-stability, hold buffer, and the wall of slack issues occur and leads to system crash. Our work utilizes the slack imbalances inherent in processor pipelines to strengthen the critical paths. We use static timing analysis (STA) to find positive slack paths which immediately succeeds the critical paths. We use slack balancing flip-flop (SBFF) and a simplified clock stretched flip-flop (CSFF) to redistribute this slack to the critical stage. This enables us to redesign the combinational fan-in logic of the critical path proportionate to the extra slack margin. The relaxed slack margins result in power and area savings in the critical fan-in logic cone. In this paper, we propose two power optimization techniques to redesign the critical stages of a 40nm processor pipeline. One approach uses path optimization directives to adjust the timing slack during the synthesis stage. The second approach uses BTWC sigma corner

library to resign critical modules with sufficient consecutive slack in the processor pipeline. The slack apportioning is done in such a way that pessimistic design margins are maintained with respect to the stretched clock edge. The whole scheme is non-speculative and does not involve any runtime voltage or frequency scaling. This guarantees power and area savings without high error rates associated with speculative techniques. Timing is closed at design-time keeping the worst design margins with respect to the delayed clock edge which avoids any meta-stability issues in the design.

The major contributions of this thesis are as follows.

- 1) A novel design method is proposed to provide a larger design margin compared to TIMBER and Razor using a TIMBER like slack balancing flip-flop (SBFF). We propose various time borrowing strategies to fully utilize the available slack. We also analyse the power/area overheads involved in using the slack balancing flip-flop.
- 2) We propose a non-speculative design methodology based on slack balancing flip-flops and critical fan-in logic optimization to reclaim power and area from a conventional worst-case processor pipeline design. We introduce the power optimization technique using the combinational logic resizing. Here we relax the timing of paths with sufficient slack and resynthesize them to achieve power and area savings. The proposed approaches downsize the short path logic along with the critical paths which removes hold buffer overheads. The non-speculative nature of our approach removes error management and latency overheads with the least possible cost for error resilience. This proves to give excellent power/area savings over the previous slack balancing flip-flop designs.
- 3) We propose library-based power optimization techniques to reduce the circuit level overhead. It introduces two techniques, sigma corner library and library pruning to get desired power and area savings. We propose a library pruning technique to filter out the most power consuming cells from the SBFF library. Redundant data sampling creates additional switching nodes in these flip-flops which doubles the internal power for certain flip-flop cells. We get considerable power saving when we discard those outlier cells from the design flow. We use statistical sigma corner libraries to relax the critical fan-in logic of the SBFFs to utilize the extra design margins. We replace the worst-case 3sigma library cells which are characterised for 10% voltage variation with a BTWC 1sigma library characterised for 5% voltage variation. The results show considerable power and area saving.
- 4) We introduce a simplified clock stretched flip-flop (CSFF) to redistribute this slack to the critical stage. We also propose a pipeline technique, which combines all the power optimization concepts and shows overall power/area savings. It will also show how multi-bit clock stretching flip-flops will give additional power and area savings. Results show that clock stretching along with the combination of power optimization techniques gives the best savings so far.
- 5) We propose a data-dependent clock stretching flip-flop for specific critical paths. It has better power reduction capabilities than a simple clock stretched flip-flop when the data activity is low but at the expense of some area. So, we use our algorithm to replace the less activated critical flip-flops using the proposed data-dependent flip-flop. This gives optimal power savings and does not result in considerable area overhead.

Figure 12 summarizes the proposed variation tolerant processor subsystems.

The remainder of this thesis is organized as follows. Chapter 2 presents state of the art. Chapter 3 presents slack-aware design methodology using time borrowing flip-flops for variation tolerant pipelined digital circuits design motivation. Chapter 4 describes a non-speculative variation tolerant pipeline design methodology using slack balancing flip-flops and logic downsizing. The library pruning and sigma corner libraries for power efficient variation tolerance are illustrated in Chapter 5. Chapter 6 describes a power and area efficient clock stretching and critical path reshaping for error resilience and Chapter 7 details data-dependent clock stretching flip-flop for power efficient variation tolerance. Conclusions are given in Chapter 8. Chapter 9 mentions publication list and Chapter 10 the references.

CHAPTER 2. STATE OF THE ART

Due to the increase in process, voltage and temperature variations with technology scaling, it is more challenging to secure IC reliability. Cost of resilience in the form of area, power, and performance is a big concern for today's energy efficient and performance-centric designs. To deal with the multi-dimensional trade-offs in chip design, careful optimizations are required at all design levels such as algorithm, architecture and circuit levels. We need a variation tolerant infrastructure at the circuit level to monitor any deviation from the normal operation. On other words, all the variation induced errors are detected at this level. A resilient architecture provides a feedback mechanism such as voltage/frequency scaling to tune the operating conditions according to the extent of variations. Also, the error management or error correction is done at the architecture level to flush out the erroneous state and bring the system back to the error-free state. Algorithms help to balance the cost of resilience, power and performance trade-offs by exploiting under-design opportunities in the design. In this section, we will briefly summarize the state-of-the-art error resilience techniques.

2.1 Circuit Level

2.1.1 Always correct

Figure 11. Canary flip-flop.

Chips do not always face worst operating conditions. Some techniques employ static variability tuning to operate the chip under process variations [7], [8] and [9]. Such chips go through post Si tuning to deal with process variations and they do not adaptively tune with time and workload. Several alternatives have been proposed to dynamically tune the operating conditions in real-time to recover the design margins. Sensor-based adaptive techniques help to combat static and dynamic variations [10], [11]. The hardware signature from sensors is used to tune the chip operating point [12], [13] and [14]. However, sensors may not be able to respond fast enough to dynamic variations. In addition, sensors need to be calibrated to determine the safe region of operation with additional cost of post-silicon tuning. Several error prediction techniques have been proposed in the literature which can predict circuit failure.

Figure 14. Canary TEA time circuit.

Figure 15. Tunable Replica Circuits.

Canary flip-flop [15] shown in Figure 13 has a delayed buffer in the data path. It predicts failure with voltage/frequency tuning. Canary TEA time circuits [16] shown in Figure 14 tunes the operating frequency until just before failure. This is a non-intrusive technique and the pipeline design need to keep worst-case design margins. This method has significant area overhead and cannot replicate local variations. Tuneable Replica circuit [17] in Figure 15 calibrates and tracks local critical path delays. Here the replica paths are monitored for variations. However, they fail to detect the actual amount of variation due to the mismatch between the actual critical path and the replica path.

2.1.2 Fail and Correct

One of the common sequential element topologies is shown in Figure 16 which is a master-slave edge triggered flip-flop. For the positive clock edge, the internal node will be transmitted to output, at all other states o/p will be latched to the previous value. This circuit will work efficiently until a timing violation occurs, in which case we need to detect the error as well as find ways to predict or avoid the error. So, we need additional elements that can help the circuit to function correctly even after timing violation.

Figure 12. Common Master-slave sequential element topology.

Timing speculation at the circuit level is highly demanded to mitigate timing errors. Various research works have reported circuit techniques which uses special flip-flops that can detect, predict and mask timing errors. These flip-flops can be categorized as Error Detection Sequential (EDS) circuits and Error Masking Sequential (EMS) circuits. The degree of change depends on whether a redundant element is used in the existing flip-flop design. Figure 17

Figure 13. Comparison between Normal flip-flops, Error Detection and Error Masking flip-flops.

shows the comparison between normal flip-flop, error detection flip-flop, and error masking flip-flop.

Error detection circuits consist of conventional flip-flop logic path and an error path that is a shadow path for capturing input data using a delayed clock. Timing error is detected by comparing the data from the conventional flip-flop path and that from the error path. Figure 18 shows how dynamic variations can be detected by adding an error detection circuit in the pipeline stage. Razor I [18], Razor II [19], DSTB and TDTB [20] Bubble Razor [21], belong to this category. Razor I use a flip-flop in the data path and a latch in shadow path to generate timing error signal. It suffers from meta-stability issues since the data is allowed to change too close to the clock edge. Meta-stability detector, short path buffer and MUX overheads make this technique power hungry. In Razor II, the meta-stability issue is avoided using a latch based data path. But latch makes the data path transparent to glitches and spurious transitions. Also, there is significant hold buffer and duty cycle control overheads. Double sampling with time borrowing (DSTB) is similar to Razor, with flip-flop as the data path. Transition detection with time borrowing (TDTB) is similar to Razor II with a latch as the data path. Bubble razor breaks the dependency between speculation window and min delay using a two-phase latch implementation but has area and power penalties. In-situ techniques do the error correction the architecture level using instruction replay or counter-flow pipelining. This adds a latency of

few clock cycles for error recovery and makes the error path design constraints very stringent. Also, they suffer from sharp design margin deterioration with increasing error rate due to the error correction latency to recover from the critical wall of slack behaviour. Figure 19 shows the Razor II circuit. Figure 20 show some of the error detection and error masking circuits.

Error masking circuits mask timing errors with extra time borrowed from consecutive stages. They are mostly latch based like TDTB and DSTB. Some have additional clock control block that provides a soft edge for the data input. The width of this time borrowing window can be adjusted by the clock control circuitry. Soft Edges flip-flops [30] in Figure 21 belong to this category. Triple Modular Redundancy [22] replicates the critical register and predicts failures as shown in Figure 22. But they suffer from considerable area, power and performance overheads. Several other error detection circuits exist which detects errors after clock edge [23], [24], [25], [26] and [27]. For TIMBER flip-flops [31] in Figure 23, if there is enough slack in the subsequent stages, timing violations can be automatically recovered without the need for complex error recovery mechanisms. Compared to error detection methods, there is no latency overhead for error masking which makes it a good candidate for performance-driven designs. Soft edge flip-flops are able to reclaim only small fixed design margins. Here the clock control circuit is internal to the flip-flop which limits the degree of softness attained. TIMBER uses discrete time borrowing coupled with error propagation to detect multistage timing errors.

Figure 18. Pipeline stage with (a) normal flip-flop (b) timing diagram and with (c) error detection sequential (EDS) (d) timing diagram.

Figure 19. Razor II.

Here the design margin reduction is limited because the time borrow window is divided into smaller intervals to deal with multi-stage errors. Also, they suffer from significant error propagation and hold buffer overheads.

Figure 20. Error detection and masking circuits (a) Razor I (b) TDTB (c) DSTB.

Figure 21. Error masking soft edge flip-flop.

Better than worst-case design techniques can be used to tune the operating point of the chip to the typical corner together with error resilience elements in the critical path to detect and correct the dynamic errors. Processor pipelines have the critical wall of slack behaviour. This leads to a rapid surge in critical paths with dynamic operating point tuning which makes the cost of resilience even higher. Razor techniques deal with high error rates by reducing the system frequency. TIMBER shares a fixed design margin among multiple pipeline stages. Also, the system frequency needs to be halved to recover from multistage errors beyond three stages. The circuit level resilience adds power overheads to ICs, which should be less than the power recovered from voltage scaling for energy efficiency. Table 1 summarises the overheads in typical speculative resilience techniques.

Table 2 shows the comparison between various error detection schemes in terms of meta-stability and hold overheads. Razor I need meta-stability detector and has related power and area overheads. This intrusive technique has 36% clock energy overhead with respect to MSFF, 145% clock energy overhead with respect to latch. As far as delay is concerned, it has a D to Q delay overhead of 12% with respect to MSFF and 62% d to q delay overhead with respect to latch. Razor II, Bubble razor, TDTB and DSTB use limited time borrowing and are latch based. Bubble Razor has two phase latch timing complexity. Besides they need bubble propagation algorithm. They have energy improvement of 60%, and throughput gain of 80%

Figure 14. Triple Mode Redundancy.

Figure 15. TIMBER flip-flop.

with an area overhead of 21%. They do not have hold timing constraint. DSTB has an error detection MSFF. They suffer from error path meta-stability, min delay related buffer overhead.

They are intrusive with 34% clock energy overhead with respect to MSFF, 143% clock energy overhead with respect to latch. D to Q delay with respect to MSFF is -12%, and with respect to latch 27%. They need duty cycle tuning for min delay constraints. They do not suffer from data path meta-stability and they have 22% energy reduction and 41% throughput gain. Their implementation is simpler than TDTB but have higher clock energy than TDTB. Timber on the other hand masks timing errors and is flip-flop based. It is more flexible which uses time-borrowing only when there are timing violations.

Table 1. Summary of error resilience techniques.

<i>Technique</i>	<i>area overhead</i>	<i>throughput penalty</i>
TRC	1%	difficult to tune
Razor	4-6%	depend on error recovery
Canary	4-6%	depend on error recovery
Timber	4-6%	no penalty

Table 2. Various error detection schemes comparison.

<i>Circuit</i>	<i>Data Path</i>	<i>Error Path</i>	<i>Meta-stability</i>	<i>Hold Constraint</i>	<i>Time Borrowing</i>
Razor I	MS FF ^a	Shadow Latch	Yes	Yes	No
Razor II	Latch	WG ^b & TD ^c	No	Yes	Possible
Bubble	2 phase Latch	MS FF	No	No	Possible
TDTB^d	Latch	WG & TD	No	Yes	Possible
DSTB^e	Latch	MSFF	No	Yes	Possible
Timber	MSFF	Shadow Latch	No	Yes	Yes

^a. MSFF Master-slave Flip-flop.

^b. WG Window Generator.

^c. TD Transition Detector.

^d. TDTB Transition Detection with Time Borrowing

^e. DSTB Double Sampling with Time Borrowing

2.2 Architecture Level

For error detection circuits, once an error is detected at the circuit level, it needs to be corrected at the architecture level as shown in Figure 24. The error correction circuits use architectural replay and counter-flow pipelining to recover from the erroneous state, which incurs substantial latencies and lead to throughput degradation. The error signals from the different flip-flops need to be combined to generate the control signal. The error signal timing is crucial in proper error recovery. This increases the design effort because of the need to deal with additional timing closures. The error correction latency usually ranges from ~ 5 clock cycles to ~ 10 clock cycles which is significant for a performance-driven design. The critical wall of slack behaviour leads to a rapid rise in the error rates. Architecture level solutions may utilize voltage and frequency scaling to reduce very high error rates as shown in Figure 25. But if a non-critical operation is followed by a critical one, we need fast response rate from the

Figure 24. Architectural error correction using Razor FF.

Figure 25. Architectural voltage scaling.

Table 1. Various error correction schemes comparison.

<i>Architecture</i>	<i>Complexity</i>	<i>Latency</i>
TDTB/DSTB architecture	Error consolidation and adaptive clock control	$\sim$ more than 5 clock cycles
Razor Lite	Less	~ 10 clock cycles
Razor voltage scaling	Voltage regulation	Voltage regulator response too slow
Razor clock gating	Less	Worst-case delay of half system frequency
Razor Counter Flow Pipelining	Flush and bubble generation overhead	~ 5 clock cycles
TIMBER	Error propagation overhead	0 clock cycles

operating point tuning mechanism failing which the system may go to an erroneous state. In *TIMBER* based error resilience [31], a timing error masking technique is introduced which avoids the need for complex architectural recovery and throughput loss. But it has the additional overhead of error propagation logic for multistage time borrowing. Also, the final write back stage of the pipeline should be able to absorb all the time-borrowing which happened in the previous stages. Table 3 shows various error correction scheme comparisons. They are evaluated based on the architectural complexity and latency involved in error correction.

2.3 Algorithmic Level

Several algorithmic level techniques [33] - [43] for cost-effective error resilience have been published to improve reliability, power, and performance. They try to recover design margins by redistributing slack to the critical pipeline stages using combinatorial and sequential optimizations. Combinatorial optimizations like *EVAL* (Environment for Variation-Afflicted Logic) speeds up the timing of critical paths through *Adaptive Body Bias* (*ABB*) as well as *Adaptive Supply Voltage* (*ASV*) [33]. It reshapes signal paths by speeding up slower paths and slowing down faster paths, which saves overall energy consumption but has significant area overheads. Another work called *Blue Shift* uses *On-demand Selective Biasing* (*OSB*) and *Path Constrained Tuning* (*PCT*) to optimize selective critical paths [34]. The tuning knobs select frequently executed critical paths, add forward body bias to some of the gates, and selectively tighten their timing constraints to achieve performance gains but have significant power overheads. Power-aware slack distribution (*SlackOptimizer*) [36] was proposed which use cell sizing to distribute slack evenly in a power and cost-efficient manner, but the benefits are small compared to the optimization effort. Another work in this area is *Selective Endpoint Optimization* (*SEOpt*), *Clock Skew Optimization* (*SkewOpt*) and *Combined Optimization* (*CombOpt*) [37], [38]. *SEOpt* claims to reduce the cost of resilience by replacing error tolerant registers with conventional ones using additional margin insertion. *SkewOpt* migrates available timing slack from non-critical paths to critical paths. *CombOpt* is the combination of *SEOpt* and *SkewOpt* and claims significant power savings. Sequential optimizations like retiming help to minimize clock period, achieve low power and maximize tolerance to variations [39], [40]. However, it restricts the space of possible retiming and may result in unnecessary area

Table 2. Various algorithms comparison.

