11-19-2015

Synthesis Techniques for Sub-threshold Leakage and NBTI Optimization in Digital VLSI Systems

Shilpa Pendyala
University of South Florida, shilpapendyala@gmail.com

Follow this and additional works at: http://scholarcommons.usf.edu/etd
Part of the Computer Engineering Commons

Scholar Commons Citation
http://scholarcommons.usf.edu/etd/6012

This Dissertation is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Synthesis Techniques for Sub-threshold Leakage and NBTI Optimization
in Digital VLSI Systems

by

Shilpa Pendyala

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

Major Professor: Srinivas Katkoori, Ph.D.
Nagarajan Ranganathan, Ph.D.
Hao Zheng, Ph.D.
Bibiche Geuskens, Ph.D.
Sanjukta Bhanja, Ph.D.
Gregory McColm, Ph.D.

Date of Approval:
November 5, 2015

Keywords: Low Power, IC Reliability, Self Similarity, Simulated Annealing, Interval Propagation

Copyright © 2015, Shilpa Pendyala
DEDICATION

I want to dedicate this PhD work to my mom and dad who lent me constant support throughout my journey.
ACKNOWLEDGEMENTS

I would like to thank Dr. Srinivas Katkoori for his continuous support, guidance, and invaluable suggestions throughout the whole PhD. I am very grateful for him explaining to me the simplest of things whenever I hit a road block.

I want to thank the committee members, Dr. Ranga, Dr. Zheng, Bibiche, Dr. Bhanja, and Dr. McColm for serving on my doctoral committee and providing their valuable feedback.

I would forever be grateful to Dr. Ranga for inspiring me with his tech talks during VLSI Algorithms and Architecture classes. It helped me look at future of chip design in a new light.

My sincere thanks to the Computer Science and Engineering department for providing me with financial support. I would also like to thank Theresa Collins, Kim Bluemer, Yvette Blanchard, Shanie Lightbourne, and Monique Mavour from Computer Science and Engineering department for their immense help throughout my stay at USF.

My graduate internship with Intel Corporation, Hillsboro, OR, provided me with hands on industry experience. I would like to thank Bibiche my manager at Intel, for giving me this opportunity and insight into circuit design domain.

Technical assistance from CSE tech support (Danny Prieto, Joseph Botto) and USF Research Computing (Tony Green, Brian Smith) is commendable without whom my research experience would not have been as easy. They provided timely help with industry tools when unexpected issues arose.
Last but not the least, I would like to thank my parents, grandparents, my sister (Swetha Pendyala), my aunt (Viplava Nekkalapudi), and my uncle (Rama Krishna Nekkalapudi) for their constant motivation and support. I am grateful for my friends in Tampa, Aswani Raheja, Sri Harsha Garapati, Akbar Shareef, Jeevan Gogineni, and many others for making my life cheerful and fun throughout my stay in Tampa. I would also like to thank my friends Sindhura Ketineni, Swathi Priya Pedavalli, Divya Paruchuri, and Aswani Raheja for keeping me sane through tough times.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................... iv
LIST OF FIGURES .......................................................................... vi
ABSTRACT .................................................................................... xiv

CHAPTER 1 : INTRODUCTION AND MOTIVATION .............................. 1
  1.1 Novelty and Contributions ....................................................... 3
  1.2 Research Problems and Proposed Work .......................................... 3
    1.2.1 Problem 1 - RTL Subthreshold Leakage Optimization in Datapaths . 3
    1.2.2 Problem 2 - NBTI Leakage Co-optimization at Gate Level .......... 6
    1.2.3 Problem 3 - NBTI Optimization in FSMs ............................... 8
  1.3 Dissertation Organization ........................................................ 9

CHAPTER 2 : BACKGROUND AND RELATED WORK ............................ 10
  2.1 Levels of Abstraction ............................................................ 10
  2.2 Leakage Mechanisms in CMOS Circuits .......................................... 11
    2.2.1 Subthreshold Leakage .................................................. 11
    2.2.2 Gate Leakage .......................................................... 12
    2.2.3 Junction Leakage ...................................................... 12
  2.3 Subthreshold Leakage Minimization .............................................. 12
    2.3.1 Standby Techniques .................................................... 12
    2.3.2 Active Techniques ...................................................... 16
  2.4 Input Vector Control Technique .................................................. 17
  2.5 Fractals and Self Similarity ...................................................... 18
  2.6 Interval Arithmetic .............................................................. 20
  2.7 NBTI Background ............................................................... 21
    2.7.1 NBTI Phenomenon..................................................... 21
    2.7.2 Device-level BTI Models................................................ 22
  2.8 Factors Affecting BTI ............................................................ 24
    2.8.1 Temperature............................................................ 24
    2.8.2 Active to Standby Ratio .............................................. 24
    2.8.3 Supply Voltage ........................................................ 24
    2.8.4 Process Variations ...................................................... 24
    2.8.5 CMOS Technology Node .............................................. 25
    2.8.6 Circuit Topology ....................................................... 25
  2.9 BTI Analysis and Optimization .................................................. 25
    2.9.1 Layout and Gate-level BTI Analysis and Optimization ............... 25
    2.9.2 Behavioral/RT Level BTI Analysis and Optimization ................. 27
    2.9.3 Micro-architectural Level BTI Analysis and Optimization .......... 27
  2.10 Finite State Machines ........................................................... 28
  2.11 State Encoding in Finite State Machines (FSMs) ......................... 28
LIST OF TABLES

Table 2.1 INV Leakage 0.18 µm Technology ................................................. 13
Table 2.2 NAND2 Leakage 0.18 µm Technology ............................................. 13
Table 2.3 NAND3 Leakage 0.18 µm Technology ............................................. 14
Table 2.4 Comparison of Standby and Active Leakage Minimization Techniques ......... 17
Table 2.5 NBTI Degradation Model ........................................................ 23
Table 2.6 NBTI Degradation Model - Parameters ........................................... 23
Table 2.7 NBTI Degradation Model with Random Input Sequence .......................... 23
Table 3.1 Time Estimation for Different Bit Widths(in Hours) ............................... 44
Table 3.2 Comparison of Monte Carlo Runs ................................................. 46
Table 3.3 Power Savings, Control Points and Area Overhead without Resource Sharing ..... 58
Table 3.4 Power Savings, Control Points and Area Overhead with Resource Sharing ........ 58
Table 3.5 Experiment 1 - MLV at PIs ....................................................... 59
Table 3.6 Experiment 2 - MLV at all DFG Nodes............................................ 60
Table 3.7 Percentage Power Savings in Top Down Propagation with Simulated Annealing .... 63
Table 3.8 Percentage Power Savings in Bottom Up Propagation with Simulated Annealing . . . 63
Table 3.9 Speed Up with Interval Propagation .............................................. 63
Table 3.10 Leakage Comparison of Pure SA with Interval Propagation + SA ............... 63
Table 4.1 Leakage and Delay Range with Simulated Annealing.............................. 76
Table 4.2 MDV Leakage and MLV Delay ...................................................... 76


**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Subthreshold Leakage Power Trend</td>
<td>1</td>
</tr>
<tr>
<td>2.1</td>
<td>MTCMOS</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Power Gating</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>Original MCNC Benchmark Circuit C17</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>New Circuit C17 with Three Gates Replaced</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Romanesco Broccoli (public domain image)</td>
<td>19</td>
</tr>
<tr>
<td>2.6</td>
<td>H Tree (public domain image)</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>Illustration of a Device under Stress due to Bias Temperature Instability (BTI)</td>
<td>22</td>
</tr>
<tr>
<td>2.8</td>
<td>Threshold Voltage ($V_t$) Degradation and Recovery Model</td>
<td>22</td>
</tr>
<tr>
<td>2.9</td>
<td>Simulated Annealing Algorithm</td>
<td>32</td>
</tr>
<tr>
<td>3.1</td>
<td>Leakage Current Distribution for 8b Adder</td>
<td>36</td>
</tr>
<tr>
<td>3.2</td>
<td>Leakage Current Distribution for 8b Multiplier</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Leakage Current Distribution for 8b Adder from 1000 Random Vectors</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>Leakage Current Distribution for 8b Multiplier from 1000 Random Vectors</td>
<td>39</td>
</tr>
<tr>
<td>3.5</td>
<td>Leakage Current Distribution for 16b Adder from 1000 Random Vectors</td>
<td>40</td>
</tr>
<tr>
<td>3.6</td>
<td>Leakage Current Distribution for 16b Multiplier from 1000 Random Vectors</td>
<td>41</td>
</tr>
<tr>
<td>3.7</td>
<td>Hurst Parameter vs Block Size Plot</td>
<td>42</td>
</tr>
<tr>
<td>3.8</td>
<td>Leakage Profile for 8b Adder</td>
<td>42</td>
</tr>
<tr>
<td>3.9</td>
<td>Leakage Profile for 8b Multiplier</td>
<td>43</td>
</tr>
</tbody>
</table>
Figure 4.11  Simulated Annealing for 50% Leakage and 50% NBTI Delay - c432 ................. 80
Figure 4.12  Simulated Annealing for 10% Leakage and 90% NBTI Delay - c432 ................. 81
Figure 4.13  Simulated Annealing for 20% Leakage and 80% NBTI Delay - c432 ................. 81
Figure 4.14  Simulated Annealing for 30% Leakage and 70% NBTI Delay - c432 ................. 81
Figure 4.15  Simulated Annealing for 40% Leakage and 60% NBTI Delay - c432 ................. 81
Figure 4.16  Simulated Annealing for 60% Leakage and 40% NBTI Delay - c432 ................. 82
Figure 4.17  Simulated Annealing for 70% Leakage and 30% NBTI Delay - c432 ................. 82
Figure 4.18  Simulated Annealing for 80% Leakage and 20% NBTI Delay - c432 ................. 82
Figure 4.19  Simulated Annealing for 90% Leakage and 10% NBTI Delay - c432 ................. 82
Figure 4.20  Simulated Annealing for Leakage - c499 ........................................ 83
Figure 4.21  Simulated Annealing for NBTI Delay - c499 ........................................ 83
Figure 4.22  Simulated Annealing for 50% Leakage and 50% NBTI Delay - c499 ................. 83
Figure 4.23  Simulated Annealing for 10% Leakage and 90% NBTI Delay - c499 ................. 83
Figure 4.24  Simulated Annealing for 20% Leakage and 80% NBTI Delay - c499 ................. 84
Figure 4.25  Simulated Annealing for 30% Leakage and 70% NBTI Delay - c499 ................. 84
Figure 4.26  Simulated Annealing for 40% Leakage and 60% NBTI Delay - c499 ................. 84
Figure 4.27  Simulated Annealing for 60% Leakage and 40% NBTI Delay - c499 ................. 85
Figure 4.28  Simulated Annealing for 70% Leakage and 30% NBTI Delay - c499 ................. 85
Figure 4.29  Simulated Annealing for 80% Leakage and 20% NBTI Delay - c499 ................. 85
Figure 4.30  Simulated Annealing for 90% Leakage and 10% NBTI Delay - c499 ................. 85
Figure 4.31  Simulated Annealing for Leakage - c880 ........................................ 86
Figure 4.32  Simulated Annealing for NBTI Delay - c880 ........................................ 86
Figure 4.33  Simulated Annealing for 50% Leakage and 50% NBTI Delay - c880 ................. 86
Figure 4.34  Simulated Annealing for 10% Leakage and 90% NBTI Delay - c880 ............... 87
Figure 4.35  Simulated Annealing for 20% Leakage and 80% NBTI Delay - c880 ............... 87
Figure 4.36  Simulated Annealing for 30% Leakage and 70% NBTI Delay - c880 ............... 87
Figure 4.37  Simulated Annealing for 40% Leakage and 60% NBTI Delay - c880 ............... 87
Figure 4.38  Simulated Annealing for 60% Leakage and 40% NBTI Delay - c880 ............... 88
Figure 4.39  Simulated Annealing for 70% Leakage and 30% NBTI Delay - c880 ............... 88
Figure 4.40  Simulated Annealing for 80% Leakage and 20% NBTI Delay - c880 ............... 88
Figure 4.41  Simulated Annealing for 90% Leakage and 10% NBTI Delay - c880 ............... 88
Figure 4.42  Simulated Annealing for Leakage - c1908 ........................................... 89
Figure 4.43  Simulated Annealing for NBTI Delay - c1908 ........................................ 89
Figure 4.44  Simulated Annealing for 50% Leakage and 50% NBTI Delay - c1908 ............... 89
Figure 4.45  Simulated Annealing for 10% Leakage and 90% NBTI Delay - c1908 ............... 90
Figure 4.46  Simulated Annealing for 20% Leakage and 80% NBTI Delay - c1908 ............... 90
Figure 4.47  Simulated Annealing for 30% Leakage and 70% NBTI Delay - c1908 ............... 90
Figure 4.48  Simulated Annealing for 40% Leakage and 60% NBTI Delay - c1908 ............... 90
Figure 4.49  Simulated Annealing for 60% Leakage and 40% NBTI Delay - c1908 ............... 91
Figure 4.50  Simulated Annealing for 70% Leakage and 30% NBTI Delay - c1908 ............... 91
Figure 4.51  Simulated Annealing for 80% Leakage and 20% NBTI Delay - c1908 ............... 91
Figure 4.52  Simulated Annealing for 90% Leakage and 10% NBTI Delay - c1908 ............... 91
Figure 4.53  Simulated Annealing for Leakage - c2670 ........................................... 92
Figure 4.54  Simulated Annealing for NBTI Delay - c2670 ........................................ 92
Figure 4.55  Simulated Annealing for 50% Leakage and 50% NBTI Delay - c2670 ............... 92
Figure 4.56  Simulated Annealing for 10% Leakage and 90% NBTI Delay - c2670 ............... 93
Figure 4.57  Simulated Annealing for 20% Leakage and 80% NBTI Delay - c2670 ............ 93
Figure 4.58  Simulated Annealing for 30% Leakage and 70% NBTI Delay - c2670 ............ 93
Figure 4.59  Simulated Annealing for 40% Leakage and 60% NBTI Delay - c2670 ............ 93
Figure 4.60  Simulated Annealing for 60% Leakage and 40% NBTI Delay - c2670 ............ 94
Figure 4.61  Simulated Annealing for 70% Leakage and 30% NBTI Delay - c2670 ............ 94
Figure 4.62  Simulated Annealing for 80% Leakage and 20% NBTI Delay - c2670 ............ 94
Figure 4.63  Simulated Annealing for 90% Leakage and 10% NBTI Delay - c2670 ............ 94
Figure 4.64  NBTI Delay and Leakage with Varying Weights - c432 .......................... 95
Figure 4.65  NBTI Delay and Leakage with Varying Weights - c499 .......................... 95
Figure 4.66  NBTI Delay and Leakage with Varying Weights - c880 .......................... 96
Figure 4.67  NBTI Delay and Leakage with Varying Weights - c1908 ........................ 96
Figure 4.68  NBTI Delay and Leakage with Varying Weights - c2670 ......................... 97
Figure 5.1  Sample FSM with $S_0 = 0$, $S_1 = 1$................................................100
Figure 5.2  Sample FSM with $S_0 = 1$, $S_1 = 0$................................................100
Figure 5.3  Framework for NBTI Optimization............................................... 102
Figure 5.4  Simulated Annealing Algorithm for NBTI Optimization ....................... 103
Figure 5.5  SA Iteration Window.............................................................106
Figure 5.6  Generic Simulated Annealing for NBTI Delay - s8 ............................. 109
Figure 5.7  Generic Simulated Annealing for NBTI Delay - ex6 ............................. 109
Figure 5.8  Generic Simulated Annealing for NBTI Delay - dk17 ............................ 109
Figure 5.9  Generic Simulated Annealing for NBTI Delay - s386 ............................ 110
Figure 5.10 Generic Simulated Annealing for NBTI Delay - s1 ............................... 110
Figure 5.11 Generic Simulated Annealing for NBTI Delay - s1494............................110

x
Figure A.37 Multiplier Distribution in Grid Cell 16 (4 x 4 Grid) ......................... 144
Figure A.38 Hurst Parameter in Adder with 16 Cells (4 x 4 Grid) ....................... 144
Figure A.39 Adder Distribution in Grid Cell 1 (4 x 4 Grid) ............................... 145
Figure A.40 Adder Distribution in Grid Cell 2 (4 x 4 Grid) ............................... 145
Figure A.41 Adder Distribution in Grid Cell 3 (4 x 4 Grid) ............................... 146
Figure A.42 Adder Distribution in Grid Cell 4 (4 x 4 Grid) ............................... 146
Figure A.43 Adder Distribution in Grid Cell 5 (4 x 4 Grid) ............................... 147
Figure A.44 Adder Distribution in Grid Cell 6 (4 x 4 Grid) ............................... 147
Figure A.45 Adder Distribution in Grid Cell 7 (4 x 4 Grid) ............................... 148
Figure A.46 Adder Distribution in Grid Cell 8 (4 x 4 Grid) ............................... 148
Figure A.47 Adder Distribution in Grid Cell 9 (4 x 4 Grid) ............................... 149
Figure A.48 Adder Distribution in Grid Cell 10 (4 x 4 Grid) ............................ 149
Figure A.49 Adder Distribution in Grid Cell 11 (4 x 4 Grid) ............................ 150
Figure A.50 Adder Distribution in Grid Cell 12 (4 x 4 Grid) ............................ 150
Figure A.51 Adder Distribution in Grid Cell 13 (4 x 4 Grid) ............................ 151
Figure A.52 Adder Distribution in Grid Cell 14 (4 x 4 Grid) ............................ 151
Figure A.53 Adder Distribution in Grid Cell 15 (4 x 4 Grid) ............................ 152
Figure A.54 Adder Distribution in Grid Cell 16 (4 x 4 Grid) ............................ 153
ABSTRACT

The rising power demands and cost motivates us to explore low power solutions in electronics. In nanometer Complementary Metal Oxide Semiconductor (CMOS) processes with low threshold voltages and thin gate oxides, subthreshold leakage power dominates total power of a circuit. As technology scales, Negative Bias Temperature Instability (NBTI) emerged as a major limiting reliability mechanism. It causes a threshold voltage shift which, over time, results in circuit performance degradation. Hence, leakage power and NBTI degradation are two key challenges in deep sub micron regime.