<i>Algorithm</i>	<i>Method</i>	<i>Area overhead</i>	<i>Power overhead</i>	<i>Performance overhead</i>	<i>Performance improvement</i>	<i>Power savings</i>	<i>Area savings</i>
<i>EVAL</i>	ABB, ASV	10.6%	N.A	N.A	40%	N.A	N.A
<i>Blueshift</i>	OSB, PCT	N.A	23%	N.A	6%	N.A	N.A
<i>Slack Optimizer</i>	Cell sizing	2.7%	N.A	N.A	N.A	23%	N.A
<i>Razor</i>	Razor FF	N.A	3%	3%	N.A	50%	N.A
<i>TIMBER</i>	Time borrow FF	10%	10%	Zero	6% slack improvement	N.A	N.A
<i>CombOpt</i>	Selective endpoint & skew opt.	N.A	N.A	N.A	N.A	16%	14%

overheads. Useful skew and clock skew scheduling [41] combined with gate sizing [42], [43] help to compensate variations and speeds up the processor pipelines. All these techniques perform selective optimizations of the critical paths to reduce the area and power overheads. But they retain meta-stability and hold buffer overheads of the speculative pipelines. Moreover, the existing optimizations based on retiming, skew scheduling and gate sizing does not fully reclaim the available slack to be traded off with power/performance. They also limit the amount of time borrowed due to hold violations.

Table 4 shows a comparison of these algorithms. The circuit, architecture as well as algorithms try to optimize different design variables such as the threshold voltage, device size, supply voltage, design constraints etc. for cost-effective reliability enhancement while keeping power and performance impacts at minimal values. The efforts and benefits vary from case to case. For example, *EVAL* gives a performance improvement of 40% at the cost of 10% area overhead. *Blueshift* show 6% performance improvement with 23% power overhead. *Slack Optimizer* and *Razor* gave power savings at the cost of area, power and performance overheads. All the techniques except the *CombOpt* do not show area savings. So, it really depends on which parameter you want to trade-off. But the traditional design environment is suboptimal and should be replaced by better optimization techniques in view of the multi-dimensional tradeoffs observed in nanometer scale designs.

Figure 16. Slack histogram of processor pipeline (a) at worst-case, (b) with 20mV voltage scaling, and (c) with 40mV voltage scaling.

Figure 17. Cost of resilience and voltage scaling trade-off.

2.4 Summary and Discussion

Conventional variation resilience approaches are based on dynamic voltage scaling and in-situ error correction to trade-off power and reliability. The operating conditions are tuned adaptively till *Point of First Failure (PoFF)*, which makes the near critical paths also timing critical. Figure 26 shows the effect of voltage scaling on the slack histogram of critical and near-critical paths in a processor pipeline. Here the baseline is designed for worst-case with a worst negative slack (*WNS*) of 10ps. It illustrates the slack histogram with the voltage scaled by 20mV from the sign off value. Here 312 paths have negative slack and the *WNS* is -600ps. It also shows the slack histogram with the voltage scaled by 40mV where 985 paths have negative slack and *WNS* is -700ps. It is obvious that the number of paths with negative slack and the *WNS* increases with voltage scaling. The newly created critical paths also need to be error resilient which creates huge overheads in existing voltage scaling designs. A suitable metric for the cost overhead of voltage scaling would be total negative slack (*TNS*). Figure 27 plots the *TNS* of the processor pipeline against the voltage scaling steps. It is evident that voltage scaling till the nominal voltage V_a does not increase the *TNS*. Between V_a and V_b *TNS* increases but can be managed using in-situ error correction. Beyond the critical voltage limit V_b , *TNS* increases rapidly and the rise in critical paths makes the error correction cost enormously high. We need to manage the error from the new critical paths failing which the system reliability will be compromised. In the event of high error rates, the system frequency has to be halved to recover from errors. The processor pipeline face varying workloads which makes it difficult for the error correction architecture to adapt quickly, leading to pipeline stalls and crashing. In this situation, the functionality of a processor pipeline cannot be guaranteed. This explains the purpose of our power optimization technique which utilizes opportunities within the chip at design-time to overcome the drawbacks of adaptive tuning of voltage and/or frequency.

CHAPTER 3. SLACK-AWARE DESIGN METHODOLOGY USING TIME BORROWING FLIP-FLOPS FOR VARIATION TOLERANT PIPELINED DIGITAL CIRCUITS

3.1 Design Motivation

The existing automated design flows are suboptimal which results in pipelines which are not fully balanced. This gives an opportunity for the designers to find underutilized slacks in processor pipeline stages. This underutilized slack can be used to reinforce the critical paths to meet performance, power and reliability targets. The existing slack balancing schemes are depicted in Figure 28. Most of these techniques are used for performance optimizations which result in area and power overheads. Retiming in Figure 28(a) has a restricted space to move and will result in multiple instantiations of registers which leads to area overheads. Soft edge flip-flops in Figure 28(b) have a limited time borrowing window and has meta-stability and hold overheads when we speculate the operating frequency. Latch-based circuits in Figure 28(c) have considerable hold buffer overheads tied to a large transparency window. Moreover, timing analysis for latch based designs is not straightforward. Useful skew in Figure 28(d) helps to tolerate more variations but the design margin improvement is limited. Clock skew scheduling and gate sizing in Figure 28(e) are used to achieve a minimal clock period with least area overhead, but the amount of optimization is bounded. The proposed work in Figure 28(f) analyses the available slack based on static timing analysis (STA) results. We insert intentional clock delays along with an error avoidance flip-flop (EAFF) at places with sufficient slack.

We investigated the slack of an industrial processor with three pipeline stages. It can be seen that a large number of critical paths are in the lower slack region while most of the subsequent

Figure 18. Slack balancing schemes: (a) Retiming (b) Soft edge flip-flop (c) Latch based data path (d) Useful skew (e) Clock skew scheduling and gate sizing and (f) Proposed Error avoidance flip-flop.

paths have sufficient slack. The proposed approach utilizes the consecutive slack of the critical paths to improve the critical operating point behaviour of the pipeline. This helps to attain better control over the error rate compared to the speculative techniques. The slack analysis results in

Figure 29 show that the critical paths/near critical paths have significant consecutive stage slack. Therefore, we can reduce the design margins of the critical paths and still recover dynamic errors by borrowing time from the next pipeline stages. We insert intentional clock delays base on available slacks which makes this approach deterministic and reliable. There is no latency overhead or error detection overhead as in Razor. Timing is closed based on the delayed clock edge which eliminates the need for any error propagation as in TIMBER.

3.2 Module Wise Slack Redistribution

Figure 19. Critical slack and consecutive slack in a processor pipeline.

Our method assigns different time-borrow bins to each critical path flip-flop, based on their maximum slack availability. Table 5 shows the critical paths present in the processor modules at different phases of the design. *Normal* represents the critical paths before any opportunistic

Table 3. Coarse grain TB map shows number of critical paths before and after EAFF insertion.

Processor Modules	<i>Normal</i>	<i>TB1</i> 0.6ns	<i>TB2</i> 1.2ns	<i>TB3</i> 1.8ns	<i>TB4</i> 2.4ns
M1	39	1	13	29	36
M2	11	0	7	9	9
M3	649	229	250	277	282
M4	43	2	3	9	31
M5	2	0	0	1	1
M6	8	0	0	0	4
M7	25	5	23	24	24
M8	81	33	38	76	78
M9	3	2	2	2	2

time borrowing and $TB1 = T_{ck}/8$, $TB2 = 2T_{ck}/8$, $TB3 = 3T_{ck}/8$ and $TB4 = 4T_{ck}/8$ represents the time borrowed by the critical flip-flops based on their maximum slack availability. This maximizes their safety margins which can be traded off with power, area or performance. In this work, the clock delay is limited to $TB1 = T_{ck}/8$, $TB2 = 2T_{ck}/8$, $TB3 = 3T_{ck}/8$ and $TB4 = 4T_{ck}/8$ for simplicity. Module M3 has the maximum number of critical paths and so has the maximum EAFF replacement.

Figure 20. (a) EAFF architecture and (b) timing diagram.

3.3 Principle of Error Avoidance Flip-flop

Figure 21. Simulation results for EAFF.

The error avoidance flip-flop (EAFF) architecture and its timing are depicted in Figure 30 [44] which is derived from TIMBER [31]. EAFF consists of the main latch, a shadow latch, and error generation logic circuits. The data path is comprised of the main latch and the slave latch forming an edge triggered master-slave flip-flop. Any delay in the input DATA signal is detected by the shadow latch. The delayed clock (DCK) is a delayed version of the system clock (CK). The delay amount decides the design margin improvement of the EAFF. The input to the slave latch is switched between the master path and the shadow path using the control signals (P0 and P1) to ensure that the delayed data passes through the slave latch. The main latch samples the input data and transfers it to the output (Q) at the positive clock edge when TG0 is closed and TG1 is open. Similarly, the slave latch samples the data and passes it to the output when TG1 is closed and TG0 is open. The data in the master latch is compared with that in the slave latch by an XOR gate. An error signal (ERROR_XOR) is generated when they are different. ERROR_XOR is sampled by P1 to avoid any glitches and fake errors which is an improvement over TIMBER. The detailed simulation result of the EAFF is depicted in Figure 31. It shows that the error in Q due to a delayed input signal is masked by the EAFF. Note that ERROR_SAMPLED filters unwanted glitches and fake errors from ERROR_XOR. ERROR_SAMPLED is an indication of the variation encountered in the critical endpoint because this signal will be triggered low every time there is a delayed data signal due to static or dynamic variations. Static time borrowing based on available slack eliminates error propagation overheads compared to TIMBER. DCK can be generated internally or externally and the delay amount determines the design margin improvement. In this work, DCK is designed to be proportional to the consecutive slack and is determined by the proposed slack analysis algorithm. The use of the shadow latch for detecting late arriving data prevents any meta-stability issues that may occur otherwise. Hold issues have to be rectified at the design stage by adding buffers. The proposed EAFF relies on the available slack in the consecutive stages. Therefore, we do not require a non-speculative pipeline stage in the design to absorb the borrowed time as in TIMBER.

3.4 Proposed EAFF Based Design Methodology

3.4.1 EAFF Replacement Strategies

Figure 32 shows the EAFF replacement strategies used in the proposed algorithm. Static timing analysis (STA) is executed as a prerequisite for the worst-case design to identify critical paths with large slack in the consecutive stages. From this slack analysis, three time-borrowing strategies are identified.

Figure 32(a) shows the case when negligible slack is available in the consecutive stages. In this case, the proposed algorithm cannot find enough slack in the next consecutive stages. Neither time borrowing nor EAFF replacement will be conducted. Our analysis shows that 85% of the critical/near critical paths in the processor have sufficient slack in the consecutive stages. Therefore, the proposed method can be effective for the design margin improvement of these critical paths.

Figure 32(b) describes another case when significant slack is present in the consecutive stages of the critical paths. Depending on the amount of the available slack, we use either coarse grain or fine grain time borrowing. Once the amount of time borrowing (TB) is fixed for each critical flip-flop, it can dynamically borrow time within that interval. TB is divided into four bands (TB1 to TB4) proportionate to the consecutive stage slack. TB1 and TB2 are used for the coarse grain TB while TB3 and TB4 are employed for the fine grain TB. If the available slack is smaller than TB1, the fine grain TB will be executed. Here, the error checking window will be a smaller fraction of the clock period compared to the coarse grain TB. The clock delay and hold buffers overheads would be lesser in the fine grain TB. Flip-flops in the TB4 cluster have higher slack in the consecutive stage and thus have better variation tolerance. Finally, when little or no slack is available in the consecutive stages, multistage TB is considered. In this condition, flip-flops in the earlier stages are replaced by EAFFs.

Figure 32(c) depicts a case with three stages where the flip-flops in the 1st and the 2nd stages will be replaced by EAFFs.

3.4.2 Design Flow and Algorithm

Figure 22. Time borrowing strategies: (a) without time borrowing, (b) single stage time borrowing, coarse grain or fine grain, and (c) multistage time borrowing.

Figure 33 shows the design flow of the proposed EAFF replacement algorithm [44]. The first step is to filter the critical endpoints in the worst-case design. In this work, we select the most critical 10% of the total critical endpoints for analysis. Their slack is less than 2% of the clock period represented by P . Before inserting EAFFs in the critical paths, we have to find the critical paths with sufficient slack. So, we execute the design flow till placement and optimization and run static timing analysis (STA) to report the worst slack S in the consecutive stages. Depending on the slack in the 2nd stage, we group the critical flip-flops into the coarse grain and fine grain time borrowing slots depending on whether S is greater than or less than the predetermined threshold Th . In our algorithm, Th is set as $T_{ck}/8$. If the slack in the 2nd stage is negligible, the slack in the 3rd stage represented by S_2 is analyzed to enable the multistage TB. If both stages have negligible slacks, no EAFF replacement is done for that critical endpoint. Once the TB mode of each flip-flop is decided, we have to replace the respective flip-flops by EAFFs. We do the flip-flop replacement using custom scripts in the synthesis engine. The design is placed and optimized again in the backend tool followed by clock tree, routing and sign off.

Figure 34 shows the pseudo code of the proposed EAFF replacement scheme. The pseudocode has three main procedures. The first one analyses the critical path slacks to filter

Figure 23. Proposed slack analysis and EAFF replacement flow.

the most critical 10% critical endpoints. The consecutive slacks of these paths are then reported. Based on the slack information we decide the time borrowing mode in the next procedure. Those endpoints with sufficient stage 1 and stage 2 slack are marked for multistage time borrowing. Those paths with sufficient stage 1 slack higher than TB are marked for coarse grain time borrowing and the paths with stage 2 slack less than TB are marked for fine grain time borrowing. Depending on the calculated slacks for all the endpoints, we replace either the stage1 or both stage 1 and stage 2 endpoints with EAFF. Depending on the amount of slack

available, we decide the clock delay.

This clock delay decides the time borrowed by the critical endpoints. The clock delay can be designed with several options. The simpler one provides the same clock delay equal to $T_{ck}/8$

Algorithm 1 Slack aware EAFF replacement

```

1. procedure Slack analysis (Initial Netlist)
2.    $P \leftarrow \text{Find Critical Paths } ()$ 
3.   for all  $p \in P$  do
4.      $S \leftarrow \text{Find Consecutive Slack } (p)$ 
5.   end for
6.   for all  $p \in P$  do
7.     if  $S > 0$  then
8.       if  $S > Th$  then
9.          $\text{CoarseGrainTB}(\text{netlist}, P)$ 
10.      else
11.         $\text{FineGrainTB}(\text{netlist}, P)$ 
12.      end if
13.    else
14.       $S2 \leftarrow \text{Find Next Consecutive Slack } (p)$ 
15.      if  $S2 > 0$  then
16.         $\text{MultistageTB}(\text{netlist}, P)$ 
17.      end if
18.    end if
19.  end for
20. # EAFF replacement
21. procedure  $TB(\text{netlist}, P)$ 
22.   for all  $TB = TB1, TB < TB4, TB = TB + TB1$  do
23.     for all  $p \in P$  do
24.       if  $s \geq TB$  then
25.          $\text{Replace Sequential } (p)$ 
26.       end if
27.     end for
28.   end for
29. return  $(\text{netlist})$ 

```

Figure 24. EAFF replacement algorithm.

for all the critical path flip-flops as shown in Figure 35(a). We choose $T_{ck}/8$ because the majority of critical paths can borrow a minimum slack equal to $T_{ck}/8$. The second option is to use higher clock delays for those critical clusters whose consecutive stage has bigger slacks as depicted in Figure 35(b). The third option is to use multiple enable signals for the different time borrowing clusters (Figure 35(c)). In this case, we can selectively activate time borrowing only when necessary based on the ERROR_SAMPLED signal.

Figure 36 explains the design environment of the proposed error resilience technique. The shaded blocks are added to the typical design environment. We develop a standard cell library with 20 different flavours of the error avoidance flip-flops to replace the normal flip-flops as and when necessary. We have the scan/non-scan versions of flip-flops with the set, reset and enable functionality. Liberate library characterization tool is used to generate the timing model

for EAFFs. The synthesis engine is amended with custom scripts to enable EAFF replacement.

The maximum delay constraints of error-tolerant designs are relaxed by the error detection

Figure 35. Clocking strategies for EAFF: (a) same clock delay used for all EAFFs, (b) different clock delays for different EAFF clusters, and (c) different clock delays with individual enable.

Figure 36. Design environment for EAFF based pipeline.