In this dissertation, interval arithmetic based interval propagation technique is introduced as an effective leakage optimization technique in high level circuits with little overhead. The concept of self similarity from fractal theory is adopted for the first time in VLSI research to handle large design space. Though there are some leakage and NBTI co-optimization techniques in literature, our vector cycling approach combined with a back tracking algorithm have achieved better results for ISCAS85 benchmarks. We did not find any previous research works on NBTI optimization of finite state machines (FSMs). The optimization techniques of NBTI optimization in FSMs is introduced in this dissertation as well and substantial NBTI optimization is reported.

Input vector control has been shown to be an effective technique to minimize subthreshold leakage. Applying appropriate minimum leakage vector (MLV) to each register transfer level (RTL) module instance results in a low leakage state with significant area overhead. For each module, via
Monte Carlo simulation, we identify a set of MLV intervals such that maximum leakage is within (say) 10% of the lowest leakage points. As the module bit width increases, exhaustive simulation to find the low leakage vector is not feasible. Further, we need to search the entire input space uniformly to obtain as many low leakage intervals as possible. Based on empirical observations, we observed self similarity in the leakage distribution of adder/multiplier modules when input space is partitioned into smaller cells. This property enables uniform search of low leakage vectors in the entire input space. Also, the time taken for characterization increases linearly with the module size. Hence, this technique is scalable to higher bit width modules with acceptable characterization time. We can reduce area overhead (in some cases to 0) by choosing Primary Input (PI) MLVs such that resultant inputs to internal nodes are also MLVs. Otherwise, control points can be inserted. Based on interval arithmetic, given a DFG, we propose a heuristic with several variations for PI MLV identification with minimal control points. Experimental results for DSP filters simulated in 16nm technology demonstrated leakage savings of 93.8% with no area overhead, compared to existing work.

Input vector control can also be adopted to reduce NBTI degradation as well as leakage in CMOS circuits. In the prior work, it is shown that minimum leakage vector of a circuit is not necessarily NBTI friendly. In order to achieve NBTI and leakage co-optimization, we propose an input vector cycling technique which applies different sub-optimal low leakage vectors to primary inputs at regular intervals. A co-optimal input vector for a given circuit is obtained by using simulated annealing (SA) technique. For a given input vector, a set of critical path PMOS transistors are under stress. A second input vector is obtained using a back tracking algorithm such that most of the critical path PMOS transistors are put in recovery mode. When a co-optimized input vector
is assigned to primary input, critical path nodes under stress with high delay contribution are set to recovery. Logic 1 is back propagated from the nodes to the primary inputs to obtain the second input vector. These two vectors are alternated at regular time intervals. The total stress is evenly distributed among transistor sets of two vectors, as the intersection of the two sets is minimized. Hence, the overall stress on critical path transistors is alleviated, thereby reducing the NBTI delay degradation. For ISCAS85 benchmarks, an average of 5.3% improvement is achieved in performance degradation at 3.3% leakage overhead with NBTI-leakage co-optimization with a back tracking algorithm compared to solely using co-optimization. A 10.5% average NBTI improvement is obtained when compared to circuit with minimum leakage input vector for 18% average leakage overhead. Also, an average NBTI improvement of 2.13% is obtained with 6.77% leakage improvement when compared to circuit with minimum NBTI vector. Vector cycling is shown to be more effective in mitigating NBTI over input vector control.

Several works in the literature have proposed optimal state encoding techniques for delay, leakage, and dynamic power optimization. In this work, we propose, for the first time, NBTI optimization based on state code optimization. We propose a SA based state code assignment algorithm, resulting in minimization of NBTI degradation in the synthesized circuit. A PMOS transistor when switched ON for a long period of time, will lead to delay degradation due to NBTI. Therefore, in combinational circuits, an NBTI friendly input vector that stresses the least number of PMOS transistors on the critical path can be applied. For sequential circuits, the state code can significantly influence the ON/OFF mode of PMOS transistors in the controller implementation. Therefore, we propose to focus on state encoding. As the problem is computational intractable, we will focus on encoding states with high state probability. The following SA moves are employed: (a)
code swap; and (b) code modification by flipping bits. Experiments with LGSYNTH93 benchmarks resulted in 18.6% improvement in NBTI degradation on average with area and power improvements of 5.5% and 4.6% respectively.
CHAPTER 1 : INTRODUCTION AND MOTIVATION

What started as a two transistor circuit transformed into processors with billions of transistors. In 1958, Jack Kilby built the first Complementary Metal Oxide Semiconductor (CMOS) integrated circuit with two transistors at Texas Instruments [2]. In 2015, Intel’s 18-core Xeon Haswell E5-2600 v3 microprocessor contained more than 5.56 billion transistors [3]. This unparalleled development is made possible due to gradual feature size reduction over the years. The revolution in integrated circuits had a strong impact in the areas such as space travel, automotive industry, etc.
In the early days of IC design, main objectives of VLSI design are power, performance, and cost. As the feature size decreased, circuit complexity increased, which brought up an entirely new set of problems. Emphasis was placed on dynamic component of power. As technology scaled, leakage power turned into a serious problem, and dynamic power minimization took a back seat. In fact, subthreshold leakage is predicted to dominate the overall power with technology scaling [1], as shown in Figure 1.1. Reliability, fault tolerance, and thermal issues emerged as non-trivial issues in chip design. In this dissertation, we address subthreshold leakage and Negative Bias Temperature Instability (NBTI) problems in VLSI design.

Subthreshold leakage optimization gained significance as supply voltage in ICs was reduced. Leakage power is expected to increase exponentially with technology scaling according to ITRS [4]. This impacts battery powered electronic devices such as cell phones, medical devices, etc. The complexity of processors used in mobile devices has increased over the years, and it is necessary to minimize leakage power dissipation in these devices.

Reliability issues such as NBTI, soft errors, electromigration, self heating, and hot carriers should be controlled to avoid circuit failure. Reliability is crucial in some applications of integrated circuits. According to ITRS 2013 [5], automotive, military, and aerospace are some examples where the circuits are subjected to high temperatures. These applications are subjected to greater stress than consumer electronics. Space based applications need to work under severe radiation. Space stations cannot afford circuit failures for tens of years at extreme temperatures. Failure in such a scenario will not only cost a great deal, but also results in severe consequences. Also, medical applications with implantable ICs cannot be neglected. Failure in this case impacts the life of the users. Reliability optimization is, therefore, a serious challenge in the semiconductor industry.
According to ITRS 2013 [5], insufficient reliability margin can lead to field failures that are costly to fix and damaging to reputation. NBTI phenomenon is one of the major reliability problems that slows the circuit over time, negating the performance improvement.

1.1 Novelty and Contributions

1. We address the subthreshold leakage and NBTI degradation problems in DSM regime
2. A leakage optimization technique is developed for RTL circuits
3. Self similarity, a concept from fractal theory, is explored to get a better handle on the large input space during module characterization
4. Leakage and NBTI co-optimization is achieved in gate level benchmarks
5. NBTI minimization problem is also addressed in finite state machines (FSMs)
6. NBTI aware state encoding is proposed to alleviate the degradation

1.2 Research Problems and Proposed Work

This section gives an outline of the three research problems addressed in this dissertation. It also gives a synopsis of the techniques adopted in each problem. Finally, experimental results are summarized.

1.2.1 Problem 1 - RTL Subthreshold Leakage Optimization in Datapaths

Digital CMOS circuits implemented in deep-submicron technology nodes can consume subthreshold leakage power that is 50% or more of the total power consumption [4]. Therefore, subthreshold leakage optimization is an active research area with several techniques reported in the literature [6, 7, 8, 9, 10, 11, 12, 13, 14]. Of these techniques, Input Vector Control (IVC) is attrac-
tive due to its low latency overhead. However, determination of minimum leakage vector for large
circuits is a problem by itself as it requires strenuous data processing and simulation time [11].

Subthreshold leakage is a strong function of current input vector applied to a digital CMOS
circuit [15]. Therefore, under idle conditions, the circuit can be put in the lowest leakage mode by
applying a minimum leakage vector (MLV). Once the circuit becomes active, it can readily process
the new input vectors without any latency overhead. It is well-known that MLV identification
problem is an intractable problem [11]. Therefore, several sub-optimal MLV identification algorithms
[6, 11, 16] have been proposed at the logic-level. In this work, we refer to sub-optimal MLVs as
low leakage vectors (LLVs). Applying appropriate LLV to each RTL module incurs significant area
and control overhead. The additional area is the result of multiplexers at the inputs of the RTL
modules to apply the low leakage vector. Control overhead is incurred to control the select lines for
these multiplexers. We propose top down and bottom up interval propagation techniques.

A straightforward way for input vector control at RTL is to apply appropriate LLV to each
RTL module instance, resulting in an overall low leakage state. However, this requires significant
area and control overhead. The additional area results because of multiplexers at the inputs of the
RTL modules to apply the low leakage vector. Control overhead is incurred to control the select lines for
these multiplexers.

Firstly, we analyze the leakage profiles of RTL modules, and propose a scalable low leak-
age interval characterization of modules for datapath intensive applications. We characterize each
functional unit in the library (Section 3.2) for low leakage data intervals based on a two-phased
Monte-Carlo simulation. A low leakage data interval is a set of inputs such that the given func-
tional unit is in a low leakage state with very high confidence. The characterization procedure
exploits empirically observed self similarity property of the module’s leakage power distributions. The set of low leakage data intervals are such that the maximum leakage across all intervals is within (say) 10% of the lowest leakage value observed. This percentage value is user specified during the characterization procedure.

As the data input size grows exponentially with the module bit-width, characterization by exhaustive simulation is not feasible. In order to make the problem of low leakage data interval detection tractable, we exploit the self-similarity property of leakage distribution of a given functional unit. Briefly, given a stochastic distribution, it is said to be self-similar [17, 18], if any arbitrary sub-distribution (built by choosing contiguous samples) is similar to the original distribution. Hurst parameter is a commonly used metric to determine self-similarity of a distribution. For two module types, adder and multiplier, we have empirically observed that their leakage distribution is self-similar. Given a confidence level ($\alpha$) and an error tolerance ($\beta$), Halter and Najm [19] have derived a formula to determine the minimum number of random vectors needed to find a LLV that is away from the MLV by $\beta\%$ with $\alpha\%$ confidence, provided the leakage distribution is a normal distribution. While the goal of any MLV heuristic is to find one LLV, our goal is to find as many LLVs as possible so that we can expand them into LLV data intervals. We observe that adder and multiplier leakage distributions to be normal and self-similar. Therefore, we can apply the Halter and Najm’s technique to search in a subspace due to gridding. More details are found in Section 3.2. Top down and bottom up interval propagation techniques are implemented over the raw interval set obtained from the characterization. The interval propagation yields a reduced set of low leakage intervals at the primary inputs. A simulated annealing algorithm is devised to find the best input vector for a given circuit out of the reduced interval set.
Following are the contributions of this problem: (a) top down interval propagation of all LLV intervals (unlike just the lowest leakage intervals in prior work); (b) bottom up interval propagation (c) allowed resource sharing; and (d) proposed self similarity based module characterization to make LLV detection more tractable. The average leakage savings of 93% are achieved for top down case with no area overhead. Bottom up propagation, however, could not only achieve 89.2% average leakage savings. In all our experiments, the optimized designs did not require any internal control points, and thus incurred no overhead in terms of area, control, and delay.

1.2.2 Problem 2 - NBTI Leakage Co-optimization at Gate Level

Reliability is a growing problem as technology size advances into deep sub-micron (DSM) technology nodes. Oxide wear out in transistors is caused due to Hot Carrier Injection (HCI), Bias Temperature Instability (BTI), and Time Dependent Dielectric breakdown (TDDB). Out of these, Negative Bias Temperature Instability (NBTI) emerged as a predominant reliability concern in nanometer regime. This phenomenon results in threshold voltage degradation in PMOS transistors thereby causing performance degradation as the circuit ages. According to ITRS 2013 [5], NBTI optimization problem has become increasingly significant as threshold voltages have been scaled down in the recent technologies. The introduction of high-k gates makes it even worse. Input vector control is a well known technique for leakage reduction. This technique can be used to reduce NBTI degradation as well. Khan et al. [20] demonstrated that minimum leakage vector of a circuit might not necessarily be NBTI friendly. This is due to the opposing effect of stacking on the two phenomena. NBTI effect increases when transistors are stacked as opposed to leakage which is reduced with stacking. For example, the minimum leakage vector for a NAND gate is all 0’s while the best NBTI vector is all 1’s. On the other hand, in case of a NOR gate, the minimum leakage
vector as well as the best NBTI vector is all 1's.

It is essential to devise a technique for co-optimizing leakage and NBTI degradation. A straightforward way is to choose a best vector that is optimized for both NBTI and leakage after several random vector simulations. This technique, however, does not yield good results due to vast input space. We implement a simulated annealing (SA) algorithm on the input space of the circuit. SA algorithm can be used to optimize leakage, NBTI degradation, or both. Output of SA algorithm to co-optimize leakage and NBTI is a co-optimal input vector to be applied at the primary inputs during idle time. Any specific input vector puts a specific set of transistors under stress. A second input vector is obtained such that it recovers critical path PMOS transistors of co-optimal input vector. A backtracking algorithm coded in C is used to identify the second input vector. PMOS transistors on critical and near critical paths are identified using Synopsys NanoTime, a spice level static timing analysis tool. As degradation varies with different input vectors, critical path changes as well. It is, therefore, essential to include near critical path transistors as well.

The two vectors are applied to primary inputs during idle time and are switched at regular intervals. NBTI is frond-loaded in nature due to the underlying phenomena [21]. The threshold voltage degradation occurs at a faster rate early in the lifetime and slows down eventually. There is a need to keep NBTI degradation in check at regular intervals rather than focusing on total degradation in a lifetime. Hence, vector cycling at regular intervals is justified with slight dynamic power overhead. The total stress time is evenly distributed among transistor sets of two vectors which are mutually exclusive. Hence, the overall stress time of critical path transistors is drastically reduced thereby alleviating NBTI delay degradation. Experimental results for subset(5) of ISCAS85 benchmarks showed a 5.3% improvement in performance degradation with 3.3% leakage overhead.
compared to leakage and NBTI co-optimization. When compared to the results obtained from using minimum leakage vector (MLV) at primary input, 10.5% NBTI improvement with 18% leakage overhead are obtained. An average NBTI improvement of 2.13% is obtained with 6.77% leakage improvement when compared to a circuit with minimum NBTI vector.

### 1.2.3 Problem 3 - NBTI Optimization in FSMs

The delay contribution of FSMs in the overall circuit delay is significant and cannot be ignored. Hence, NBTI optimization techniques for FSMs are to be researched. In this work, we present a technique along with elaborate framework to optimize NBTI delay degradation in finite state machines. The states of a finite state machine vary by the duration of FSM operation in each state for a given input sequence. If NBTI degradation during high probability states is minimized, overall NBTI degradation can be improved. A simulated annealing (SA) optimization is proposed to achieve the same. This can be done in two ways: (a) Generic SA; and (b) State probability based SA. The delay degradation is considered as cost function. SA moves in each iteration for state probability based SA consists of (a) Swap between high probability states and used states; and (b) Swap between high probability states and unused states. SA moves for generic SA consists of: (a) Swap between used states; (b) Swap used and unused states. Total simulation time can be chosen by the user and it determines the quality of the solution. State probability based technique has more scope of improvement when few states have large probability numbers. This is due to the fact that input space is reduced and focused on high probability states which have more control over NBTI improvement in a lifetime.

Each SA iteration works with a new state code assignment to estimate gains in NBTI optimization. One major drawback of this scheme is that the re-encoding of states disturbs hamming
distance during the state transitions. This in turn can incur dynamic power overhead. However, if considerable NBTI optimization is achieved, slight overhead is acceptable. Once the state codes for all states are established, FSM is synthesized to a gate level circuit using Cadence BuildGates. Gate level synthesis of FSM with new state encoding in each simulated annealing iteration might cause area and leakage overhead as well. Threshold voltage degradation of the transistors is calculated using net probabilities and NBTI long term degradation model [22]. Net probabilities of a circuit for a given input sequence are estimated using Synopsys Primetime tool. An in-house framework derives a NBTI degraded circuit with degraded models for the transistors. A static timing analysis tool is used to measure the delay of the critical path of degraded circuit from which NBTI delay degradation is calculated. The framework obtains NBTI degradation for any given state code assignment of an FSM circuit, which is used as a cost function in simulated annealing. Experiments with a subset (8) of LGSYNTH93 benchmarks resulted in 18.6% NBTI improvement on average with area and power improvements of of 5.5% and 4.6%, respectively.

1.3 Dissertation Organization

The rest of the dissertation is organized as follows: Chapter 2 presents background, related work, and terminology. Chapter 3 proposes the leakage minimization problem. It also introduces a self similarity based Monte Carlo characterization to make the technique scalable. Chapter 4 proposes in detail the gate level NBTI and leakage minimization technique. Chapter 5 presents the NBTI optimization of FSMs. Finally, Chapter 6 draws conclusions and outlines future work. Appendix A provides self similarity plots for 8b adder and 8b multiplier. Appendix B consists of copyright clearance forms of the published work.
CHAPTER 2 : BACKGROUND AND RELATED WORK

We will first give a brief overview of levels of abstraction in CMOS design, leakage mechanisms, and subthreshold leakage minimization techniques. As this work is related to input vector control technique, we will survey IVC techniques reported in the literature. After providing a brief overview of interval arithmetic, we establish the terminology used in this dissertation. We also introduce the concept of fractals and self similarity. We then discuss the related work in NBTI optimization as well as NBTI - leakage co-optimization. An overview on FSMs and simulated annealing algorithms is also presented.

2.1 Levels of Abstraction

VLSI circuit models has three levels of abstraction [23]: architectural, logic and geometrical. At architectural level, we work with a set of high level operations such as addition, multiplication, or data transfer. An architectural model consists of a datapath and a controller which controls the dependencies in datapath. At logic level, the circuit evaluates logic functions and implements them with logic gates. At geometrical level, circuit is represented a set of geometrical shapes. A geometric model consists of a floor plan or layout with transistors and interconnects represented by geometric shapes like rectangles.
2.2 Leakage Mechanisms in CMOS Circuits

Leakage power has three sources: subthreshold, gate, and junction leakage currents. Subthreshold leakage is most dominant leakage source amongst the three.