Figure 37. Timing speculation window: (a) for Razor flip-flops, max delay is relaxed and min delay is deteriorated by $T_{ck}/2$, (b) for TIMBER, the max delay improvement is less than $T_{ck}/2$, the min delay is deteriorated by $T_{ck}/2$, and (c) for the proposed EAFF, max delay and min delay constraints in each stage is proportional to the consecutive stage slack.

window while the minimum delay constraints are tightened by the same amount. For Razor [18] based error detection and masking circuits, the error checking window (T_w) is a half clock cycle wide or a fraction of the clock pulse. Razor flip-flops can relax maximum delay by $T_{ck}/2$ and constrain minimum delay by the same amount as described in Figure 37(a). TIMBER divides the available time borrow window into multiple stages and reclaims a fixed max delay equal to one-third of the clock phase in each pipeline stage (Figure 37 (b)). TIMBER also needs to insert buffers in the short paths to deal with the minimum delay constraints. Post-silicon frequency tuning changes the clock pulse width, which demands duty cycle tuning to deal with the minimum delay violations. So, the savings from a relaxed max delay is less compared to the overheads in terms of error handling and min delay constraints. In the proposed EAFF, the checking window depends on the consecutive slack and each pipeline stage is designed with the right amount of max delay speculation and min delay overhead (Figure 37 (c)). The flip-flops dynamically borrow time within the respective time borrowing window. The critical path with more slack in the consecutive stage has proportional design margin improvement at the cost of hold buffer overhead. The proposed EAFF based methodology sets the maximum and minimum delay constraints proportionate to the amount of slack available in the consecutive stage which leads to an optimum number of hold buffers in each pipeline stage. For each critical flip-flop, design margin is improved based on deterministic slack evaluation during the design stage. The design margin improvement is more reliable and predictable, which results in less overhead compared to real-time design margin speculation and subsequent error handling. A good performance metric for the error tolerant flip-flops in each pipeline stage would be the ratio of max delay savings to the min delay increase under minimal error handling overheads as shown in Equation 3.1 below.

$$performance\ metric = \frac{\Delta T_{maxdelay}}{\Delta T_{mindelay}} \quad (3.1)$$

This metric should be equal to one so that we will benefit from the error resilience methodology. For Razor, the max delay can be speculated up till $T_{ck}/2$ and the min delay for each Razor stage should be constrained to $T_{ck}/2$ which makes the performance metric less than or equal to one. Note that here the performance metric can be equal to one only if the voltage or frequency scaling allows the max delay of each critical endpoint to be speculated till $T_{ck}/2$. But it has huge error correction overheads due to the critical wall of slack behaviour of the pipeline stages. A higher error rate would dismiss all the benefits attained from the maximum delay saved because of the cost of error handling. For TIMBER, the maximum delay speculation is less than $T_{ck}/2$ whereas the minimum delay holds buffers overhead during design-time still follow the constraint $T_{ck}/2$. This makes the performance metric less than one which is not desirable even though the error handling overhead is less than Razor. For the proposed EAFF, the max delay gain for each critical endpoint is proportional to the available slack in the next stage. The hold buffer overhead at each critical flip-flop is also proportional to the slack available. There is no error correction or error propagation overhead for the

maximum delay improvement. This gives a performance metric equal to one for the proposed methodology with minimum overheads.

Figure 25. Slack improvement with EAFF for processor modules, M1, M2, M3, M4, M7 and M8.

3.5 Results and Analysis

Error resilience experiments were conducted on a 40-nm embedded processor pipeline. The worst-case corner design was used as the baseline to compare the power, area, and performance. The slack analysis was done for the worst-case design and proper EAFFs were inserted into the selected critical endpoints. Power and area after EAFF insertion were compared against the baseline with no EAFFs.

3.5.1 Slack Improvement

All the processor modules show significant slack improvement as shown in Figure 38. M5 and M6 are excluded because it has a negligible number of critical paths. The slack improvement after EAFF insertion is 10× the worst-case slack. The slack histogram after EAFF insertion is tilted to the right, which shows that EAFF helps to smooth out the critical wall of slack behaviour of the design. Any dynamic variation induced error can be easily masked using the available time borrowing of up to $T_{ck}/2$. Thus the design margins inserted by the worst-case corner is fully reclaimed. Additional savings can be obtained if we tune the operating point beyond Point of First Failure (PoFF) using the slack margin speculatively or during design. The critical wall of slack behaviour is improved compared to other error tolerant techniques. Module M3 in the execute (EXE) pipeline stage has the most number of critical paths.

Figure 26. Percentage of paths improved using the proposed time borrowing strategies.

Figure 39 shows the percentage of critical/near critical paths in each processor module that gets timing margin improvement using the proposed strategies. Results show that 7 out of 9 modules get timing margin improvement on all critical paths. The three different strategies (coarse grain, fine grain, and multistage borrowing schemes) help to recover worst-case design margins from 85% of the critical endpoints. Modules M3 and M8 with the most number of critical paths obtain design margin improvement on 85% and 74% of the paths, respectively.

Figure 27. (a) Chip power overhead for EAFF design and (b) Chip area overhead for EAFF design.

3.5.2 Power and Area overheads

EAFFs have power and area overhead compared to master-slave flip-flops. However, the proposed method has minimal architectural overheads caused by error correction and propagation. Figure 40(a) shows the chip wise power overhead plotted against the time borrow capability of the critical endpoints. We use $T_{ck}/8$ as the minimum time which can be borrowed by a critical path and the subsequent time-borrow band is incremented by $T_{ck}/8$. A maximum number of critical endpoints will be optimized for the time-borrow value $T_{ck}/8$ which reduces

with increase in time borrowed. The peak chip power overhead is 5% when the critical paths borrow time up to $T_{ck}/8$ from the consecutive stages. This is the point where maximum EAFF replacements happen. Figure 40(b) shows the chip wise area overhead plotted against the time borrow capability of the critical endpoints. For example, TB1 represents the design with normal flip-flops being replaced by EAFF for all critical endpoints that have a slack equal to $T_{ck}/8$ (TB1) in the consecutive stage. The maximum area overhead after EAFF replacement is limited to a mere 0.2% at TB1. Even though some of the paths have a time borrowing capability above $T_{ck}/2$, we limit the maximum amount of time that can be borrowed by an EAFF as $T_{ck}/2$ (TB4).

Figure 41(a) shows the percentage of SBFFs being inserted for the different slack relaxation margins under consideration. 60% of critical flip-flops are replaced by EAFFs when we relax the paths with a slack margin greater than or equal to $TB1 = T_{ck}/8$. 52% of flip-flops are replaced by EAFFs when we set the consecutive slack window equal to $TB4 = T_{ck}/2$.

Figure 41(b) shows the area overhead plotted against the relaxation window. It is evident that we get the maximum area overhead of $\sim 0.2\%$ for the lower relaxation windows where more EAFFs are getting replaced compared to higher relaxation windows where fewer EAFFs are getting replaced.

Figure 41(c) shows the power overhead of EAFF which shows a maximum power overhead of 5.4% for the slack relaxation window of $T_{ck}/8$. The power and area overhead for any time borrowing window depends on the number of EAFFs beings replaced as well as the flipflop flavour which is being replaced. Some of the EAFF flavours have notably higher power consumption compared to the other flavours which make the power overhead non-linear.

Figure 28. (a) Percentage EAFF replacement (b) Chip area overhead and (c) Chip power overhead for EAFF design.

Figure 29. Slack distribution of processor using BTWC voltage corner for: (c) Vdd=0.97 V (d) Vdd=0.98 V.

3.5.3 Better than Worst-case Design Opportunity

- 3.6 Once the pipeline is reinforced with EAFF, there is an opportunity for BTWC design. Figure 42, Figure 43, and Figure 44 and Figure 45 show the simulated voltage scaling effects on the slack distribution of the processor design. The design is usually signed off for worst-case voltage corner which is Vdd=0.99V. In this case, we interpolate the logic timing for different BTWC voltage corners such as Vdd=0.98V, 0.97V, 0.96V and 0.95V. Voltage scaling shifts the slack distribution curve left towards negative slack. The number of paths with negative slack also increases as we employ more and more voltage scaling. Figure 44 shows the slack distribution of consecutive pipeline stage at Vdd=0.98 V. There is still a huge amount of slack in the consecutive stage for most of the paths, which show that with EAFF's, the paths with additional slack are protected from the voltage scaling related timing errors. This proves that time borrowing is effective to deal with voltage variations as well as to design a pipeline using a BTWC voltage corner. This gives an opportunity for power saving for low power designs.

Figure 30. Slack distribution of processor after using BTWC voltage corner for (a) Vdd=0.95 V (b) Vdd=0.96 V.

Figure 31. (a) critical stage and (b) consecutive stage slack distribution of processor for BTWC voltage corner for $V_{dd}=0.98V$.

3.7 Conclusions

In this chapter, we proposed an efficient error avoidance methodology that can improve the reliability and at the same time provide flexibility for power and area tuning. Compared to previous approaches, the design margin enhancement made at each pipeline stage is proportionate to the slack available in the consecutive stage. This results in a hold buffer overhead proportional to the available slack rather than inserting hold buffers for the entire speculation time period. This deterministic approach removes error correction and error propagation overheads compared to the other error resilience techniques. Experimental results show that in spite of a chip wise power overhead of 5% and area overhead of 0.2%, the proposed method is able to employ BTWC voltage corners which result in enormous power savings. Moreover, we get maximum design margin improvement compared to the existing slack balancing techniques.

Figure 32. Slack distribution of processor for different voltage corners.

CHAPTER 4. A POWER EFFICIENT VARIATION TOLERANT PIPELINE DESIGN METHODOLOGY USING SLACK BALANCING FLIP-FLOPS AND LOGIC RESHAPING

Speculative voltage scaling and error correction for power-aware variation tolerance incur huge cost overheads for error recovery. Moreover, such techniques are prone to meta-stability issues due to runtime design margining. On the other hand, performance-aware variation tolerance using gate sizing and clock skew scheduling result in an increase in combinational paths with critical delays. In this paper, we propose a design-time technique to relax the critical timing margins of a processor pipeline while still ensuring correct functionality at worst operating conditions. The proposed method utilizes positive slack available in the pipeline stages and re-distributes it to the preceding critical logic stage using Slack Balancing Flip-flops (SBFFs). We also propose a critical logic power minimization method which relaxes the slack margins of all the timing paths including the short paths converging into the SBFF and resynthesizes the pipeline for power and area savings. Our experiments show that the proposed method could relax the slack margins of the critical logic stage in the range of 11 percent to 44 percent of the clock period. The proposed logic reshaping results in 12 percent and 8 percent power and area savings respectively compared to the worst-case design approach. In addition, the timing budgeting and timing correction using opportunistic slack eliminates critical operating point behaviour, meta-stability issues and hold buffer overheads encountered in existing resilience techniques. In traditional processors, pipelines are not evenly distributed due to sub-optimal balancing techniques. The design margin improvements from the existing slack balancing techniques are always bounded due to the automation tool limitations. The proposed custom approach leverages the inherent slack imbalance present in processor pipelines. Our design methodology has two phases. In the first phase, critical endpoints with sufficient consecutive slack are replaced by the proposed slack balancing flip-flops (SBFF) for time borrowing. In the second phase, critical paths with enough slack margin are downsized for power and area reduction.

Figure 33. Design margin Vs error rate for different error resilience techniques.

4.1 Motivation

Conventional design margining approaches [18] – [21] are based on dynamic operating point tuning and in-situ error correction to trade-off power and reliability. The operating conditions are tuned adaptively till Point of First Failure (PoFF), which makes the near critical paths also timing critical. In the event of high error rates, the system frequency has to be halved to recover from errors. The proposed approach which is an extension of our previous work [44] leverages the underutilized slack present in processor pipelines after the tool based optimizations. We use static timing analysis (STA) to look for near critical endpoints with sufficient consecutive slack after placement and logic optimizations. The critical endpoints are then back annotated to the synthesis tool and we relax the timing margins of all the paths in critical stage proportionate to the available slack. After relaxing the path slacks, we resynthesize the critical fan-in logic and replace the endpoints with slack balancing flip-flops (SBFF). SBFFs are designed to sample data at a delayed clock edge. Re-synthesis with relaxed timing margins delays the long paths and pushes them towards the SBFF delayed clock edge. The delayed data still maintain safe design margins with respect to the delayed clock edge of the SBFF. This removes meta-stability issues prevalent in the speculative flip-flop pipelines. The methodology removes the critical wall of slack issues from the data path. Unlike the in-situ techniques, the short paths converging to the SBFF also get delayed which removes the hold buffer overheads resulting from delayed data sampling. Re-synthesis downsizes and restructures the logic gates in the critical fan-in stage which gives significant power and area savings. This results in the optimal power-aware design of the processor pipeline from ground up without the failure risks of runtime speculative operating point tuning. They are not limited by the gate sizing and skew optimization constraints in typical combinatorial optimizations which results in the maximum slack power trade-off along the processor pipeline. Figure 46 illustrates how the proposed method is superior over Razor which has critical operating point behaviour and TIMBER which has limited design margin improvement.

Figure 34. Critical path slacks (pipe_1) and consecutive stage slacks (pipe_2) of the processor.

Typical processor pipelines are highly unbalanced with respect to the slack present in various combinational logic stages. Traditional logic synthesis tools focus on optimizing the combinational logic and do not support any trade-offs between the combinational paths separated by sequential elements. Sequential optimizations like retiming, which is used for slack balancing across registers, introduce additional power and area overheads. To address the above issues, we investigated a processor with three pipeline stages and found that majority of paths in the near critical slack region (pipe1_slack) have sufficient slack in the consecutive stage (pipe_2 slack) (Figure 47) after physical design. We leverage this slack to relax and resynthesize the critical fan-in logic and sample the resulting delayed inputs using a slack balancing flip-flop. Instead of reducing design margins at runtime using adaptive techniques, the proposed approach downsizes the critical fan-in logic of the pipeline for power savings. Worst-case design margins are still met with respect to the delayed clock edge of the slack balancing flip-flop which helps to eliminate meta-stability issue faced by adaptive in-situ error correction schemes. Moreover, we also relax the short paths by the same design margin which eliminates the need for SBFF induced short path buffer insertions later in the design flow. In contrast to the existing slack redistribution techniques, we insert intentional clock delays to fully balance the available slack at the critical endpoints and redesign the whole critical logic stage with the relaxed time constraints. This results in a pipeline optimized for low power without the need for runtime voltage speculation which is vulnerable to critical operating point behaviour.

4.2 Proposed Variation Tolerant Pipeline Design

4.2.1 Slack Balancing Principle

To explain slack balancing, we consider a small circuit as shown in Figure 48 with four registers. The timing graph shows nodes corresponding to the registers and edges corresponding to the combinational paths. The maximum delay for each combinational path is shown beside the edges. This circuit is optimized for a minimum feasible clock period $T=11$. The combinational path (d, b) has the maximum delay of 11. In this critical path, time can be borrowed from the consecutive stage (b, d) whose delay is 9. We replace the flip-flop b by the slack balancing flip-flop (SBFF) with a clock delay equal to the slack present in the consecutive stage (b, d) , which is 2. Now for the same clock period $T=11$, we have an extra margin of 2 in the fan-in paths (a, b) and (d, b) . The extra margin is used to relax and downsize the logic in the fan-in paths making sure it does not create other timing violations. This procedure also delays the AND gate in the critical fan-in which is assumed to be a short path with much less delay. Because the short path also gets delayed by the gate downsizing, we can eliminate the need for additional hold buffers.

We compare the proposed technique with existing speculative error correction schemes such as Razor[18] and TIMBER[31]. Figure 49(a) shows conceptually how the slack margins are relaxed in a typical voltage scaling based resilience scheme. We consider four timing paths $P1$, $P2$, $P3$, and $P4$ with different slack margins. Here, we assume $P3$ a part of a non-critical fan-

in logic and P1, P2 and P4 in the critical fan-in logic with P1 being a critical short path. Energy can be reduced by scaling the supply voltage, which increases the delay along these paths. The

Figure 35. A circuit example and timing graph to explain slack balancing.

Figure 36. Speculative power-aware flip-flop: (a) slack margin at sign off, (b) slack margin relaxed with timing speculation flip-flop, and (c) slack margin traded off with power.

Figure 37. Performance-aware optimizations: (a) slack margin at sign off, (b) slack margin relaxed with clock skew scheduling and gate sizing (c) slack margin traded off with frequency.

Figure 38. Proposed SBFF: (a) slack margin at sign off, (b) slack margin relaxed with SBFF, and (c) slack margin used to downsize logic.

timing speculation window for a typical speculative error resilience scheme is $T_{ck}/2$. So, the paths are allowed to relax the delay margins by $T_{ck}/2$ as shown in Figure 49(b). Voltage scaling is done beyond PoFF until the critical operating voltage the near critical path P3 also becomes timing critical as depicted in Figure 49(c). Moreover, runtime voltage scaling causes the inputs to change close to the clock edges creating meta-stability issues. They also require additional buffers along P1 to ensure that it will not corrupt the delayed data and hence the error generated. Even if timing speculation is not done till $T_{ck}/2$, we still need to insert buffers for an equivalent delay of $T_{ck}/2$ to ensure that the error signal is not corrupted. Unless we use a fine grain voltage scaling, even non-critical paths like P3 will become critical. Also for performance-driven design margin relaxation techniques, the slack relaxations are suboptimal as shown in Figure 50(b). Frequency scaling results in new critical path formations in P3 as shown in Figure 50(c) which makes the scaling bounded.