2.2.1 Subthreshold Leakage

The subthreshold leakage [24] is the current flowing between the source and the drain when the transistor is in weak inversion mode. It arises due to carrier diffusion of nonzero minority carriers. The following parameters affect the subthreshold leakage current:

1. Drain-Induced Barrier Lowering (DIBL):
   The increase in drain voltage of short channel device lowers the threshold voltage. This effect is called drain-induced barrier lowering. In a short channel device, the effect of decreasing channel length brings source and drain diffusion close to each other, which induces significant drain current. Hence, the subthreshold leakage has strong dependence on drain bias.

2. Body Bias:
   The threshold voltage raises with increase in the potential between source and body. This is called body effect. Optimized body bias can be identified to minimize the leakage.

3. Temperature:
   Threshold voltage is dependent on the temperature, which causes temperature dependence of subthreshold leakage current.
2.2.2 Gate Leakage

Carriers tunnel through a thin gate dielectric when a voltage is applied across the gate. This causes gate leakage [25]. It is a strong function of dielectric thickness. It also depends on voltage across the gate. Stacking with OFF transistors closer to the rail can alleviate gate leakage.

2.2.3 Junction Leakage

If source or drain is at a different potential from the substrate, it causes junction leakage [25]. BTBT and GIDL increases the severity of junction leakage current. A strong reverse bias between source and drain effects BTBT. A strong drain bias during device OFF mode impacts the GIDL.

2.3 Subthreshold Leakage Minimization

Subthreshold leakage control can be in standby mode or active mode. Standby techniques include Multi Threshold CMOS (MTCMOS), Power gating, and Super Cutoff CMOS (SCCMOS). Active mode leakage control techniques include input vector control, force stacking, sleepy stack, and power gating with stacking.

2.3.1 Standby Techniques

MTCMOS [7] as shown in Figure 2.1 and power gating [9] as shown in Figure 2.2 involve disconnecting supply voltage and/or ground to the circuit through sleep transistors. These need to be appropriately sized to reduce delay penalty (in active mode) and wake-up time (to restore circuit to active mode). Thus, these techniques incur both area and delay overheads. In power gating, both sleep and logic transistors have low threshold voltage, while in MTCMOS, the sleep transistors
are high $V_t$. In SCCMOS style [26], the sleep transistor is driven into super cutoff mode resulting in an order of magnitude leakage reduction in sleep transistor. These savings are at the expense of complex controller design and large delay penalty.

![MTCMOS Diagram](image)

**Figure 2.1: MTMCMOS**

**Table 2.1: INV Leakage 0.18 µm Technology**

<table>
<thead>
<tr>
<th>Input</th>
<th>Leakage (nA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Best: 100.3</td>
</tr>
<tr>
<td>1</td>
<td>Worst: 227.2</td>
</tr>
</tbody>
</table>

**Table 2.2: NAND2 Leakage 0.18 µm Technology**

<table>
<thead>
<tr>
<th>Input</th>
<th>Leakage (nA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Best: 37.84</td>
</tr>
<tr>
<td>01</td>
<td>2nd Worst: 100.30</td>
</tr>
<tr>
<td>10</td>
<td>95.17</td>
</tr>
<tr>
<td>11</td>
<td>Worst: 454.50</td>
</tr>
</tbody>
</table>
Figure 2.2: Power Gating

Table 2.3: NAND3 Leakage 0.18 \textmu m Technology

<table>
<thead>
<tr>
<th>Input</th>
<th>Leakage (nA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Best: 22.84</td>
</tr>
<tr>
<td>001</td>
<td>37.84</td>
</tr>
<tr>
<td>010</td>
<td>37.84</td>
</tr>
<tr>
<td>011</td>
<td>2nd Worst: 100.30</td>
</tr>
<tr>
<td>100</td>
<td>37.01</td>
</tr>
<tr>
<td>101</td>
<td>95.17</td>
</tr>
<tr>
<td>110</td>
<td>94.87</td>
</tr>
<tr>
<td>111</td>
<td>Worst: 852.40</td>
</tr>
</tbody>
</table>
Figure 2.3: Original MCNC Benchmark Circuit C17. Total Leakage is 831.08 nA under the Optimal MLV [6].

Figure 2.4: New Circuit C17 with Three Gates Replaced. Total Leakage is 476.88 nA under the Same MLV [6].
2.3.2 Active Techniques

As the leakage depends only on the current input vector, during the idle mode, we can apply the minimum leakage vector (MLV). Thus, MLV needs to be determined \textit{a priori} and incorporated into the circuit. This technique is known as the Input Vector Control (IVC). Tables 2.1, 2.2, and 2.3 show how the leakage varies with input vector for inverter, NAND2 and NAND3, respectively [6]. Best case input vector results in minimum leakage in the circuit. For an \(n\)-input module, as the input space grows exponentially (\(2^n\)), MLV determination heuristics have been proposed [6, 11]. Figures 2.3 and 2.4 show how the gate replacement technique is implemented in [6]. The minimum leakage vector from exhaustive search corresponds to total leakage current of 831.08 nA as shown in 2.3. Gates in worst leakage state make significant contribution to total circuit leakage. Gate replacement technique replaces the high leakage gates with low leakage gates such that the functionality remains the same in active mode. NAND2 gate is replaced by NAND3 and complement of SLEEP signal is connected to third input of NAND3 as shown in 2.4. The sleep signal ensures that the functionality is not changed in active mode and the leakage is reduced in sleep mode. Since this replacement change the gate output, fanout gates are replaced as well to ensure correct functionality during active time. Gate replacement reduces the circuit leakage to 468.88 nA which is a 43\% reduction. Transistor stacking is effective in reducing leakage power. Hence, several design techniques [27, 28, 29] that favor increased transistor stacking are proposed.

Table 2.4 compares standby and active techniques. Compared to the standby techniques, the main advantage of active techniques is that the circuit can switch from idle to active mode with little delay penalty. For IVC technique the area penalty occurs due to additional hardware needed to incorporate MLV into the circuit. Area overhead is also incurred due to usage of high Vt cut off
Table 2.4: Comparison of Standby and Active Leakage Minimization Techniques

<table>
<thead>
<tr>
<th>Item</th>
<th>Standby</th>
<th>Active</th>
</tr>
</thead>
<tbody>
<tr>
<td>Techniques</td>
<td>MTCMOS, Power Gating, SCCMOS</td>
<td>Input Vector Control, Force Stacking, Sleepy Stack, Power Gating with Stacking</td>
</tr>
<tr>
<td>Delay Penalty</td>
<td>Large</td>
<td>Small to None</td>
</tr>
<tr>
<td>Controller Design</td>
<td>Complex</td>
<td>Simple</td>
</tr>
<tr>
<td>Area Overhead</td>
<td>Considerable overhead due to high $V_t$ cutoff transistor</td>
<td>Slight overhead due to additional hardware to incorporate MLV</td>
</tr>
<tr>
<td>Leakage Reduction Method</td>
<td>Module cut off</td>
<td>Transistor stacking</td>
</tr>
<tr>
<td>Leakage Savings</td>
<td>Very good</td>
<td>Good</td>
</tr>
</tbody>
</table>

transistor. Delay penalty is incurred if this additional hardware is in the critical path of the circuit. Leakage saving of stand by techniques is better than active techniques because selective modules are completely cut off. A choice between stand by and active techniques is made by taking the advantages and drawbacks of both techniques into consideration.

2.4 Input Vector Control Technique

As IVC incurs little delay penalty, many researchers have proposed using IVC bound leakage power minimization. To the best of our knowledge, all the proposed IVC techniques are at the logic level.

Abdollahi, Fallah, and Pedram [11] propose gate-level leakage reduction with two techniques. The first technique is an input vector control wherein SAT based formulation is employed to find the minimum leakage vector. The second technique involves adding NMOS and PMOS transistors to the gate in order to increase the controllability of the internal signals. The additional transistors increase the stacking effect, leading to leakage current reduction. The authors report over 70% leakage reduction at the expense of up to 15% delay penalty for MCNC91 benchmarks at 0.18 µm technology node.
Gao and Hayes [30] present integer linear programming and mixed integer linear programming (MILP) approaches for leakage reduction by means of input vector control. MILP performs better than ILP and is thirteen times faster. Average leakage current is about 25% larger than minimum leakage current.

IVC technique does not work effectively for circuits with large logic depth. Yuan and Qu [6] have proposed a technique to replace the gates of worst leakage state with other libraries in active mode. A divide-and-conquer approach is presented that integrates gate replacement, an optimal MLV searching algorithm for tree circuits, and a genetic algorithm to connect the tree circuits. Compared with the leakage achieved by optimal MLV in small circuits, the gate replacement heuristic and the divide-and-conquer approach can reduce on average 13% and 17% leakage, respectively for MCNC91 benchmarks at 0.18 µm technology node.