In our proposed technique, we use static time borrowing to trade-off slack margin for power and area gains. Figure 51(a) shows the slack margins of the baseline worst-case design. We look for positive slack in the pipeline stages and replace the critical endpoints with SBFFs which borrows a time TB proportional to the available slacks as depicted in Figure 51(b). As a result, the critical fan-in paths P1, P2, and P4 are relaxed by TB. We re-synthesize the pipeline to downsize the fan-in logic of SBFF for power and area reductions (Figure 51(c)). After resizing the logic, timing closure is done on the data paths with respect to the delayed clock edges of SBFF. Our approach is deterministic and non-speculative, unlike the usual dynamic operating point scaling techniques. Safe design margins are kept with respect to the delayed clock edge of the SBFF which prevents meta-stability issues in the pipeline. Unlike conventional resilience schemes, the short path P1 is also delayed by the corresponding TB window TB1. This eliminates the need for additional short path buffers along P1 to prevent it from corrupting the delayed data. Note that the time borrow window TB is fully utilized for power and area reduction compared to other optimization techniques. Unlike clock skew scheduling which is mostly used for performance enhancements, the proposed method targets critical logic power minimization by utilizing maximum available slack to downsize all logic paths in the critical stage.

4.2.2 Slack Balancing Flip-flop (SBFF)

The slack balancing flip-flop (SBFF) and the timing diagram are illustrated in Figure 52. The architecture is very similar to EAFF [44], but we use it to capture the delayed data coming from the under-designed fan-in logic. SBFF is a simplified version of *TIMBER* [31] without any error propagation logic. SBFF consists of a main latch, a shadow latch, and a clock control circuit. The data path has a main latch and a slave latch forming an edge triggered master slave flip-flop. The proposed optimization technique downsizes the SBFF input logic for power reduction. This delays the *DATA* signal, which is detected by the shadow latch using a delayed clock (*DCK*). The delay amount decides the design margin improvement. The input to the slave latch is switched between the master path and the shadow path using the control signals (*P0* and *P1*) to ensure that the delayed data passes through the slave latch. As shown in Figure 52, the main latch samples the input data and transfers it to the output (*Q*) at the positive clock (*CK*) edge when *TG0* is closed and *TG1* is open. Similarly, the slave latch samples the data and passes it to the output at the delayed clock edge (*DCK*) when *TG1* is closed and *TG0* is open.

The data in the master latch is compared with that in the slave latch by an *XOR* gate. *XOR_OUT* becomes ‘1’ when they are different. The *XOR* output is sampled by *P1* to avoid sampling any glitches and to generate *ACTIVITY MONITOR*. Depending on the real-time variation in *DATA* and the logic delay, the master latch and shadow latch will sample with the same value or a different value provided a data transition happens in the clock cycle. Thus, *ACTIVITY MONITOR* signal is a measure of the real-time data rate. In Figure 53(a), the monitor signal is triggered only once when the data rate is low. The monitor signal is triggered continuously in all the clock cycles as shown in Figure 53(b) when the data rate is high.

4.2.3 Pipeline Design Flow using SBFF

We use standard CAD tools and custom add-on scripts for SBFF insertion and logic optimization for the proposed pipeline design. A standard cell library with 20 different flavours of SBFFs was developed using library characterization tools to replace the normal flip-flops. The library has four sets of non-scan flip-flops with two different drive strengths and four sets

Figure 52. (a) Proposed SBFF and (b) timing diagram.

Figure 53. Data activity monitoring in SBFF: (a) Intermittent triggering of activity monitor with low data activity, (b) continuous triggering of activity monitor with high data activity.

of scan flip-flops with three different drive strengths. Custom scripts are used to enable slack analysis, time budgeting and SBFF replacement. Figure 54 shows the proposed design flow using SBFF. It starts with filtering the critical paths based on post placement and optimization STA results. In this work, we consider ~10% of the total flip-flops which are most critical

whose slack is less than 2% of the clock period. In the next step, we search for the critical endpoints with sufficient consecutive slacks. These endpoints are back annotated to the synthesis engine. The timing for all the critical fan-in paths is relaxed proportional to the available slack. The pipeline is then resynthesized with the new timing constraints. We then replace the endpoints by SBFFs and run the placement and clock tree. The clock delays for the SBFFs are inserted after the clock tree is built. This helps to preserve the traditional tool based optimizations. We limit the inserted clock delay to $TB1 = T_{ck}/8$, $TB2 = 2T_{ck}/8$, $TB3 = 3T_{ck}/8$ and $TB4 = 4T_{ck}/8$ for simplicity. The pipeline is then signed off keeping worst-case design margins.

4.2.4 Meta-stability and Hold Issues

This section describes the mathematical formulation for max delay and min delay of an SBFF and compare it with MSFF. It shows how the max delay is improved or deteriorated compared

Figure 39. Design flow using SBFF.

to the existing techniques. We also describe Mean Time Between Failure for the speculative techniques due to the meta-stability issue along the data path and the error path. The different parameters involved show the dependency on error signal timing in addition to the data path timing. Speculative techniques relax max delay and constraint min delay by $T_{ck}/2$. The max delay savings is minimal if we consider error handling and min delay overheads. For a typical Razor pipeline shown in Figure 55, the max delay, T_{max} can be speculated up to $T_{ck}/2$ till the falling edge of the clock and is calculated as

$$T_{max} = T_{ck} + T_{ck}/2 - T_{su,ck-f} \quad (4.1)$$

$T_{su,ck-f}$ represents the setup timing of the Razor flip-flop clock falling edge. The effective max delay improvement $T_{max,eff}$ is limited by the critical operating point and is calculated as

$$T_{max,eff} = T_{ck} + T_{cop} - T_{su,ck-f} \quad (4.2)$$

T_{cop} represents the effective amount by which max delay is relaxed till the critical operating point. Min delay requirement, T_{min} for Razor is tightened by $T_{ck}/2$ and is calculated as

$$T_{min} = T_{ck}/2 + T_{h,ck-f} \quad (4.3)$$

$T_{h,ck-f}$ represent the hold timing requirement of the Razor clock falling edge. Thus, for a max

Figure 40. (a) Speculative flip-flop pipeline, (b) Data path meta-stability, and (c) Error path meta-stability.

Figure 41. (a) SBFF pipeline, (b) Data path timing.

delay improvement of T_{cop} , there is a min delay buffer overhead of $T_{ck}/2$. Additional buffer overheads on near critical endpoints dismiss the benefits of Razor technique for high error rates.

SBFF relaxes the max delay and tightens the min delay of the critical fan-in logic by an amount T_B proportional to the consecutive stage slack. The SBFF max delay, $T_{max,SBFF}$ is calculated as

$$T_{max,SBFF} = T_{ck} + T_B - T_{su,ck_del-r} \quad (4.4)$$

T_{su,ck_del-r} represents the setup time constraint with respect to SBFF delayed clock rising edge. The logic downsizing makes sure that the short path delay also increases by T_B thus eliminating the need for additional hold buffers. The SBFF min delay, $T_{min,SBFF}$ is calculated as

$$T_{min,SBFF} = T_B + T_{h,ck_del-r} \quad (4.5)$$

T_{h,ck_del-r} represents the hold timing requirement with respect to the SBFF delayed clock edge. As long as the T_B constraint is satisfied, the D input meets the worst-case setup timing of the delayed clock. This eliminates the need for error correction and the error path is used to monitor the frequency of variations. We can represent the maximum delay constraint on T_B as

$$T_{Bmax,SBFF} \leq T_{ck/2} - T_{su,ck_del-r} \quad (4.6)$$

Runtime dynamic timing speculation techniques have meta-stability risks at the clock edges along data path and error path. Data path meta-stability will arise if D input changes too close to the positive clock edge due to voltage scaling or PVT variations as depicted in Figure 55(b). As a statistical measure of the reliability from metastable errors, we express the *Mean Time between Failures*, $MTBF_{data-path}$ as

$$MTBF_{data-path} = \frac{\exp(\frac{t_{res}}{\tau_c})}{f_{data} \cdot T_0 \cdot f_{clk}} \quad (4.7)$$

Here t_{res} represents the resolution time to exit the metastable state, T_0 is the meta-stability window, τ_c is the meta-stability resolution time constant, f_{dat} is the frequency of data transition and f_{clk} is the clock frequency. The meta-stability resolution time t_{res} can be calculated as

$$t_{res} = T_{clk} - t_{ck2out} - t_{comb} - t_{su} \quad (4.8)$$

Here t_{ck2out} is the clock to output delay of the Razor flip-flop, t_{comb} is the combinational logic delay succeeding the Razor stage and t_{su} is the setup time constraint of the subsequent flip-flop as shown in Figure 55(b). A higher t_{res} amounts to better protection from metastable errors. For the Razor design, t_{ck2out} is worse than the conventional flip-flops due to the redundant latch which makes the t_{res} smaller. This introduces additional constraints on the combinational delay of the next pipeline stage. The error path is also prone to meta-stability because the dynamic voltage scaling pushes the data too close to the negative clock edge as shown in Figure 55(c). The corresponding *Mean Time between Failures*, $MTBF_{err-path}$ is represented as

$$MTBF_{err-path} = \frac{\exp(\frac{t_{res}}{\tau_c})}{f_{data} \cdot T_0 \cdot f_{clk}} \quad (4.9)$$

Here the meta-stability resolution time t_{res} can be expressed as

$$t_{res} = T_{ck} - T_B - t_{ck2err} - t_{comb} - t_{su} \quad (4.10)$$

Here t_{ck2err} is the clock to output delay for the error output signal with respect to the negative clock edge. As shown in Figure 55(c), the t_{res} window for error path is too small and this puts additional constraints on the error path delay $t_{err-path}$. Resolution time can extend up to 3 clock cycles which means the error signal will take three clock cycles to resolve fully in the event of meta-stability. Any timing issues along the error path will affect the error recovery process and the system may not be able to get back to the normal error-free state.

Mean Time between Failures for the SBFF data path can be represented as

$$MTBF_{data-path,SBFF} = \frac{\exp(\frac{t_{res}}{\tau_c})}{f_{data} \cdot T_0 \cdot f_{clk}} \quad (4.11)$$

Here t_{res} represents the resolution time to come out of the metastable state, T_0 is the meta-stability window of the delayed clock edge, τ_c is the meta-stability resolution time constant, f_{data} is the frequency of data transition and f_{clk} is the clock frequency. The meta-stability resolution time t_{res} can be calculated as

$$t_{res,SBFF} = T_{ck} - t_{ck_del2out} - t_{comb} - t_{su} \quad (4.12)$$

Here $t_{ck_del2out}$ is the clock to output delay of the SBFF, t_{comb} is the combinational logic delay

Table 4. Summary of error resilience techniques.

<i>Resilience technique</i>	<i>Data path</i>	<i>Operating point</i>	<i>Meta-stability</i>	<i>Buffer overhead</i>	<i>IPC penalty</i>	<i>Probability of high error rate</i>	<i>Speculation window, Tw</i>
Razor I [18]	Flip-flop	Voltage scaled beyond PoFF	Data path & error path	High	Yes	Yes	$T_{ck}/2$
Razor II [19]	Latch	Voltage scaled beyond PoFF	Error path	High	Yes	Yes	$T_{ck}/2$
DSTB [20] TDTB	Latch	Voltage scaled beyond PoFF	Error path	High	Yes	Yes	$T_{ck}/2$
Bubble Razor [21]	Latch	Voltage scaled beyond PoFF	Error path	Low	Yes	Yes	$T_{ck}/2$
TIMBER [31]	Flip-flop	Frequency scaled beyond PoFF	Data path & error path	High	No	Yes	$T_{ck}/6$
Proposed SBFF	Flip-flop	Worst-case operating corner	No	Low	No	No, design time tuning	TB, relaxation window

MS = Meta-stability, Tw = error detection window, T_{ck} = clock period, DVFS = Dynamic Voltage Frequency Scaling, IPC = Instructions per Cycle, PoFF = Point of First Failure.

succeeding the SBFF and t_{su} is the setup time constraint of the subsequent flip-flop as shown in Figure 56(b). Timing constraints in (4.6) make sure that the set-up constraints are met with the delayed clock edge and so the probability of metastable errors is very low. The absence of meta-stability makes $t_{ck_del2out}$ smaller than t_{ck2out} of Razor FF which results in more resolution time for SBFF in the event of any timing upset. The design margin improvement using the proposed non-speculative approach is reliable and predictable, which result in less overhead

Figure 42. Cell Downsizing (a) Area Vs Downsizing, (b) Delay Vs Downsizing.

Algorithm Slack analysis & Downsizing (SizeOpt)

1. **Procedure** *SizeOpt* (*Initial Netlist*)
 2. Run *STA* to find critical paths P and consecutive slacks
 3. $P \leftarrow \Phi$
 4. **for all** timing endpoints p in the *netlist* **do**
 5. **if** $slack(p) < Th$ **then**
 6. $P \leftarrow P \cup \{p\}$
 7. **end if**
 8. **end for**
 9. **for all** $TB = TB1, TB \leq TB4, TB = TB + TB1$ **do**
 10. **for all** $p \in P$ **do**
 11. **if** $consecutive\ slack(p) \geq TB$ **and** $< 2TB$ **then**
 12. Replace endpoint register by *SBFF* with $DCK = TB$
 13. Adjust timing of fan-in logic by TB
 14. Downsize fan-in logic and close timing w.r.t DCK
 15. **end if**
 16. Calculate Power and Area savings
 17. **return** ($netlist_{TB}$)
 18. **end for**
 19. **end for**
-

Figure 43. SBFF replacement and size based optimization algorithm.

compared to real-time design margin speculation. Table 6 summarizes different speculative error resilient schemes and the proposed non- speculative SBFF scheme.

4.3 Power Optimization Algorithm

The proposed power optimization approach uses the additional slack margin from SBFF to downsize the critical fan-in logic. Figure 57 shows the area reduction and the delay increase when we downsize a typical inverter cell in the standard library. However, the SBFFs in the critical endpoints absorbs this delay increase because the delayed data is safely captured by the delayed clock of the SBFF. This results in fewer resources along the critical data paths, leading

to power and area savings with respect to the baseline design. To realize this, we propose a *Slack Analysis & Downsizing (SizeOpt)* algorithm as described in Figure 58. The procedure takes an *Initial Netlist* which is designed based on worst-case corners. Utilizing the *STA* results of this worst-case design, we compute the slack available in the pipelines. We filter the critical endpoints whose slack margin is less than the predefined threshold (Th), which is set as 2 percent of the clock period in this work. Depending on whether the *consecutive slack* in the 2nd stage falls in $TB1$, $TB2$, $TB3$ or $TB4$, we replace the endpoint by an *SBFF* with a clock delayed by the corresponding TB . If the slack in the 2nd stage is less than $TB1$, no *SBFF* replacement is executed.

Once we have *SBFF* in the selected endpoints, we can relax the timing margins of those endpoints by updating the design constraints. This is done by using specific synthesis commands for the path delay adjustments. Once the path delay is adjusted, the critical fan-in logic is downsized. For the best overall power/area savings, we start from the paths with the highest slack in the range of $TB4$. We replace those paths with *SBFF*s and relax their timing with path adjust commands in the synthesis stage. This process is continued until there is no more critical path which can be relaxed by at least $TB1$. This makes sure that the design undergoes a path relaxation proportional to the available slack.

Figure 44. Slack improvement in different modules.

4.4 Results and Analysis

Error resilience experiments were conducted on a 40nm embedded processor with a three-stage pipeline. The conventional worst-case corner design was used as the baseline to compare the power and area. Based on the slack analysis, we found an opportunity to relax the slack margin in the data path modules shown in Figure 59. The design has ~20K logic cells with 720 critical paths. 60% of the critical endpoints have sufficient slack and can be replaced by SBFFs. Execute stage has the maximum number of critical endpoints and ~326 paths in the Execute stage are relaxed by a maximum slack margin of TB4.

The proposed power optimization approach downsizes the critical data paths that have SBFFs by a predefined set of relaxation margins. Here the actual power and area savings depend on how many endpoints are relaxed and for which particular relaxation margin. Table 7 and Figure 60 shows the power/area savings of the proposed design methodology. Using our algorithm, we get a maximum power saving of ~12% as shown in Figure 60(a). We get the maximum power savings in the Execute stage which has the maximum number of paths beings relaxed by TB4. On the contrary, a typical speculative error resilience approach like TIMBER [31] suffer from a power overhead of 39%. We get an area savings of ~8% with the proposed approach as shown in Figure 60(b). TIMBER has an area overhead of ~1.7%. Again, the area savings predominantly comes from the Execute stage path relaxations. It is evident that the

Figure 45. SizeOpt power optimization technique (a) Power savings, (b) Area savings with respect to worst-case design.

higher relaxation margins of TB4 gives more optimization opportunity and reduces the overheads which result in power and area savings.

Table 5. Power and Area savings of the proposed design methodology.

<i>Power (nW)</i>	<i>Baseline</i>				<i>TIMBER [31]</i>				<i>Proposed</i>			
	<i>#cells</i>	<i>leakage</i>	<i>dynamic</i>	<i>total</i>	<i>#cells</i>	<i>leakage</i>	<i>dynamic</i>	<i>total</i>	<i>#cells</i>	<i>leakage</i>	<i>dynamic</i>	<i>total</i>
<i>Fetch</i>	1976	1463	265127	266590	2014	1481	482389	483870	1751	1103	401268	402372
<i>Decode</i>	1874	920	149114	150034	1900	805	154762	155567	1796	573	162912	163485
<i>Execute</i>	16414	14825	1307869	1322695	16356	14714	1777447	1792162	15053	11039	941299	952339

<i>Area (um2)</i>	<i>Baseline</i>				<i>TIMBER [31]</i>				<i>Proposed</i>			
	<i>#cells</i>	<i>cell area</i>	<i>net area</i>	<i>total</i>	<i>#cells</i>	<i>cell area</i>	<i>net area</i>	<i>total</i>	<i>#cells</i>	<i>cell area</i>	<i>net area</i>	<i>total</i>
<i>Fetch stage</i>	1976	4980	6714	11695	2014	5217	6928	12145	1751	4391	6236	10627
<i>Decode stage</i>	1874	3316	6640	9955	1900	3264	6677	9940	1796	2768	6430	9199
<i>Execute stage</i>	16414	44741	62899	107639	16356	46919	62784	109702	15053	39790	58245	98034

4.5 Conclusion

In this chapter, we propose a non-speculative pipeline design methodology to relax the slack margins proportionate to the available slack using slack balancing flip-flops (SBFFs). The proposed design methodology handles worst-case variations by sampling the data at a delayed clock edge. The additional slack margin obtained from SBFFs is used to downsize the critical fan-in logic for power and area savings. Existing speculative error detection schemes have critical operating point behaviour and their effectiveness largely depend on the error rate at any given point of time. The proposed approach is non-speculative which eliminates critical wall of slack behaviour. The speculative techniques need to be periodically tuned to adapt to the ambient operating conditions and workload. They rely on run-time voltage scaling which results in meta-stability issues and compromises the functionality of the chip. The deterministic timing closure used in the proposed approach eliminates meta-stability and error correction overheads. Moreover the logic downsizing eliminates the need for hold buffers. Experimental results of the proposed design methodology show a power and area savings of 12% and 8% respectively compared to the worst-case design.

CHAPTER 5. LIBRARY PRUNING AND SIGMA CORNER LIBRARIES FOR POWER EFFICIENT VARIATION TOLERANCE

In this chapter, we propose two power optimization techniques based on standard cell libraries to design a 40nm processor pipeline. We use slack balancing flip-flops (SBFFs) at places with sufficient slack presence along the critical paths to borrow time. The extra time margin created by the SBFF is used to under design their critical fan-in logic using BTWC sigma corner library cells. Figure 61 shows the area speed trade-off for a typical processor pipeline for the worst-case (WC) 3sigma and the BTWC 2sigma process types. The results show that for the same logic area, a 2sigma design can be signed off at a higher frequency B than the 3sigma design frequency A. Also for the same synthesis period, a 2sigma design consumes a lesser logic area C compared to the logic area A of a 3sigma design. This proves the power reduction capabilities of a BTWC sigma design. We also evaluate our SBFF library

Figure 61. Area Vs Speed trade-off for different process conditions.

Figure 62. Power comparison between master-slave flip-flop (MSFF) and error tolerant slack balancing flip-flop (SBFF) standard cells.

based on their power, delay and area cost metrics prune the outlier cells. Figure 62 shows the power comparison between the normal master-slave flip-flops (MSFF) and the SBFFs.

5.1 Proposed Design Methodology

5.1.1 Slack Balancing

After investigating a three-stage processor pipeline, we see that all the critical paths (~420) in a particular module in the execute (*EXE*) pipeline stage have considerable slack in the consecutive stage as shown in Figure 63. So, we relaxed the design margins of the critical paths using a BTWC sigma corner and sample the delayed inputs using our slack balancing flip-flops [16]. To explain slack balancing, we consider a small circuit as shown in Figure 64 with four registers. This circuit is optimized for a minimum feasible clock period $T=11$. Combinational path (*d*, *b*) has the maximum delay. For this critical path, there is an opportunity to borrow

Figure 63. Critical path slacks and consecutive path slacks in the execute (EXE) pipeline stage module M1 of a processor pipeline.

Figure 64. A processor pipeline with SBFF and BTWC sigma corner logic.

some time from the consecutive stage (*b*, *d*) whose delay is 9. So, we replace the flip-flop *b* by the SBFF with a clock delay equal to the slack present in the consecutive stage (*b*, *d*) which is

2. Now for the same clock period $T=11$, we have an extra margin of 2 in the critical path (d , b) which can be used to swap the logic in that path with a BTWC sigma corner logic. This gives an opportunity to optimize power and area in the critical fan-in logic of the flip-flop b .

Figure 46. (a) SBFF architecture and (b) timing diagram.

The slack balancing flip-flop architecture and the clock control circuit is illustrated in Figure 65. This architecture is a simplified version of *TIMBER* [31]. The data path has a main latch and a slave latch forming an edge triggered master-slave flip-flop. The delayed input DATA signal is detected by the shadow latch which uses a phase shifted version of the clock (DCK). As shown in Figure 65(b), the main latch samples the input data and transfers it to the output (Q) at the positive clock (CK) edge when TG0 is closed and TG1 is open. Similarly, the slave latch samples the data and passes it to the output at the delayed clock edge (DCK) when TG1 is closed and TG0 is open. The data in the master latch is compared with that in the slave latch by an XOR gate. The input to the slave latch is switched between the master path and the shadow path using the control signals (P0 and P1) to ensure that the delayed data passes through the slave latch. The XOR output signal is sampled by P1 to avoid sampling any glitches to generate ACTIVITY MONITOR. Based on the activity rates in DATA signal and the logic delay, the master latch and shadow latch will sample the same value or different values, provided a data transition happens in the clock cycle. ACTIVITY MONITOR signal is a measure real-time data rates.

5.1.2 Design Flow

The proposed design flow using SBFF is shown in Figure 66. The standard cell library with 16 flavours of SBFFs is developed using Cadence Liberate. The SBFF library is pruned to filter out the cells with huge power/area overheads compared to their MSFF counterparts. The statistical sigma corner libraries were developed using Cadence Variety. We begin with static timing analysis (STA) of the design and based on the results we filter the critical paths whose slack is below a certain threshold Th . We look for modules whose critical endpoints have sufficient consecutive slack. We replace the critical fan-in logic cells of those modules with their BTWC sigma corner equivalents which gives power and area savings. The endpoints are replaced by an SBFF with the clock delay proportional to the consecutive slack. The synthesis engine is amended with custom scripts to enable slack

analysis and SBFF replacement. We use existing CAD tools, static timing analysis, and custom add-on scripts for SBFF insertion and logic optimization.

5.2 Power Optimization Techniques

With advanced technology nodes, the design complexity is very high with the need to tackle the power performance trade-offs in a more efficient way. Once the design is reinforced with SBFFs, we use power optimization techniques to get power and area savings without compromising the reliability. Power efficient libraries help to bring down the optimization efforts considerably. In this paper, we use Library pruning and BTWC sigma corner libraries for power optimization.

Figure 47. Proposed design flow with BTWC sigma corner logic and SBFF.

Table 6. SBFF flavors used in the proposed design methodology.

<i>Reference</i>	<i>Drive strength</i>	<i>Description</i>
<i>FF_1, FF_2</i>	<i>1X, 2X</i>	D-Flip-flop, positive-edge triggered, q-only
<i>FF_3, FF_4, FF_5</i>	<i>1X, 2X, 8X</i>	D-Flip-flop, positive-edge triggered, q-only
<i>FF_6</i>	<i>1X</i>	D-Flip-flop, positive-edge triggered, low-asynchronous-clear, q-only
<i>FF_7, FF_8, FF_9</i>	<i>1X, 2X, 8X</i>	D-Flip-flop, positive-edge triggered, low- asynchronous -clear, q-only
<i>FF_10, FF_11</i>	<i>1X, 2X</i>	D-Flip-flop, positive-edge triggered, low- asynchronous -clear/set, q-only
<i>FF_12, FF_13, FF_14</i>	<i>1X, 2X, 8X</i>	D-Flip-flop, positive-edge triggered, low- asynchronous -clear/set, q-only
<i>FF_15, FF_16</i>	<i>1X, 2X, 8X</i>	D-Flip-flop, positive-edge triggered, low- asynchronous -clear/set, sync hold, q-only

5.2.1 Library Pruning

Error-tolerant techniques require special error handling flip-flops to detect and mask the timing errors. We use SBFF as the error detection element in this paper. In order to enhance the processor pipeline, we developed 16 different flavours of SBFFs as shown in Table 8. We characterized the SBFFs and compared the power, area and delay metrics with the corresponding master-slave flip-flops. Figure 67 shows the power and area comparison between the different flip-flop pairs from MSFF and SBFF library. It is evident that few pairs like FF_5, FF_9, and FF_14 consume twice the power compared to other MSFFs and SBFFs in the library. Area of all the cells is twice as compared to the SBFF cells. Figure 68 shows the timing comparison between the flip-flop pairs of the SBFF and MSFF library. The C to Q delays for the FF_5, FF_9, and FF_14 SBFFs is ~30% more than that of the corresponding MSFFs. This analysis helps to prune the SBFF library cells based on power, area and delay metric to avoid the outlier cells which are power and area hungry.

Figure 48. (a) Power comparison between MSFF and SBFF standard cells and (b) Area comparison between MSFF and SBFF standard cells.

5.2.2 Sigma corner library cell swaps

Figure 49. (a) Delay comparison between MSFF and SBFF standard cells and (b) Setup time comparison between MSFF and SBFF standard cells.

In conventional design flows timing sign off is done using the worst-case library which is pessimistic and results in power and area overheads. So, we developed a BTWC sigma corner library based on relaxed design margins. This library is used to synthesis the design modules which has an extra slack presence. We choose our statistical sigma corner libraries from σ_0 , σ_1 , σ_2 , and σ_3 each with different sigma scaling parameters ranging from 0 to 1 with σ_3 being the worst-case corner with pessimistic design margins as shown in Figure 69(a). Also, we have two BTWC voltage corners characterized for each σ corner. The V_{10p} library represents the WC voltage corner and the V_{5p} library represents the BTWC voltage corner with relaxed voltage margins. Figure 69(b) shows the delay values of a standard inverter cell for the different library variants. It is evident that as we sweep the sigma values right to left from worst-case sigma corner to BTWC corner, the inverter delay reduces. Also,

Figure 50. (a) Relaxed design margins with sigma corner libraries and (b) Delay reduction with sigma scaling and voltage scaling.

Algorithm 1 BTWC sigma library cell swap (SigmaOpt)

1. **Procedure** *SigmaOpt* (*Initial Netlist*)
 2. Run *STA* to find critical paths P and consecutive slacks
 3. $P \leftarrow \Phi$
 4. **for all** timing endpoints p in the *netlist* **do**
 5. **if** $slack(p) < Th$ **then**
 6. $P \leftarrow P \cup \{p\}$
 7. **end if**
 8. **end for**
 9. **for all** $\sigma = 0, \sigma \leq 2, \sigma = \sigma + 1$ **do**
 10. **for all** $i = 0, i \leq m, i = i + 1$ **do**
 11. Find modules M_i with *consecutive slack* (P_i) $\geq TB$
 12. Replace registers at endpoints P_i by *SBFF* with $DCK=TB$
 13. Adjust timing of fan-in logic by TB
 14. Swap the logic library with σ library
 15. Close timing w.r.t DCK
 16. **end for**
 17. Calculate Power and Area savings
 18. **return** ($netlist_{\sigma}$)
 19. **end for**
-

Figure 51. SBFF replacement and sigma based optimization algorithm.

once we tune the voltage from V_{10p} to V_{5p} corner, the inverter delay reduces. This helps to leverage the BTWC corner libraries for power optimization while using SBFFs for pessimistic worst-case variations. Based on this, we propose an algorithm *BTWC sigma library cell swap* (*SigmaOpt*) for power optimization as shown in Figure 70. The procedure takes an *Initial Netlist* which is designed based on worst-case corners. The procedure runs static timing analysis and filters the paths P with *slack* less than a predetermined threshold Th . In our experiments, we mark paths with less than 2% slack margin critical. The procedure looks for design modules M_i which can be relaxed by $TB = \{TB1, TB2, TB3 \text{ or } TB4\}$. We use $TB1 = T_{ck}/8$, $TB2 = T_{ck}/4$, $TB3 = 3T_{ck}/8$ and $TB4 = T_{ck}/2$ for our experiments. Once we find a module M_i which can be relaxed by TB , we replace all the critical endpoint registers with an *SBFF* whose clock is delayed by TB . This is followed by swapping the critical fan in logic with a *BTWC sigma* corner library. Once a module is relaxed with the chosen *sigma* corner library, we look for other modules which satisfy the same *consecutive slack* criteria. This procedure generates a netlist *netlist_{sigma}* for the *sigma* corner input library set.

5.3 Results and Analysis

We use a typical three-stage processor pipeline in 40nm technology node for our experiments. After slack analysis, it was found that all critical paths (~420) in a design module in the *EXE* stage of the processor pipeline can be relaxed by $TB2$. This allowed us to redesign the critical fan-in logic of the module using the proposed library pruning and BTWC sigma corner libraries. We designed the pipeline using a worst-case 3sigma process corner with 10% voltage variation ($P_{3\sigma}$, V_{10p}) using normal MSFFs as the reference benchmark (*WC*). We evaluated the power and area for a typical error tolerant flip-flop TIMBER [31] based pipeline at the same worst process and voltage corner ($P_{3\sigma}$, V_{10p}). To prove the proposed power optimization techniques, we chose 1sigma process corner with 5% voltage variation ($P_{1\sigma}$, V_{5p}) to design the processor pipeline. We implemented two sets of designs on this BTWC corner, one design (*BTWC1*) with the full 16 cell SBFF library and the second design with a pruned version of the SBFF library (*BTWC2*). Table 9 shows the power comparisons between the baseline, TIMBER and the proposed BTWC design and Table 10 shows the area comparisons. Figure 71. shows the power and area of the different designs. We get a power saving of 39% for the *BTWC1* technique and 47% for the pruned *BTWC2* technique. Similarly, we get area savings of 2.5% and 2.8% for the *BTWC1* and *BTWC2* respectively.

5.4 Conclusions

In this chapter, we propose a power efficient methodology to design error tolerant pipelines using library tuning techniques. We find an opportunity to relax the slack margins of a processor pipeline using slack balancing flip-flops. The extra slack margin is leveraged to under design the critical fan-in logic using BTWC sigma corner libraries. We also propose a

Table 7. Power savings of the proposed design methodology.

Power (nW)	Baseline				TIMBER [31]				Proposed			
	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total
Execute	2567	1680.6	275349.6	277030.2	2567	2055.2	355132.6	357187.8	1887	2694.1	165850.3	168544.4

Table 8. Area savings of the proposed design methodology.

Area (um ²)	Baseline				TIMBER [31]				Proposed			
	#cells	cell area	net area	total	#cells	cell area	net area	total	#cells	cell area	net area	total
Execute stage	2567	6967	10619	17586	2567	9686	10615	20301	1887	8374	8764	17138

Figure 52. SigmaOpt and Library Pruning power optimization techniques (a) Power savings, (b) Area savings with respect to worst-case design.

library pruning technique to filter out power hungry SBFFs from the design flow. Experimental results show power and area savings of 47% and 2.8% respectively compared to the worst-case design when we use our better than worst-case methodologies based on for the proposed techniques. This design-time error tolerance is non-speculative which eliminates critical wall of slack behaviour.

CHAPTER 6. POWER AND AREA EFFICIENT CLOCK STRETCHING AND CRITICAL PATH RESHAPING FOR ERROR RESILIENCE

6.1 Motivation

Typical processor pipelines are highly unbalanced with respect to the slack present in various combinational logic stages. Traditional logic synthesis tools focus on optimizing the combinational logic and do not support any trade-offs between the combinational paths separated by sequential elements. Sequential optimizations like retiming may introduce additional power and area overheads. Figure 72 shows the effect of retiming on the processor pipeline. It is obvious that the number of critical endpoints has doubled after retiming and there is hardly any slack improvement. In contrast to the existing techniques, we come up with a design-time optimization which is energy efficient and reliable. We use a *40nm* processor core with *26K logic gates* and *7000 flip-flops*. We chose the most critical endpoints with slack less than 2% of the clock period. Based on our investigation, we see that for most of the timing paths in the near critical slack region (*pipe_1 slack*) there is sufficient consecutive stage slack (*pipe_2 slack*) as shown in Figure 72(c). Results show that 85% of the critical paths get a mean slack improvement of 42X if they borrow slack from the consecutive stage. We leverage this slack to relax the design margins of the critical paths and sample the delayed inputs using a clock stretched flip-flop. This allows us to reshape the combinational logic along most of the critical paths to get power and area savings. Instead of reducing design margins at runtime using adaptive techniques, the proposed approach under designs the critical logic of the pipeline to get power savings. This deterministic slack analysis approach offloads the speculative overheads associated with a typical error resilience techniques. The design margin improvement along the critical paths helps the pipeline to tolerate more process variations. Worst-case design margins are still met after under designing with respect to the delayed clock edge of the flip-flop. This helps to eliminate all the timing speculation overheads and the critical wall of slack issues related to adaptive in-situ error correction schemes. Unlike a fixed speculation window,

Figure 53. (a) Critical path slacks (pipe_1) (b) retimed pipe_1 slack and (c) consecutive stage slacks (pipe_2) of the processor.

the

Table 9. Comparison of error resilience techniques.