2.5 Fractals and Self Similarity

"Fractals are shapes made of parts similar to the whole" [17]. Property of scaling is exhibited by fractals, which means the degree of irregularity in them tends to be identical at all scales [17, 18]. Figure 2.5 [31] shows a romanesco broccoli which is an example of a naturally occurring fractal. Figure 2.6 [32] shows another self similar fractal structure called H-tree. VLSI structures with bit sliced design exhibit self similarity in the input space due to uniformity in their layouts. The main classifications of fractals are time or space, self-similar or self-affine, and deterministic or stochastic. Space fractals are the data structures exhibiting the fractal property in the space domain, while time fractals are those exhibiting fractal property in time domain. Self-similar fractals have symmetry over entire scale, which is identical to recursion, i.e., a pattern inside another pattern. The details
are spread across finer and finer scales with certain constant measurements. The Hurst parameter 
\((0<H<1)\), is a measure of the correlation or long range dependence in the data which leads to the 
fractal behavior of the data set [33]. If the Hurst value is 0.5, then it is a random process. A process 
with Hurst value less than 0.5 exhibits anti-persistence. When Hurst value is between \(0.5<H<1\), 
the process has long term persistence [17].

![Romanesco Broccoli](https://via.placeholder.com/150)

Figure 2.5: Romanesco Broccoli ([31] public domain image). This image is taken from wikimedia.org.

A stochastic process can be said to exhibit fractal behavior [34] if the Hurst value \((H)\) is 
between 0.5 and 1. Methods to estimate Hurst parameter are described in [33]. In this work, we 
used the R/S plot method.

Self similarity was used to detect anomalous events in network traffic [35]. Results from [35] 
show that self similarity of network traffic is temporarily disturbed in the event of an attack. In the 
Radjassamy et al. [33], a vector compaction technique was proposed to generate a compact vector 
set which is representative of original vector set such that it mimics power-determining behavior of
the latter. Qian et al. [36] described Hurst parameter based financial market analysis where series with high $H$ are predicted more accurately than those with $H$ close to 0.5.

2.6 Interval Arithmetic

Interval arithmetic (IA) [37] is concerned with arithmetic operations such as addition and subtraction on intervals. The intervals can be either discrete or continuous. IA has been extensively applied in error bound analysis arising in numerical analysis. In this work, we are concerned with integer arithmetic. Therefore, we restrict our discussion to integer intervals. For floating point intervals, rounded interval arithmetic can be used. An interval $I = [a, b]$ represents all integers $a \leq i \leq b$. Further, the above interval is a closed interval as it includes both extreme values. We can have an open interval, such as $I = (a, b)$ where $a < i < b$. The width of an interval is the difference between the extreme values $|b - a|$. If the interval width is zero, then the interval is referred to as a degenerate interval (for example $[a, a]$), where $a$ is an integer.
Given two intervals \( U = [a, b] \) and \( V = [c, d] \), the following equations hold:

\[
U + V = [a + c, b + d] 
\]

\[
U - V = [a - d, b - c] 
\]

\[
U \ast V = [\min(a \ast c, a \ast d, b \ast c, b \ast d), \max(a \ast c, a \ast d, b \ast c, b \ast d)] 
\]

\[
U \div V = [\min(a \div c, a \div d, b \div c, b \div d), \max(a \div c, a \div d, b \div c, b \div d)] 
\]

2.7 NBTI Background

As the CMOS technology scaling entered sub-100nm regime, \textit{Bias Temperature Instability} (BTI) effect in devices has been identified as a major aging mechanism affecting circuit lifetime and reliability. Negative BTI (NBTI) affects PMOS devices, while positive BTI (PBTI) affects NMOS devices.

2.7.1 NBTI Phenomenon

Briefly, the NBTI effect can be described as follows (Figures 2.7 and 2.8): when a PMOS transistor is ON, the Hydrogen (H) atoms in the Si-SiO\(_2\) interface get released and combine to form H\(_2\) molecules. Under the influence of the electric field, these molecules drift into gate oxide layer leaving Si\(^+\) interface traps. The net effect is an increase in threshold voltage, resulting in speed degradation. Thus, when a PMOS transistor is active it is said to be in \textit{stress} mode. When the device is turned off, it can \textit{recover} partially (i.e., the threshold voltage recovers). In an NMOS device, a similar effect known as PBTI (Positive BTI) can occur. We can distinguish between two
scenarios for a device – static-BTI and dynamic-BTI. In the static case, the gate voltage is held constant for a long time i.e., the device is under stress for a prolonged period. In the dynamic case, the gate voltage switches frequently, putting the device in alternating stress and recovery modes.

![Device under Stress due to Bias Temperature Instability (BTI)](image)

Figure 2.7: Illustration of a Device under Stress due to Bias Temperature Instability (BTI)

\[ \Delta V_t = (1 + m) \chi \cdot q \cdot N_{IT}/C_{ox} \]  \hspace{1cm} (2.5)

\[ N_{IT}(t) = \left( k_I N_o \right)^{2/3} \left( k_H/k_{H_2} \right)^{1/3} \cdot (6D_{H_2}t)^{1/6} \]  \hspace{1cm} (2.6)

\[ N_{IT}(t_o + t_r) = \frac{N_{IT}(t_o)}{1 + \sqrt{\frac{\xi_{tr}}{(t_o + t_r)}}} \]  \hspace{1cm} (2.7)

\[ \Delta D = \gamma \frac{n \Delta V_t}{(V_{gs} - V_t)} \]  \hspace{1cm} (2.8)

Figure 2.8: Threshold Voltage \((V_t)\) Degradation and Recovery Model

### 2.7.2 Device-level BTI Models

Based on Reaction-Diffusion (RD) theory, a comprehensive device level NBTI model (Eqs. 2.5-2.7) has been proposed by Alam et al. [38] that accurately predicts the temporal \(\Delta V_t\) degradation and recovery. Eq (2.5) models \(V_t\) degradation. Eq (2.6) predicts interface trap count \((N_{IT})\) generated due to BTI stress. Eq (2.7) predicts \(N_{IT}\) that do not anneal during the recovery phase. Eq (2.8) predicts the additional transistor delay due to \(V_t\) degradation. The parameters in Eqs 2.5-
2.8 are: \( m \) = carrier mobility degradation that contributes to \( V_t \) degradation; \( q \) = electron charge; \( C_{ox} \) = oxide capacitance; \( \chi \) = 1 for NBTI and 0.5 for PBTI; \( N_0 \) = initial bond density; \( k_H = H \rightarrow H_2 \) conversion rate; \( k_{H_2} = H_2 \rightarrow H \) conversion rate; \( N_{IT}(t_o) \) = interface trap count at the beginning of recovery phase; \( \xi \) = diffusion constant; \( t_o \) = input period; \( t_r \) = recovery duration; \( n \) = velocity saturation index; and \( \gamma \) = activity factor.

Tables 2.5 and 2.6 report a compact prediction model [39] for NBTI degradation. Table 2.7 reports an NBTI long term prediction model [22] with random input sequence.

| Table 2.5: NBTI Degradation Model |

\[
V_{th} \text{ under NBTI} \\
\begin{array}{|c|c|}
\hline
\text{Static} & A(1 + \delta)t_{ox} + \sqrt{C(t - t_0)}^{2n} \\
\hline
\text{Dynamic} & (K_v(t - t_0)^{0.5} + 2\sqrt{\Delta V_{th0}})^{2n} \\
\text{Stress} & \Delta V_{th0}(1 - (2\xi_1 t_e + \sqrt{\xi_2 C(t - t_0)})/(2t_{ox} + \sqrt{C}t)) \\
\text{Recovery} & \hline
\end{array}
\]

| Table 2.6: NBTI Degradation Model - Parameters |

\[
K_v \quad (qt_{ox}/\epsilon)^3 K^2 C_{ox} (V_{gs} - V_{th}) \sqrt{C_e}^{(2E_{ox}/E_0)} \\
C \quad T_0^{-1} e^{-E_o/kT} \\
t_e \quad t_{ox} \sqrt{(t - t_0)/t_1 - \sqrt{\xi_2 C(t - t_0)/2\xi_1}} \quad t - t_0 \geq t_1 \\
E_o (eV) \quad 0.49 \\
E_0 (V/nm) \quad 0.335 \\
\delta \quad 0.5 \\
K(s^{-0.25}, C^{-0.5}, nm^{-2}) \quad 80000 \\
\xi_1 \quad 0.9 \\
\xi_2 \quad 0.5 \\
T_0 \quad 10^{-8} \\
\]

| Table 2.7: NBTI Degradation Model with Random Input Sequence |

\[
| \Delta V_{th,t} | \quad \text{considering long term effect of NBTI} \\
\Delta V_{th,t} \quad (\sqrt{k_7^2 T_{clk} \alpha /\min(\alpha, 1 - \alpha)}(1 - \beta_t^{(2n)})^{2n} \\
\beta_t \quad 1 - (2\xi_1 t_e + \sqrt{\xi_2 C(1 - \alpha) T_{clk}})/(2t_{ox} + \sqrt{C}t) \\
\]

23
2.8 Factors Affecting BTI

The following are the circuit operating parameters and technology factors that affect BTI.

2.8.1 Temperature

H generation rate rises with temperature leading to increased interface traps resulting in exacerbated $\Delta V_t$ degradation ($\Delta V_t$ is exponentially dependent on temperature).