<i>Feature</i>	<i>Design level optimization</i>	<i>Sequential Optimization based on error detection</i>	<i>Combinational & Sequential Optimization</i>	<i>Sequential Optimization based on error masking</i>	<i>Proposed</i>
Speculation mechanism	ABB, ASV, OSB, PCT	Adaptive voltage/frequency scaling	Adaptive voltage/frequency scaling	Adaptive voltage/frequency scaling	Non-speculative, based on cell downsizing and BTWC library
Error handling	Duplicate paths	Duplicate Latch/FFs	Duplicate Latch/FFs	Duplicate Latch/FFs	No error
Clock tree loading	No	Yes	Yes	Yes	No
Short path padding	No	Yes	Yes	Yes	Yes
Sequential overhead	Large	Large	Large	Large	None
Combinational overhead	Small	Small	Small	Small	None
Meta-stability	Yes	Yes	Yes	Yes	No
Techniques	EVAL, Blueshift	Razor I, Razor II, Bubble Razor, DSTB, TDTB	Slack Optimizer, Skew Optimizer, CombOpt	TIMBER, soft edge flip-flop	Clock stretched flip-flop

MS = Meta-stability, Tw = error detection window, Tck= clock period.

proposed method uses an elastic window which allows timing speculation proportional to the available slack. Table 11 summarises the features of the different error resilience techniques.

6.2 Proposed Design Methodology

6.2.1 Slack Balancing

To explain the proposed slack balancing technique, we consider a small circuit as shown in Figure 73 with four registers. This circuit is optimized for a minimum feasible clock period $T=11$ under worst operating conditions. Combinational path (d , b) has the maximum delay which is the critical path. For this path, there is an opportunity to borrow some time from the

Figure 54. A processor pipeline with CSFF and under designed logic.

consecutive stage (b, d) whose delay is 9. So, we replace the flip-flop b by the clock stretched flip-flop with a clock delay equal to the slack present in the consecutive stage (b, d) which is 2. Now for the same clock period $T=11$, we have an extra margin of 2 in the critical path (d, b) which can be used to under design the logic for power and area savings. We use two techniques to under design the critical path logic. The first one relaxes the critical path slack by the extra slack margin available and redesigns the logic with smaller cell equivalents from the same logic library. The second technique identifies modules with sufficient consecutive

Figure 55. Redundant nodes in typical error resilient flip-flops.

path slacks and swaps the critical path logic library with a BTWC sigma corner library. Thus, we trade-off the extra slack margin for power and area savings using the slower cells in the sigma corner library.

6.2.2 Clock Stretched flip-flop

In this section, we will explain the proposed clock stretching scheme [45] to relax the design margins on the critical paths. These flip-flops aims to simplify the design compared to other error resilient circuits which have redundant nodes as shown in Figure 74. As shown in Figure 75, the flip-flop architecture is similar to master-slave flip-flop except for the clock control signal P . The stretched clock P allows the data input to change beyond the positive clock edge. P is derived from the original clock CK and the delayed version of the clock DCK . During the low phase of P , transmission gate $TG0$ is open whereas $TG1$ is closed and master latch $L0$ samples the input data. During the low to high transition of P , transmission gate $TG1$ is open and $TG0$ closed and the shadow latch $L1$ sample the data to output. The transparency window for the master latch $L0$ allows the delayed input to be sampled to the intermediate node. This window determines the time borrowed from the consecutive stage. The time borrowed TB is equal to the phase difference between CK and DCK . We choose four TBs which are multiples of $T_{ck}/8$ for each critical endpoint register based on the slack available in the consecutive stage, where T_{ck} is the clock period. Our approach has no redundant data sampling and error handling circuitry compared to other resilience circuits because we relax a fixed amount of slack at each endpoint during design-time. The data input D is timed in such a way it maintains pessimistic design margins with respect to the clock edge P which avoids meta-stability overheads in the design. We also propose a multi-bit flip-flop structure to combine the individual flip-flop data paths which give additional power and area savings as detailed in the results section. This transistor count of the proposed error resilience flip-flop is half of that of other error resilient architectures resulting in a considerable area and power saving.

Figure 56. (a) CSFF [45] architecture (b) timing diagram and (c) Multi-bit flip-flop structure.

Multi-bit flip-flop is an effective power reduction technique which is commonly used in low power designs as shown in Figure 76. The MBFF standard cell library used in our design has 36 flavours of flip-flops with 18 dual bit and 18 quad bit flip-flops respectively. The multi-bit flip-flops are of HVT flavour with 1x and 2x drive strength scan able and non-scan able variants. We use the Cadence native MBFF flow to infer multi-bit flip-flops in the logic

Figure 57. Multibit flip-flop.

synthesis stage as shown in Figure 77. The MBFF inserted design is compared to the baseline design with normal flip-flops to calculate the power savings at each design stage. We tried

Figure 58. Multi-bit flip-flop synthesis flow.

multiple MBFF synthesis attributes to analyze the trade-offs between MBFF coverage and power savings. Based on post-synthesis and post CCOpt results, it was found that we get maximum power saving with power-area based multi-bit merging from different buses. The

max power saving is for quad bit MBFF as shown in Table 12. Also, the delay worsens with the number of bits.

Table 10. Max power saving for quad bit MBFF.

<i>Bit number</i>	<i>Dynamic power</i>	<i>Leakage power</i>	<i>Area</i>	<i>Delay</i>
1	1	1	1	1
2	0.8765	0.83	0.89	1.01
4	0.81	0.77	0.8421	1.028

Figure 59. Proposed design flow with CSFF.

6.2.3 Design Flow

The proposed design flow using CSFF is shown in Figure 78. We take the processor design through Cadence RTL Compiler synthesis run, placement and optimization in Encounter. The design signed off at worst corner is taken as the baseline. We used the critical endpoints report from RTL Compiler and the slack report from the Encounter run to filter those paths whose slack is below a certain threshold T_h . After this we analyse the consecutive stage slack of these critical endpoints and categorise them with a time borrowing window TB based on the slack available. This is followed by their clocks being stretched by using a DCK which is phase shifted by TB . In the next step, we optimize the combinational fan-in logic power of those endpoints whose slack is relaxed. We look for logic modules in which all the critical paths have sufficient slack in the consecutive stage. We replace the logic library of those modules with a BTWC sigma corner library and redesign the pipeline with the new library. For

the other critical endpoints outside the module, the individual slacks are relaxed and the pipeline is re-synthesised based on the new slack. We use existing CAD tools, static timing analysis and custom add-on scripts for the logic optimization. We characterised 16 flavours of TIMBER flip-flops using Cadence Liberate to compare the results of our approach with typical error resilience circuits. The BTWC sigma corner libraries were developed using Cadence Variety. We used Cadence Multi-bit flow to combine the flip-flops using add on scripts during the synthesis phase.

6.3 Power Optimization Techniques

The design complexity is high for advanced technology nodes with the need to tackle the power performance trade-offs in a more efficient way. Once the design is reinforced with clock stretching circuits, we use custom logic power optimization techniques to get power and area savings without compromising the reliability. In this paper, we use logic reshaping and BTWC sigma corner libraries for power optimization.

6.3.1 Power optimization by logic reshaping (SizeOpt)

The critical path logic of typical worst-case corner designs is made up of power and area hungry cells to meet the delay requirements. With rising design margins in advanced nodes, more power is being spent on optimizing this logic to meet the guard bands. We propose a technique [45] to utilize available slack and redistribute it to the critical path logic using CSFF.

Algorithm 1 Slack analysis & Downsizing (SizeOpt)

1. **Procedure** *SizeOpt* (*Initial Netlist*)
 2. Run *STA* to find critical paths P and consecutive slacks
 3. $P \leftarrow \Phi$
 4. **for all** timing endpoints p in the *netlist* **do**
 5. **if** $slack(p) < Th$ **then**
 6. $P \leftarrow P \cup \{p\}$
 7. **end if**
 8. **end for**
 9. **for all** $TB = TB4, TB \geq TB1, TB = TB - TB1$ **do**
 10. **for all** $p \in P$ **do**
 11. **if** $consecutive\ slack(p) \geq TB$ **and** $< 2TB$ **then**
 12. Replace register CLK with stretched CLK , with $DCK=TB$
 13. Relax fan-in logic timing by TB
 14. **end if**
 15. **end for**
 16. **end for**
 17. Redesign the pipeline and close timing
 18. Calculate Power and Area savings
 19. **return** (*netlist*)
-

Figure 60. Pseudo code for slack analysis and reshaping.

This creates extra slack margins in the combinational paths preceding the critical endpoints. We leverage this slack to relax the path timing constraints. Once we relax the path slacks, we resynthesize the pipeline. The synthesis tool picks smaller cell equivalents from the logic library which leads to smaller and fewer cells in the critical path resulting in power and area savings. The extra slack margin helps to downsize the logic at the expense of cell delay increase. Figure 79 shows the pseudo code (*SizeOpt*) for the slack analysis and critical path reshaping. P represents the critical paths under consideration and S represents the corresponding consecutive slacks. For each critical path in P, we choose a DCK phase shift from $TB = \{TB1, TB2, TB3 \text{ or } TB4\}$. We fixed TB value as a multiple of $T_{ck}/8$, with $TB1=T_{ck}/8$, $TB2= T_{ck}/4$, $TB3= 3T_{ck}/8$ and $TB4= T_{ck}/2$ for our experiments. For reshaping, we relax the timing margin on selected paths by TB and resynthesize the logic so as to get power and area savings. This process relaxes the short paths as well as long path slacks in the critical fan-in. Thus, the short paths get the same delay increase as the long path. This prevents the need for additional buffers along the short paths.

6.3.2 Power optimization by library tuning (SigmaOpt)

In conventional design flows timing sign off is done using the worst-case library which is pessimistic and results in power and area overheads. So, we developed a BTWC sigma corner library based on relaxed design margins. This library is used for logic synthesis in those design modules which has an extra slack presence. After investigating the three-stage processor pipeline, we see that all the critical paths (~390) in a particular module in the execute (*EXE*) pipeline stage have considerable slack in the consecutive stage. This allowed us to use a BTWC sigma corner to design that particular logic module. Figure 80(a) shows the area speed trade-off for a typical processor pipeline for the worst-case (WC) 3sigma and a BTWC 2sigma process library. The results show that for the same logic area, a BTWC design can be signed off at a higher frequency B than the 3sigma design frequency A. Also for the same synthesis period, a 2sigma design consumes a lesser logic area C compared to the logic area A of a 3sigma design. This proves the area as well as power reduction capabilities of a BTWC sigma library design.

Figure 80. (a) Area Vs Speed trade-off for different process conditions and (b) Delay reduction with sigma scaling and voltage scaling.

We did library characterizations for different sigma scaling parameters ranging from 0 to 1. Sigma scaling parameter of 1 represents *sigma3* which is the worst-case corner with pessimistic design margins. *Sigma2*, *sigma1*, and *sigma0* have scaling parameters of 0.66, 0.33 and 0 respectively with *sigma 0* being the optimistic library corner. Each sigma corner library was characterized by two sets of voltage corners. The V_{10p} library variant represents the WC voltage corner and the V_{5p} version represents the BTWC voltage corner with relaxed voltage margins. Figure 80(b) shows the delay values of a standard inverter cell for the different library variants. It is evident that as we sweep the sigma values right to left from WC sigma corner to BTWC corner, the inverter delay reduces. Also, once we tune the voltage from V_{10p} to V_{5p} corner, the inverter delay reduces. Thus, we achieve faster timing closure with fewer resources to meet the worst design margins. We leverage the BTWC corner libraries for power optimization while using CSFFs for pessimistic worst-case variations in real silicon. Based on this, we propose an algorithm BTWC sigma library cell swap (*SigmaOpt*) for power optimization as shown in Figure 81. The procedure takes an *Initial Netlist* which is designed based on worst-case corners. The procedure runs static timing analysis and filters the paths P with *slack* less than a predetermined threshold Th . In our experiments, we mark paths with less than 2% slack margin

Algorithm 2 BTWC sigma library cell swap (*SigmaOpt*)

```

1. Procedure SigmaOpt (Initial Netlist)
2. Run STA to find critical paths  $P$  and consecutive slacks
3.  $P \leftarrow \Phi$ 
4. for all timing endpoints  $p$  in the netlist do
5.   if slack( $p$ ) <  $Th$  then
6.      $P \leftarrow P \cup \{p\}$ 
7.   end if
8. end for
9. for all  $\sigma = 0, \sigma \leq 2, \sigma = \sigma + 1$  do
10.  for all  $i = 0, i \leq m, i = i + 1$  do
11.    if module  $M_i$  consecutive slack ( $P_i$ )  $\geq TB$  then
12.      Replace register  $CLK$  with stretched  $CLK$ , with  $DCK=TB$ 
13.      Swap the logic library with  $\sigma$  library
14.    end if
15.  end for
16.  Redesign the pipeline and close timing
17.  Calculate Power and Area savings
18.  return (netlistsigma)
19. end for

```

Figure 81. Pseudo code for sigma corner based optimization.

critical. The procedure looks for design modules M_i which can be relaxed by $TB = \{TB1, TB2, TB3 \text{ or } TB4\}$. We use $TB1=T_{ck}/8$, $TB2= T_{ck}/4$, $TB3= 3T_{ck}/8$ and $TB4= T_{ck}/2$ for our experiments. Once we find a module M_i which can be relaxed by TB , we use the clock stretcher to delay their clocks by TB . This is followed by swapping the critical fan in logic with a *BTWC sigma* corner library and redesigning the pipeline for power and area savings. Once a module is relaxed with the chosen *sigma* corner library, we look for other modules which satisfy the same *consecutive slack* criteria. This procedure generates a netlist *netlist_{sigma}* for the *sigma* corner input library set.

6.4 Results and Analysis

6.4.1 Flip-flop level savings

Figure 62. Area reduction compared to TIMBER flip-flop.

Table 11. Power delay savings of Proposed circuit

<i>Variables</i>	<i>Timber FF</i>	<i>Proposed</i>
<i>C to Q delay rising</i>	91.62ps	77.5ps
<i>C to Q delay falling</i>	91.83ps	80.67ps
<i>Max power@ 1.1V</i>	3.21uW	2.608uW
<i>Min Power@1.1V</i>	1.857uW	1.478uW

Figure 83. Path relaxation in different modules of a processor pipeline.

We use a simplified flip-flop structure to balance the slack between different pipeline stages. Compared to the reference *TIMBER* flip-flop, the proposed circuit is similar to a standard master-slave flip-flop which makes it easy for the design tools to use it in the design flow. Figure 82 shows the layout for the proposed flip-flop and *TIMBER* flip-flop. An area reduction of 60% compared to *TIMBER* is attained per flip-flop for the proposed scheme. Table 13 shows the flip-flop level savings such as C to Q delay, minimum power, and maximum power compared to *TIMBER*. We get a rising C to Q delay reduction of 18.21%, falling C to Q delay reduction of 13.83%, maximum power saving of 23% and minimum power saving of 25.6% compared to the reference *TIMBER* flip-flop.

6.4.2 Chip level savings

The proposed methodology is used to design the critical pipeline stages of an industrial processor. Power and area comparisons are being done against the baseline designed for worst-case and with a *TIMBER* based error resilient architecture. The reference design (*WC*) used worst-case 3sigma process corner and 10% voltage variation ($P_{3\sigma}$, V_{10p}) with normal MSFFs. *TIMBER* [31] based pipeline used the same worst process and voltage corner ($P_{3\sigma}$,

Table 15. Power savings of the proposed design methodology

Power (μW)	Baseline				TIMBER [31]				Multibit				SizeOpt				SigmaOpt			
	#cells	Pl	Pd	Ptot	#cells	Pl	Pd	Ptot	#cells	Pl	Pd	Ptot	#cells	Pl	Pd	Ptot	#cells	Pl	Pd	Ptot
Fetch	1976	1.5	265.1	266.6	2014	1.5	482.4	483.9	1920	1.5	261.2	262.7	1751	1.1	184	185.1	1976	1.5	265.1	266.6
Dec.	1874	0.9	149.1	150	1900	0.8	154.8	155.6	1867	0.9	144.9	145.8	1796	0.7	157.3	158	1874	0.9	149.1	150
Exe.	16414	14.8	1307.9	1322.7	16356	14.7	1777.4	1792.2	15964	14.8	1262.2	1277	15053	11.2	471.7	482.9	15734	15.5	1118.6	1134.1

Table 16. Area savings of the proposed design methodology

Area (mm ²)	Baseline				TIMBER [31]				Multibit				SizeOpt				SigmaOpt			
	#cells	Acell	Anet	Atot	#cells	Acell	Anet	Atot	#cells	Acell	Anet	Atot	#cells	Acell	Anet	Atot	#cells	Acell	Anet	Atot
Fetch	1976	0.005	0.007	0.012	2014	0.005	0.007	0.012	1920	0.005	0.007	0.012	1751	0.004	0.006	0.01	1976	0.005	0.007	0.012
Dec.	1874	0.003	0.007	0.01	1900	0.003	0.007	0.01	1867	0.003	0.007	0.01	1796	0.003	0.006	0.009	1874	0.003	0.007	0.01
Exe.	16414	0.045	0.063	0.108	16356	0.047	0.063	0.11	15964	0.045	0.062	0.107	15053	0.038	0.058	0.096	15734	0.043	0.061	0.105

V_{10p}). Three power optimization techniques were used to achieve power and area savings against the worst-case baseline design. The *Multibit* technique combined the individual flip-flops in the design into dual bit and quad bit flip-flops. AS shown in Figure 83, the fetch stage has 39 critical paths with sufficient consecutive slack, and execute stage has 377 paths with sufficient slack. Module M1 in EXE stage has sufficient consecutive slack present in all the 343 critical paths. The *SizeOpt* technique used clock stretching and combinational logic reshaping along the critical paths with sufficient slack using the worst-case 3sigma process corner and 10% voltage variation ($P_{3\sigma}$, V_{10p}). The *SigmaOpt* technique used clock stretching and 1sigma process corner with 5% voltage variation ($P_{1\sigma}$, V_{5p}) to design module M1 in the EXE pipeline stage. Table 17 shows the power comparison results between *WC* baseline, *TIMBER*, and the proposed *Multibit*, *SizeOpt* and *SigmaOpt* schemes in the fetch, decode and execute pipeline stages. Table 18 shows the area comparison between baseline, *TIMBER* and proposed schemes. Figure 84(a) and Figure 84(b) shows the power and area comparisons between the baseline, *TIMBER* and the proposed schemes. Power comparison results show 81% power overhead for *TIMBER*, 1.4% power savings for *Multibit* and 31% power savings for the *SizeOpt* in the fetch stage against the *WC* baseline design. In the execute pipeline stage, *TIMBER* has a power overhead of 36% compared to the baseline whereas the proposed schemes give a power saving of 3.5%, 63% and 37% for the *Mutibit*, *SizeOpt*, and *SigmaOpt*

Table 17. Power savings of the combined optimizations

Power (μW)	Baseline				TIMBER [31]				CombOpt			
	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total
Fetch stage	1976	1.5	265.1	266.6	2014	1.5	482.4	483.9	1695	1.1	180.1	181.2
Decode stage	1874	0.9	149.1	150	1900	0.8	154.8	155.6	1789	0.7	153.1	153.8
Execute stage	16414	14.8	1307.9	1322.7	16356	14.7	1777.4	1792.2	14012	11.968	396.947	408.915

Table 18. Area savings of the combined optimizations

Area (mm^2)	Baseline				TIMBER [31]				CombOpt			
	#cells	cell area	net area	total	#cells	cell area	net area	total	#cells	cell area	net area	total
Fetch stage	1976	0.005	0.007	0.012	2014	0.005	0.007	0.012	1695	0.004	0.006	0.01
Decode stage	1874	0.003	0.007	0.01	1900	0.003	0.007	0.01	1789	0.003	0.006	0.009
Execute stage	16414	0.045	0.063	0.108	16356	0.047	0.063	0.11	14012	0.035	0.056	0.091

Figure 84. (a) Power comparisons and (b) Area comparisons between WC baseline, *TIMBER* [31] and the proposed schemes.

respectively. Area comparisons in the EXE stage show an area overhead of 2% for *TIMBER* and a saving of 1%, 11%, and 3% respectively for the *Multibit*, *SizeOpt* and *SigmaOpt* schemes against the baseline. The combined optimizations with multi-bit flip-flops, size optimization as well as sigma corner library represented by *CombOpt* gives an overall power and area savings as shown in Table 19 and Table 20 respectively. As shown in Figure 85(a), the combined optimizations give a power savings of 32% and 69% in the fetch and execute stages respectively. It also gives an area savings of 15% in the execute stage as shown in Figure 85(b). The chip specification is given in Table 21 and the chip layout is shown in Figure 86.

Figure 85. (a) Power comparisons and (b) Area comparisons between WC baseline, *TIMBER* [31] and the combined optimizations.

6.5 Conclusions

The proposed methodology is able to retain the throughput advantages of error masking circuits and at the same time reduce the power/area overheads compared to *TIMBER*. The margins reclaimed by the slack aware clock stretching during design-time is used to reshape the combinational circuits for power/area savings. Experimental results show a power and area saving of 32% and 16% respectively in the fetch pipeline stage. In the execute stage we get a power and area saving of 69% and 15% respectively. On the other and speculative techniques like *TIMBER* have power and area overhead of 36% and 2% respectively in the execute pipeline stage. Timing closure is done at design-time with respect to the stretched clock which

Table 19. Chip specifications.

Process (design)	Foundry	GlobalFoundries
	Process	CMOS040LP
	Metal	5 1x (Thin) metal low-K dielectric, 2 6x (Thick) metal in TEOS/FTEOS dielectric
	Code	C040LP 5L1x2T6x LB
Complexity	Std. cells:	148 K-gates
	Memory:	32KB ISRAM, 32KB DSRAM
	:	4KB ROM
	Analog:	DCO/PLL/Crystal
Main characteristics	Max. Frequency	180MHz

Figure 86. Chip layout.

removes meta-stability overheads. The combinational logic reshaping is done in a such a way that it adds delay to the short paths as well which prevents the need for additional hold buffers. Moreover, the proposed technique does not exhibit the critical operating point behaviour which makes it a reliable option for error resilience and power savings.

CHAPTER 7. DATA-DEPENDENT CLOCK STRETCHING FLIP-FLOP FOR POWER EFFICIENT VARIATION TOLERANCE

7.1 Motivation

In this chapter, we propose a power optimization technique based on clock stretching flip-flops (CSFF) and data-dependent clock stretching flip-flops (DDCSFF) to design a 40nm processor pipeline. We use two design-time variables n and α in (7.1) to reduce the dynamic power consumption in the processor pipeline. CSFFs are used to reduce the number of nodes n in the flip-flop by removing redundant data sampling compared to other error resilient circuits. Activity factor α is taken into account by using DDCSFF which results in lower power for low data activity rates. We use CSFF and DDCSFFs at places with sufficient slack presence along the critical paths to borrow time. The extra time margin created by the flip-flop is used to under design their critical fan-in logic by logic downsizing. Figure 87 shows the activity analysis results on a processor pipeline. We use five fft application programs to check the activity rate on the critical flip-flops. The results show that ~ 253 flip-flops have sufficiently low data activity rates. This creates an opportunity to use data-dependent clock stretching on those flip-flops which will reduce the overall power consumption of the processor subsystem. The paths with high data activity rates are build using clock stretching flip-flops. The proposed DDCSFF is an enhanced form of our previous clock stretched flip-flop for slack distribution along selected critical paths. The major contributions of this chapter are as follows.

- 1) We propose a design technique to filter out critical path flip-flops with sufficient slack in the consecutive stage and sufficient activity rate.
- 2) We relax the critical fan-in logic of the selected flip-flops to utilize the extra design margins. We replace low activity critical flip-flops with a data-dependent clock stretching version and the high activity ones with a clock stretching flip-flop. The results show considerable power and area saving.

Figure 63. (a) Flip-flop activity factor for fast_fft (b) Flip-flop activity factor for simple fft, fft32, fft64 and fft128.

$$P = n \cdot \alpha \cdot f \cdot 1/2CVdd^2 \quad (7.1)$$

7.2 Proposed Design Methodology

7.2.1 Slack Balancing

After investigating a three-stage processor pipeline, we see that all the critical paths (~420) in a particular module in the execute (EXE) pipeline stage have considerable slack in the consecutive stage. So, we relax the design margins of the critical paths using by under designing the critical logic and sample the delayed inputs using our choice of clock stretching flip-flops [16]. To explain slack balancing, we consider a small circuit as shown in Figure 88 with seven registers. This circuit is optimized for a minimum feasible clock period $T=11$. Combinational path (d, b) and (g, f) has the maximum delay. For this critical path, there is an opportunity to borrow some time from the respective consecutive stages (b, d) and (f, g) whose delay is 9. We do an activity factor check on the critical flip-flops b and f. We replace the flip-flop b which has a high activity factor by the CSFF with a clock delay equal to the slack present in the consecutive stage (b, d) which is 2. The flip-flop which has a low activity rate is replaced by a data-dependent CSFF. Now for the same clock period $T=11$, we have an extra margin of 2 in the critical paths (d, b) and (g, f) which can be used to reshape the logic in those paths. This gives an opportunity to optimize power and area in the critical fan-in logic of the flip-flops b and f.

Figure 89 shows the clock stretching circuit along with a master-slave flip-flop. The design is similar to master-slave flip-flop except for the clock control signal P. The stretched clock P allows the data input to change beyond the positive clock edge. P is derived from the original clock CK and the delayed version of the clock DCK. During the low phase of P, transmission

Figure 64. Slack balancing using CSFF and DDCSFF.

gate TG0 is open whereas TG1 is closed and master latch L0 samples the input data. During the low to high transition of P, transmission gate TG1 is open and TG0 closed and the shadow latch L1 sample the data to output. There is a transparency window for the master latch L0 compared to the normal master-slave flip-flops during which any delayed input is also sampled to the intermediate node. This window is decided by the delay clock DCK. We fix the delay time for DCK based on the slack analysis results. In our experiments, we use a value of $T_{ck}/8$, $T_{ck}/4$, $3T_{ck}/8$, and $T_{ck}/2$ for DCK delay proportionate to the consecutive slack, where T_{ck} is the clock period. There is no error speculation because the design margin relaxed at each critical path is fixed during the design-time based on the slack available in the consecutive stage. For dynamic voltage/frequency scaling, a transition detector can be used for error generation which is optional. There is no meta-stability issue as the input is not sampled very close to the clock transition. This approach has less overhead compared to the TIMBER circuit. TIMBER overheads include redundancy, error generation logic and clock control signal generation. Also, the speculative nature of error resilience necessitates an error propagation logic in TIMBER. Figure 90 shows data-dependent CSFF architecture. Here we add an XOR gate to check whether there is any change in output data Q with respect to the input D. We use latch-based clock gating circuit to enable the stretched clock P when there is a valid change in D with respect to Q and disable P when there is no change. This helps to reduce the power of the sequential element when the data activity is too low.

Figure 65. (a) CSFF [45] architecture (b) timing diagram.

Figure 66. (a) Data-dependent CSFF architecture (b) timing diagram.

Figure 67. Design flow for slack and data analysis and reshaping.

7.2.2 Design Flow

The proposed design flow using CSFF and DDCSFF is shown in Figure 91. We begin with static timing analysis (STA) of the design and based on the results we filter the critical paths whose slack is below a certain threshold T_h . We look for modules whose critical endpoints have sufficient consecutive slack. We use dynamic analysis to check the activity factor along the critical endpoints. Those endpoints are replaced by CSFF or DDCSFF depending on the activity factor, with the clock delay proportional to the consecutive slack. Once the design is reinforced with extra slack margins using the proposed flip-flops, we downsize the critical fan-in logic cells which give power and area savings. We use existing CAD tools, static timing analysis, and custom add-on scripts for flip-flop insertion and logic optimization. The synthesis engine is amended with custom scripts to enable slack analysis and flip-flop replacement.

7.3 Power Optimization Techniques

With advanced technology nodes, the design complexity is very high with the need to tackle the power performance trade-offs in a more efficient way. Once the design is reinforced with proposed flip-flops, we use a power optimization technique to get power and area savings without compromising the reliability. In this paper, we use logic reshaping for power optimization.

Algorithm 1 Slack and data analysis & reshaping

```

1. procedure Slack analysis (Initial Netlist)
2.    $P \leftarrow \text{Find Critical Paths } ()$ 
3.   for all  $p \in P$  do
4.      $S \leftarrow \text{Find Consecutive Slack } (p)$ 
5.   end for
6.   for all  $FF_{\text{endpoint}} \in P$  do
7.      $A \leftarrow \text{Find Activity Factor } (FF_{\text{endpoint}})$ 
8.   end for
9.   for all  $TB = TB1, TB \leq TB4, TB = TB + TB1$  do
10.    for all  $p \in P$  do
11.      if  $s \geq TB$  and  $< 2TB$  then
12.        if  $A \geq A_{Th}$  then
13.          Clock stretch p by TB
14.        else
15.          Gate the clock stretching signal
16.        end if
17.        Reshape p by TB
18.      end if
19.    end for
20.  end for
21. return (netlist)

```

Figure 68. Pseudo code for slack and data analysis and reshaping.

We use custom scripts for slack analysis that is plugged into the existing CAD tools. The critical paths under consideration are evaluated for consecutive stage slack. When there is no consecutive slack we do not modify the pipeline stage. When there is sufficient slack in second pipeline stage, we use the stretched clock for the critical endpoint register. This enables the critical stage to borrow a time TB proportionate to slack2. The pipeline stage 1 design margin is relaxed by TB. So, we reshape the critical path combinational logic proportionate to the time borrowed. This downsizes the critical path logic and results in power and area savings. Figure 92 shows the pseudo code for the slack analysis and critical path reshaping. P represents the critical paths under consideration and S represents the corresponding consecutive slacks. For each critical path in P, the clock stretching values are fixed from $TB1=T_{ck}/8$ to $TB4=T_{ck}/2$. The critical path is also reshaped by the same amount. For reshaping, we relax the timing margin on selected paths and resynthesize the logic so as to get power and area savings. The critical end-points tend to have the same range of consecutive slack. This helps to cluster the endpoints with the same pipe2 slack and use a common clock control for them.

7.4 Results and Analysis

We use a typical three stage processor pipeline in 40nm technology node for our experiments. After slack analysis, it was found that all critical paths (~420) in a design module in the EXE stage of the processor pipeline can be relaxed by TB2. This allowed us to redesign the critical fan-in logic of the module using the proposed methodology. Power and area comparisons are being done against the baseline designed for worst-case and with a TIMBER based error resilient architecture. The reference design (WC) used worst-case 3sigma process corner and 10% voltage variation (P3sigma, V10p) with normal MSFFs. TIMBER [31] based pipeline used the same worst process and voltage corner (P3sigma, V10p). The proposed technique used data-dependent clock stretching and combinational logic reshaping along the critical paths with sufficient slack using the worst-case 3sigma process corner and 10% voltage variation (P3sigma, V10p). Table 22 shows the power comparison results between WC baseline, TIMBER and the proposed schemes in the execute pipeline stage. Table 23 shows the area comparison between baseline, TIMBER and proposed scheme. Figure 93(a) and (b) shows the power and area comparisons between the baseline, TIMBER and the proposed scheme. In the execute pipeline stage, TIMBER has a power overhead of 36% compared to the baseline whereas the CSFF give a power saving of 64%. Once we use CSFF and DDCSFF together in the same design we get a power saving of 70% which is 6% more than using the CSFF alone. Area comparisons in the EXE stage show an area overhead of 2% for TIMBER and a saving of 11% for the CSFF scheme against the baseline. Once we use CSFF and DDCSFF together in the same design we get an area saving of 9.5% which is 1.5% less than using the CSFF alone. This is due to the added clock gating circuitry required for the DDCSFFs.

7.5 Conclusions

In this chapter, we propose a power efficient methodology to design error tolerant pipelines using data-dependent clock stretching. We find an opportunity to relax the slack margins of a processor pipeline using clock stretching flip-flops. The extra slack margin is leveraged to under design the critical fan-in logic by reshaping. We also propose a data activity analysis technique which is used to insert data dependency into the clock stretching flip-flops. Experimental results show power and area savings of 70% and 9.5% respectively in the execute

stage compared to the worst-case design when we use our data-dependent clock stretching technique. This design-time error tolerance is non-speculative which eliminates critical wall of slack behavior.

Table 20. Power savings of the proposed optimization.

Power (μW)	Baseline				TIMBER [31]				CSFF				CSFF + DDCSFF			
	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total	#cells	leakage	dynamic	total
Execute stage	16414	14.8	1307.9	1322.7	16356	14.7	1777.4	1792.2	15053	11.2	471.7	482.9	15053	10.72	438.7	449.42

Table 21. Area savings of the proposed optimization.

Area (μm^2)	Baseline				TIMBER [31]				CSFF				CSFF + DDCSFF			
	#cells	cell area	net area	total	#cells	cell area	net area	total	#cells	cell area	net area	total	#cells	cell area	net area	total
Execute	16414	44741	62899	107639	16356	46919	62784	109702	15053	37612	58360	95971	15053	39079.4	58360	97439.4

Figure 69. Proposed power optimization technique: (a) Power savings, (b) Area savings with respect to worst-case design.

CHAPTER 8. SUMMARY

8.1 Summary of Work-done

Table 23. Power and Area comparisons of the proposed optimizations.