2.8.2 Active to Standby Ratio

During active (standby) time period, the circuit is under dynamic (static) BTI condition. It is shown [40] that the delay degradation can be as high as 5X under static NBTI condition compared to that in dynamic NBTI.

2.8.3 Supply Voltage

Intuitively, we would expect a lower supply voltage leads to less $\Delta V_t$ degradation due to reduced electric fields. However, it is shown [40] that there is an optimum $V_{dd}$ to minimize NBTI. This is due to increased sensitivity of circuit speed to $\Delta V_t$ although the variation itself is small.

2.8.4 Process Variations

Process variations in oxide thicknesses, doping levels, and channel lengths, etc., result in $V_t$ variations. A lower $V_t$ transistor has more BTI degradation compared to a higher $V_t$. 
2.8.5 CMOS Technology Node

As we descend into smaller technology nodes (≤45nm) BTI effect worsens with thinner gate oxides and high-k dielectrics. In above 45nm nodes, NBTI was dominant. In 45nm and below nodes, PBTI can no longer be ignored [41, 42, 43]. Khan et al. [42] reported that in 45nm technology, PBTI impact in NAND gates is 1.27X times higher than NBTI impact. In comparison, in NOR gates, NBTI impact is 2.19X that of PBTI. Thus, both effects need to be considered.

2.8.6 Circuit Topology

Transistor stacking affects NBTI negatively, therefore, NOR gates are worse off compared to NAND gates. Note that stacking is helpful in subthreshold leakage reduction and has been effectively used to minimize leakage [44]. Thus, leakage and BTI optimization objectives are conflicting when it comes to circuit topology selection.

2.9 BTI Analysis and Optimization

We organize the related research into three categories: (a) layout- and gate-level; (b) high-level (behavioral/RTL/micro-architectural); and (c) NBTI aware state encoding in FSMs

2.9.1 Layout and Gate-level BTI Analysis and Optimization

Paul et al. [45, 46] proposed early gate level degradation model and evaluated the severity of BTI in 70nm node. In [47], they also compared the severity of BTI in random logic and memories and found that memories are more susceptible. Kumar, Kim, and Sapatnekar [48] proposed gate-level model in terms of input signal probabilities and employed it in an NBTI-aware technology
mapping algorithm to yield better area (10%) and lower-power (12%) solutions as compared to those obtained with pessimistic NB Ti assumptions.

Wang et al. [49] proposed a technique to identify NB Ti critical gates on the critical path wherein only 1% of gates needed re-design to assure less than 10% speed degradation in 10 years. Wang et al. [40] proposed NB Ti timing analysis framework (cycle-accurate device- and gate-level models). For ISCAS89 benchmarks implemented in 65nm technology, they conclude: (a) under dynamic NB Ti, the delay degradation is a strong function of temperature and less sensitive to supply voltage; (b) static NB Ti impact is 5 times more than that of dynamic NB Ti. Bild et al. [50] proposed an internal node control technique to reduce static NB Ti impact. MILP formulation and linear heuristic approaches yielded 26.7% reduction of NB Ti impact for the ISCAS85 benchmarks. Wu and Marculescu proposed gate-level optimization based on logic restructuring and pin-reordering [51] and path sensitization [52] with little or no area overhead.

To reduce standby subthreshold leakage, input vector control (IVC) [53, 54, 55, 56, 57, 58, 59] is widely adopted. As the input values are held constant for long periods of idle time, several transistors will be under prolonged static BTI stress. Thus, sub-threshold leakage optimization is in conflict with BTI optimization, wherein, periodically a transistor should be switched off to allow for BTI recovery. Thus, researchers proposed the co-optimization of circuit aging and leakage power. Wang et al. proposed gate replacement technique [60, 61] for co-optimization problem with 15-30% improvement over IVC only technique. Lin et al. [62] propose transmission gate based technique with 18% and 33% improvements in leakage power and NB Ti degradation, at the expense of 19% area overhead. Firouzi et al. [63] have proposed linear programming based co-optimization with the focus on pareto curve based on both phenomena. Adaptive body bias (ABB) and Adaptive supply
voltage (ASV) techniques have been effectively employed by Kumar, Kim, and Sapatnekar [41] to recover from BTI degradation. Khan et al. [42] reported that in 45nm technology, PBTI impact in NAND gates is 1.27X times higher than NBTI impact. In comparison, in NOR gates, NBTI impact is 2.19X that of PBTI.

2.9.2 Behavioral/RT Level BTI Analysis and Optimization

Chen et al. [64] proposed a high level synthesis framework in which the degraded delay estimates guide resource binding algorithm while minimizing the leakage power with multi-$V_t$ assignment. Kumar et al. [65] present an RTL NBTI estimation wherein signal probability distributions at the source code level are propagated through delay macromodels. The estimates are accurate within 10% with a speedup of 18.2 times compared to gate-level simulations.

2.9.3 Micro-architectural Level BTI Analysis and Optimization

DeBole et al. [66] proposed an early-design stage NBTI estimation framework and evaluate micro-architectural components in three target technology nodes (65nm, 45nm, and 30nm). Abella et al. [67] from Intel propose an NBTI-aware processor (“Penelope”), with various strategies in combinational blocks and memory blocks (Register Files, Caches, etc.), leading to 13-18% reduction in guardbanding. Fu et al. proposed micro-architecture strategies for high-performance processors [68] and Network-on-Chips (NoCs) [69] and address simultaneous NBTI and process variation issues. For NoCs, they addressed the problem for virtual channel, intra- and inter-router aspects with guardband reduction of 47% while improving throughput by 27%. Bhardwaj et al. [70] proposed aging-aware adaptive routing algorithms for NoCs. Sylvester et al. [71] proposed a self-healing architecture for combating not only NBTI but also other aging mechanism such as
oxide breakdown, process variations, etc. Other micro-architectural approaches are modeling and analysis [72], register level self-immunity [73, 74], and GPGPU Register allocation [75].

2.10 Finite State Machines

A finite state machine (FSM) is used to model a sequential component of a digital system. Sequential logic synthesis is the process of generating an optimized hardware implementation from a state diagram. It starts with state minimization and state encoding to optimize area, power, and/or delay [76]. State minimization determines a functionally equivalent FSM with a minimum number of states. State encoding assigns unique state codes to each state of the FSM. Several state encoding schemes are in existence but binary state encoding is most common form of encoding used. The state codes can be optimized for area, power and/or delay [77].

Dynamic power of a circuit depends on switching activity. Hence, low power state encoding is formulated to minimize the number of state bit switches per transition. The state encoding problem is NP-hard and many heuristic algorithms have been proposed to assign codes in such a way that the states with high transition probability have a small hamming distance [76].

2.11 State Encoding in Finite State Machines (FSMs)

Yuan et al. [76] proposed a state duplication technique in case of FSMs with unused states. If a state has high number of transitions, an additional duplicate state is added and is assigned an unused state code. The transitions are divided among the original state and duplicate state so as to minimize the hamming distance. This technique reduces dynamic power of the circuit, but can only be employed in case of FSMs with unused state codes. In Roy et al. [78], low power state encoding
is achieved using a simulated annealing algorithm. The moves of simulated annealing include a swap between two existing state codes, or between an existing state code and an unused state code. Lee et al. [79] proposed a pseudo Boolean SAT based algorithm to minimize the peak current of an FSM. According to empirical studies in Gupta et al. [80], it is found that two hot and three hot state encoding schemes are suited to achieve optimal area and performance. Two hot encoding is solved as a constraint solving graph problem. Pradhan et al. [81] implements power gating technique for partitioning of state transitioning graph along with state encoding of FSM for low power. Genetic algorithm is employed for state encoding and partitioning problem. Significant power savings are achieved.

2.12 Simulated Annealing

Simulated annealing algorithms [82] are heuristic algorithms that are iterative and stochastic in nature. They start with a complete initial solution. The objective function improves the solution incrementally. In each iteration, search space is limited to neighborhood of the current solution and a new solution is identified. In greedy algorithms, only the solutions with lower cost than current solution are accepted in each iteration, consequently finding the local minimum. This is the main drawback of greedy algorithms. Simulated annealing rectifies this drawback.

2.12.1 Principle

"Annealing refers to controlled cooling of high temperature materials in material science" [82]. It aims to modify the atomic structure of the material until it stabilized in minimum energy configuration. At high temperature, atoms are in chaos while at low temperature, they reach an ordered state. The rate of cooling influences the final structure. Slow cooling ensures a more orderly material
structure and minimum energy configuration. At each temperature of the cooling schedule, atoms cool down and stabilize. Any drastic changes in the configuration are reduced at low temperatures. If the cooling rate is slow and size of temperature decrements are small, global minimum is most likely to be obtained. If not, a local minimum is obtained instead.

2.12.2 Simulated Annealing Optimization

Finding minimum cost solution in an optimization problem is similar to finding minimum energy state in material science [82]. The algorithm starts with a chaotic high cost solution and reaches a structured low cost solution. The simulated annealing algorithm starts with an arbitrary initial solution. At each step, a random walk is performed in the neighborhood to find a new solution with a small perturbation to current solution. The acceptance or rejection of this new solution is decided with a probability that depends on temperature parameter. At high temperature, the probability of acceptance is high and it reduces gradually with temperature. "The acceptance condition is:

\[ e^{-\frac{(\text{cost(curr_sol)}-\text{cost(next_sol)})}{T}} > r \]  \hspace{1cm} (2.9)

where curr_sol is the current solution, next_sol is the new solution after perturbation, T is the temperature parameter and r is a random number between 0 an 1 based on a uniform distribution" [82].