<i>Variables</i>	<i>Timber FF[31],SBFF[44]</i>	<i>SBFF +reshape[46]</i>	<i>SBFF+ sigma[47]</i>
<i>Power in exe module</i>	+29%	-33%	-39%
<i>Area in exe module</i>	+15%	-0.5%	-2.5%

Figure 94 shows the various power reduction techniques which were used to design a power efficient variation tolerant pipeline. An error detection and masking circuit called slack balancing flip-flop is proposed which is used to build the processor pipeline stage from the ground up. The design technique is based on slack analysis and time borrowing. The technique improves the safety margins of the critical paths in a fast and efficient way by using SBFFs and the available slack in the subsequent stages. The slack distribution before and after using the proposed technique show very good slack improvement on the majority of the critical paths which are vulnerable to process, voltage and temperature variations. The opportunity for voltage scaling and frequency improvement using the proposed technique is also studied which helps to reduce the cost of resilience. There is no throughput degradation because we use available slack and mask timing errors instead of instruction replay for error correction. Architectural overhead is minimal compared to other time borrowing concepts because we do not need error propagation logic. The area overhead and power overhead is very minimal and can be traded off by combining the proposed method with other voltage scaling and other optimization strategies. On the circuit level, we reduced the area and power impact of SBFFs

Table 22. Power and Area comparisons of the proposed optimizations.

<i>Variable</i>	<i>Timber FF[31] ,SBFF [44]</i>	<i>SBFF +reshape [46]</i>	<i>CSFF+ reshape [45]</i>	<i>CSFF +sigma [48]</i>	<i>MBCSFF + reshape+ sigma[48]</i>	<i>DDCSFF + reshape [49]</i>
<i>Power in exe stage</i>	+36%	-12%	-63%	-37%	-69%	-70%
<i>Area in exe stage</i>	+2%	-8.4%	-11%	-3%	-15%	-9.5%

by replacing it with a simple clock stretching CSFF. Again, we combine multiple CSFFs together and to further improve the design. The path activation probabilities based on benchmark programs helped to design a new data-dependent clock stretching flip-flop DDCSFF which gave additional power saving on the CSFF. On the algorithmic level, the proposed flip-flops were combined with better than worst-case design techniques to obtain better area, power and performance with added resilience. We propose logic reshaping and BTWC sigma corner library which uses a BTWC voltage and process corner.

Table 24 shows the power and area comparisons between the proposed optimization techniques. The simple TIMBER FF or our version EAFF gave a power and area overhead of 36% and 2% respectively in the execute stage. Whereas, once we employ additional optimization techniques we get significant power and area savings. SBFF and logic reshaping gave a power and area savings of 12% and 8.4% respectively. When we remove redundancy overheads of SBFF using a CSFF, we get power and area savings of 63% and 11% respectively. So, we get additional power savings of 51% and area saving of 9% due to CSFF. CSFF together with sigma optimization gave power and area saving of 37% and 3% respectively. This is lower than logic reshaping because sigma optimization is only applied to a small module in the execute stage and so the saving is less. The combination of MBCSFF, sigma and size optimization gave a power and area saving of 69% and 15% respectively. This is the best area saving we get out of all the techniques. Finally, DDCSFF gave a power saving and area saving of 70% and 9.5% respectively. Here we have additional 1% power saving compared to MBCSFF because we take data activity rate to gate the clock signal. But the area saving is not as good as MBCSFF because we need additional clock gating circuitry. Table 25 shows the power and area comparison of the proposed sigma and size optimization on the SBFF against the baseline worst-case design. Here we compare the parameters within the execute stage module where sigma optimization is applied and find that TIMBER and SBFF give 29% and 15% power and area overhead respectively. Whereas, the size-based optimization gave power and area saving of 33% and 0.5% respectively. Sigma optimization give a power and area saving of 39% and 2.5% respectively. This shows that sigma optimization is superior to size-based optimization within the module.

Table 26 shows the comparison summary of the proposed techniques with the existing schemes. Compared to all the existing speculative techniques, the proposed methods are non-speculative and deterministic in nature. We build the processor pipeline from ground up utilizing existing opportunities in the design. This helps to avoid all the error handling costs in terms of power, area and latency overheads. There is no need to combine all the error signals and there is no need for redundancy on the flip-flop level. This reduces clock tree loading, sequential and combinational overheads associated with the existing speculation mechanisms. The timing closure at design time helps to avoid meta-stability and short path hold buffer issues.

Table 24. Comparison summary of the proposed optimizations with state of art.

<i>Feature</i>	<i>Design level optimization</i>	<i>Sequential Optimization based on error detection</i>	<i>Combinational & Sequential Optimization</i>	<i>Sequential Optimization based on error masking</i>	<i>Proposed</i>
<i>Speculation Mechanism</i>	ABB, ASV, OSB, PCT	Adaptive voltage/frequency scaling	Adaptive voltage/frequency scaling	Adaptive voltage/frequency scaling	Non-speculative, based on cell downsizing and BTWC library
<i>Error Handling</i>	Duplicate paths	Duplicate Latch/FFs	Duplicate Latch/FFs	Duplicate Latch/FFs	No error
<i>Clock Tree Loading</i>	No	Yes	Yes	Yes	No
<i>Short Path Padding</i>	No	Yes	Yes	Yes	No
<i>Sequential Overhead</i>	Large	Large	Large	Large	None
<i>Combinational Overhead</i>	Small	Small	Small	Small	None
<i>Meta-stability</i>	Yes	Yes	Yes	Yes	No
<i>Techniques</i>	EVAL[33], Blueshift[34]	Razor I[18], Razor II[19], Bubble Razor[21] DSTB,DTB[20]	Slack Optimizer[36], Skew Optimizer[37], CombOpt[38]	TIMBER[31], soft edge flip-flop[30]	SBFF[44], CSFF[45], MBCSFF[48], DDCSFF[49]

8.2 Future Work

Figure 71. Slack histogram vs. voltage scaling.

With the aim to maximize yield, dynamic variation tolerance the resilient design proposed can be improved using opportunistic methods to optimize performance, power, and area on the circuit as well as architectural level. We plan to combine the design-time resilience techniques with runtime voltage/frequency scaling to reap more benefits. As shown in Figure 95, the slack of the critical and near critical paths are affected by voltage scaling. When the voltage is scaled by 10mV, there is a sharp rise in the number of paths with a negative slack. Existing speculative approaches would need to take care of the new errors. But as shown in Figure 96, the consecutive stages have considerable slack even when the voltage is scaled by 10mV. This is an opportunity which can be used in our proposed techniques. The design time savings that we achieve from our optimizations can be improved through voltage scaling.

Figure 72. Consecutive stage slack for voltage scaling.

Figure 73. Frequency scaling opportunity.

Figure 97 shows frequency scaling opportunity with the proposed optimization techniques. As shown in the figure, the critical stage in a conventional execute pipeline has a slack of 0ns. The critical stage determines the clock frequency which limits the maximum operating frequency of the chip to 128MHz in this case. Whereas in an opportunistic SBFF pipeline, the critical stages are replaced by SBFF and so the slack is relaxed. Thus, a critical operation can be finished in 7ns time which results in a frequency improvement of 11%. It depends on the application, whether we want to trade-off the extra slack for power/area gains or for frequency enhancement.

Also, we can train the circuit to enable or disable time borrowing for the individual flip-flops depending on the data activity rates. This will help to reduce the redundancy overheads associated with the time borrowing hardware. Apart from processor pipelines, the proposed slack balancing and power optimizations can be done on any digital circuit, which makes it flexible and scalable.

CHAPTER 9. PUBLICATIONS

- [44] M. Jayakrishnan, J. P. D. Gyvez, A. Chang, and T. Kim, "Slack-aware Timing Margin Redistribution Technique Utilizing Error Avoidance Flip-flops and Time Borrowing," IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC), pp. 159-164, Oct. 2015.
- [45] M. Jayakrishnan, A. Chang, and T. Kim, " Power and Area Efficient Clock Stretching and Critical Path Reshaping for Error Resilience," IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC), Sept. 2016.
- [46] Jayakrishnan, M.; Chang, A.; Kim, T.T.-H. Opportunistic Design Margining for Area and Power Efficient Processor Pipelines in Real Time Applications. J. Low Power Electron. Appl. 2018, 8, 9.
- [47] M. Jayakrishnan, A. Chang, and T. Kim, " Library Pruning and Sigma Corner Libraries for Power Efficient Variation Tolerant Processor Pipelines," IFIP/IEEE International Conference on Very Large-Scale Integration (VLSI-SoC), Oct. 2017.
- [48] M. Jayakrishnan, A. Chang, and T. Kim, "Power and Area Efficient Clock Stretching and Critical Path Reshaping for Error Resilience," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [Submitted].
- [49] M. Jayakrishnan, A. Chang, and T. Kim, "Data-dependent Clock Stretching for Power Efficient Variation Tolerant Processor Pipelines," [To be submitted].

CHAPTER 10. BIBLIOGRAPHY

- [1] S. Borkar, T. Karnik, S. Narendra, J. Tschanz, A. Keshavarzi, and Vivek De, "Parameter Variations and Impact on Circuits and Microarchitecture," Proceedings of DAC, Anaheim, CA, USA, June 2003, pp. 338-342.
- [2] S. Ghosh and K. Roy, "Parameter variation tolerance and error resiliency: New design paradigm for the nanoscale era," Proc. IEEE, vol.98, no. 10, pp. 1718–1751, Oct. 2010.
- [3] A. Asenov, A. R. Brown, J. H. Davies, S. Kaya, and G. Slavcheva, "Simulation of intrinsic parameter fluctuations in decananometer and nanometer-scale MOSFETs," IEEE Trans. Electron Devices, vol. 50, no. 9, pp. 1837–1852, Sep. 2003.
- [4] S. R. Nassif, "Modeling and analysis of manufacturing variations," in Proc. Custom Integrated Circuit Conf., 2001, pp. 223–228.
- [5] S. Borkar, "Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation," IEEE Micro, Vol. 25, No. 6, pp. 10-16, November-December 2005.
- [6] K. A. Bowman, S. G. Duvall, and J. D. Meindl, "Impact of Die-to-Die and Within-Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration," IEEE Journal of Solid State Circuits, Vol. 37, February 2002, pp. 183–190.
- [7] A. Tiwari et al., "ReCycle: Pipeline adaptation to tolerate process variation," in Proc. Intl. Symposium on Computer Architecture, pp. 323–334, 2007
- [8] M. Wiecekowsi et al., "Timing yield enhancement through soft edge flip-flop based design," in Proc. Custom Integrated Circuits Conference, pp. 543–546, 2008
- [9] L. X. Liang et al., "Revival: A variation-tolerant architecture using voltage interpolation and variable latency," Proc. Intl. Symposium on Microarchitecture, vol. 29, pp. 127–138, 2009
- [10] Ronald G. Dreslinski, Michael Wiecekowsi, David Blaauw, Dennis Sylvester, and Trevor Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," Proc. IEEE, Vol. 98, No. 2, Feb. 2010.
- [11] K. Bowman et al., "Circuit techniques for dynamic variation tolerance," in Proc. Design Automation Conference, pp. 4–7, 2009
- [12] M. Meijer, B. Liu, R. van Veen and J. Pineda de Gyvez, "Post-Silicon Tuning Capabilities of 45nm Low- Power CMOS Digital Circuits," Proceedings of Symposium on VLSI Circuits, Kyoto, Japan, June 2009, pp.110-111.
- [13] T. Kim, R. Persaud, and C. H. Kim, "Silicon odometer: An on-chip reliability monitor for measuring frequency degradation of digital circuits," in Proc. Very Large Scale Integr. (VLSI) Circuits Symp., 2007, pp. 122–123.
- [14] Puneet Gupta et al., "Underdesigned and Opportunistic Computing in Presence of Hardware Variability," IEEE Transactions on Computer-Aided Design of integrated circuits and systems, vol. 32, no. 1, January 2013.
- [15] T.Sato and Y. Kunitake "A simple flip-flop circuit for typical-case designs for DFM," Proc. Of ISQED, San Jose, CA, USA, March 2007, pp. 539-544.
- [16] A.K. Uht, "Going Beyond Worst-Case Specs with TEAtime," Computer, Volume 37, March 2004, pp. 51-56

- [17] J. Tschanz et.al. "Tunable Replica Circuits and Adaptive Voltage-Frequency Techniques for Dynamic Voltage, Temperature, and Aging Variation Tolerance," Proc. of Symposium on VLSI Circuits, Kyoto, Japan, June 2009, pp. 112-113.
- [18] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, T. Mudge, and K. Flautner. Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation. Proceedings of the 36th Symposium on Microarchitecture (MICRO-36), San Diego, CA, 2003.
- [19] S. Das, C. Tokunaga, S. Pant, W. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance," IEEE J. Solid-State Circuits, vol. 44, no. 1, pp. 32-48, 2009.
- [20] K. Bowman, J. Tschanz, N. Kim, J. Lee, C. Wilkerson, S. Lu, T. Karnik, and V. De, "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance," IEEE J. Solid-State Circuits, vol. 44, no. 1, pp. 49-63, 2009.
- [21] M. Fojtik et al., "Bubble Razor: Eliminating timing margins in an ARM cortex-M3 processor in 45 nm CMOS using architecturally independent error detection and correction," IEEE J. Solid-State Circuits, vol. 48, no. 1, pp. 66-81, Jan. 2013.
- [22] T. Kehl, "Hardware Self-Tuning and Circuit Performance Monitoring," Proc. of ICCD, Cambridge, MA, USA, October 1993, pp. 188-192.
- [23] P. Franco et al., "On-line delay testing of digital circuits," in Proc. VLSI Test Symposium, pp. 167-173, 1994.
- [24] M. Favalli et al., "Sensing circuit for on-line detection of delay faults," IEEE Trans. VLSI Systems, vol. 4, pp. 130-133, 1996.
- [25] C. Metra et al., "On-line detection of logic errors due to crosstalk, delay, and transient faults," in Proc. Intl. Test Conference, pp. 524-533, 1998.
- [26] M. Nicolaidis, "Time redundancy based soft error tolerance to rescue nanometer technologies," in Proc. VLSI Test Symposium, pp. 86-94, 1999.
- [27] Y. Tsiatouhas et al., "A sense amplifier based circuit for concurrent detection of soft and timing errors in CMOS ICs," in Proc. Intl. On-line Testing Symposium, pp. 12-16, 2003.
- [28] D. Bull, S. Das, K. Shivashankar, G. Dasika, K. Flautner, and D. Blaauw, "A power-efficient 32-bit ARM processor using timing-error detection and correction for transient-error tolerance and adaptation to PVT variation," IEEE J. Solid-State Circuits, vol. 46, no. 1, pp. 18-31, Jan. 2011.
- [29] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner, and T. T. Mudge, "A self-tuning DVS processor using delay error detection and correction," IEEE J. Solid-State Circuits, vol. 41, pp. 792-804, 2006.
- [30] V. Joshi. David Blaauw. Deunis Sylvester, "Soft-edge flip-flops for improved timing yield: design and optimization," in Proc. IEEE/FACM Int. Conf. Computer-Aided Design, Nov. 2007, pp. 667-673.
- [31] M. Choudhury et.al. "TIMBER: Time borrowing and error relaying for online timing error resilience," Proc. of DATE, Dresden, Germany, March 2010, pp. 1554-1559.
- [32] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, "Opportunities and Challenges for Better Than Worst-Case Design", Proc. Asia and South Pacific Design Automation Conf., 2005, pp. 2-7

- [33] S. R. Sarangi, B. Greskamp, A. Tiwari, and J. Torrellas. EVAL:Utilizing processors with variation-induced timing errors, in International Symposium on Microarchitecture, November 2008.
- [34] B. Greskamp, L. Wan, W. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen and C.Zilles, “BlueShift: Designing Processors for Timing Speculation from the Ground Up”, IEEE International Symposium on High Performance Computer Architecture, 2009, pp. 213–224.
- [35] J. Patel, “CMOS Process Variations: A Critical Operation Point Hypothesis”, Online Presentation, 2008.
- [36] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, “Slack Redistribution for Graceful Degradation Under Voltage Overscaling”, Proc. ASP-DAC, 2010, pp.825–831.
- [37] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, “Designing a Processor From the Ground Up to Allow Voltage/Reliability Tradeoffs”, IEEE International Symposium on High-Performance Computer Architecture, January 2010.
- [38] A. B. Kahng, S. Kang, J. Li, “A New Methodology for Reduced Cost of Resilience”, Proc. GLSVLSI, 2014, pp. 157-162.
- [39] X. Liu, M. C. Papaefthymiou, and E. G. Friedman, “Maximizing performance by retiming and clock skew scheduling,” in 36th Design Automation Conf., June 1999, pp. 231–236.
- [40] José Monteiro, Srinivas Devadas, Abhijit Ghosh, Retiming sequential circuits for low power, Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design, p.398-402, November 07-11, 1993, Santa Clara, California, USA.
- [41] Neves, José Luis, and Eby G. Friedman. "Optimal clock skew scheduling tolerant to process variations." Proceedings of the 33rd annual Design Automation Conference. ACM, 1996.
- [42] Sathyamurthy H, Sapatnekar S, Fishburn J (1998) Speeding up pipelined circuits through a combination of gate sizing and clock skew optimization. IEEE Trans Computer Aided Design of Integrated Circuits Systems 17(2):173–182.
- [43] Xi, Joe G., and Wayne W-M. Dai. "Useful-skew clock routing with gate sizing for low power design." Proceedings of the 33rd annual Design Automation Conference. ACM, 1996.