The simulated annealing algorithm is stochastic which means that two different runs return different solutions. This difference arises from probabilistic decisions such as acceptance and rejection of new solutions.
2.12.3 Simulated Annealing Algorithm

The algorithm [82] starts with an arbitrary initial solution and generates a new solution by small perturbation. The resulting trial\_cost (line 14) is compared with curr\_cost (line 7, 8). If the new cost is better, new solution is accepted (lines 16, 17). If not, the new solution is still accepted probabilistically depending on the random number r (lines 20, 21).

The acceptance probability depends on both cost and temperature. At high temperatures, acceptance is frequent. At low temperatures, probability of acceptance is low. Better results are obtained with high initial temperature and slow rate of cooling, but the run time is increased as well.

2.13 Chapter Summary

In this chapter, we introduced various leakage mechanisms and optimization techniques that exist in the literature. We discuss several works that incorporated input vector control. The concepts of fractals, self similarity, and interval arithmetic are introduced as well. NBTI background is introduced and literature survey on NBTI optimization is reported. Device models with NBTI are explained and existing optimization techniques are discussed. Further, FSMs are explained briefly and existing work on state encoding of FSMs is discussed. Finally, simulated annealing is briefly explained.
Algorithm simulated_annealing_algorithm

Input: Initial Solution $init\_sol$

Outputs: optimized new solution $curr\_sol$

begin

$T = T_0$

$i = 0$

curr_sol = COST(curr_sol)

curr_cost = T_0$

while($T > T_{\text{min}}$) do

while(stopping criterion is not met) do

$i = i + 1$

$(a_i, b_i) = \text{SELECT\_PAIR}(curr\_sol)$

trial_sol = TRY\_MOVE($a_i, b_i$)

trial_cost = COST(trial_sol)

$\Delta cost = \text{trial\_cost} - \text{curr\_cost}$

if $\Delta cost < 0$

curr_cost = trial_cost

curr_sol = MOVE($a_i, b_i$)

else

$r = \text{RANDOM}(0,1)$

if ($r < e^{-\Delta cost/T}$)

curr_cost = trial_cost

curr_sol = MOVE($a_i, b_i$)

end

end

$T = \gamma * T$

end

end Algorithm

[82]

Figure 2.9: Simulated Annealing Algorithm
CHAPTER 3 : INTERVAL ARITHMETIC BASED RTL INPUT VECTOR CONTROL FOR DATAPATH LEAKAGE MINIMIZATION ²

Input vector control is a well known technique in the area of leakage reduction. However, there is a need to minimize area overhead as well. Hence, in this chapter, we introduce an interval propagation based leakage minimization technique to obtain leakage savings and minimize area overhead. At RT level, input design space is a large and increases with bit width of the modules. To work with this input space, we introduce a self-similarity based module characterization in this chapter to search for low leakage vectors. Definitions and terminology used in this chapter are presented and experimental results are reported as well.

3.1 Definitions and Terminology

This section introduces definitions and terminology that are used in this chapter.

Definition 1. Leakage Power Function, \( \mathcal{P}(V, t, w) \) returns the leakage power of a module (of type \( t \) and bit width \( w \)) when input vector \( V \) is applied.

---

²This chapter was published in


Permissions are included in Appendix B.
The above function is defined for the purpose of presenting our idea. For example \( P([2, 3], +, 8) \) will return the leakage value of an 8-bit adder with inputs 2 and 3.

**Definition 2.** Estimated Lowest Leakage, \( P_{\text{low}}(t, w) \), is the lowest leakage value of a functional unit i.e., minimum \( P(V, t, w) \) for some \( V \).

\[ P_{\text{low}}(t, w) \] can be obtained by either lower bound leakage analysis [86] of the underlying circuit or by simulation.

**Definition 3.** Optimization Tolerance, \( \varepsilon \), is the permitted deviation from the estimated lowest leakage in any module.

\( \varepsilon \) is a user-specified constant. For example, if \( \varepsilon = 0.1 \), we can tolerate up to 10% increase in the leakage of any module in the design.

**Definition 4.** Low Leakage Vector, \( V(t, w) \), is an input vector \( v \) of a module of type \( t \) and width \( w \), such that

\[ P(v, t, w) \leq (1 + \varepsilon)P_{\text{low}}(t, w) \]  

(3.1)

**Definition 5.** Low Leakage Interval, \( L \), is an input interval such that for every vector \( v \in L \), \( v \) is a low leakage vector.

**Definition 6.** Low Leakage Interval Set, \( \mathcal{L}(t, w) \), is the set of all low leakage intervals of a given module of type \( t \) and \( w \).

**Definition 7.** Data Flow Graph, \( G(V, E) \), is a directed graph such that \( v_i \in V \) represents an operation and \( e = (v_i, v_j) \in E \) represents a data transfer from operation \( v_i \) to \( v_j \).
We also assume two functions at our disposal, $T(op)$ and $W(op)$, that return the type and width of a given operation $op$, respectively.

3.2 Self Similarity Based Monte Carlo Characterization

In this Section, we first study the leakage profiles of adder and multiplier modules, and then introduce a Monte Carlo module characterization based on self-similarity to extract low leakage interval set of a functional unit.

3.2.1 Leakage Profile and Scope for Optimization

Figures 3.1 and 3.2 show the leakage current distribution of an 8-bit ripple carry adder and an 8-bit parallel multiplier, respectively from exhaustive simulation. Figures 3.3 and 3.4 show the leakage current distribution of an 8b adder and multiplier, respectively from 1000 random vectors. Figures 3.5 and 3.6 show the leakage current distribution of an 16b adder and multiplier, respectively from 1000 random vectors. The data for the module instances have been generated by 1000 random vectors of the CMOS layouts in 16nm technology node. We observed that, even with increased bit size/complexity, the leakage distribution is normal for adder/multiplier. Several works in previous literature like [87] and [88], that looked into power distribution of circuits presented normal distribution in their work as well. The leakage power values are measured using Synopsys Nanosim [89] with PTM [90, 91] technology parameter values for 16nm node. Hence, it is safe to assume that the power distribution of adder or multiplier module with any bit width is normal.

Lets consider the exhaustive simulation of 8b adder and multiplier. The leakage current range for 8b adder is $[0.084\mu\text{A}, 4.3\mu\text{A}]$ and that of the 8b multiplier is $[1.4\mu\text{A}, 56\mu\text{A}]$. Thus, the approximate max-to-min ratio for adder and multiplier are 51 and 40 respectively. For $\varepsilon=0.1$, the
number of distinct low leakage vectors for the adder is 119. The percentage of input space that puts the adder in a low-leakage state ($\varepsilon=0.1$) is $(119/(2^8 \times 2^8)) \times 100 = 0.18\%$. These vectors can be merged to obtain low leakage intervals. The number of such intervals is 80. Similarly, for multiplier, the number of low leakage vectors is 490, and the size of the interval set is 329. The percentage of input space that puts the multiplier in a low-leakage state ($\varepsilon=0.1$) is $(490/(2^8 \times 2^8)) \times 100 = 0.74\%$. Based on these numbers, we can see that it only takes a small percentage of input space to reduce the leakage power of the module by a great extent. Our next challenge is to locate all these low leakage intervals in the entire input space.
3.2.2 Self Similarity

We observed that the input space of adder and multiplier modules exhibit self similarity property. The size vs hurst parameter plot of input space in Figure 3.7 shows the self similarity in an 8-bit parallel multiplier module. As the size of the block is increased, Hurst parameter of leakage always lied in self-similarity range. As self similarity is a fractal property, leakage distribution of adder/multiplier modules is fractal. Hence, we can predict that the adder/multiplier circuit exhibits the same kind of distribution (normal) as the parent distribution in Figures 3.3 and 3.4, irrespective of its size. We found empirically that when the input space of the module is divided in 4, 16 or 64 grid cells, each of the cells shows normal distribution as well. Also, the $H$ parameter of each cell lies in the self similarity range. The concept of self similarity enabled the formation of leakage
Figure 3.3: Leakage Current Distribution for 8b Adder from 1000 Random Vectors

intervals uniformly in the entire input space and made the Monte Carlo characterization scalable as discussed in Subsection 3.2.3.

3.2.3 Monte Carlo Technique

As the input space of a n-bit module instance grows exponentially, exhaustive simulation based low leakage interval extraction is not feasible. Therefore, we propose a two phased Monte-Carlo (MC) based approach.

Typically, an MC based approach has four steps: (a) input space determination, (b) input sampling based on a probability distribution, (c) computation of property of interest, and (d) result aggregation. Table 3.2 gives an outline describing the two Monte Carlo runs.
Figure 3.4: Leakage Current Distribution for 8b Multiplier from 1000 Random Vectors

\[ n = \frac{\log_e (1 - \alpha)}{\log_e (1 - \beta)}. \] (3.2)

Our work incorporated a two-stage Monte-Carlo approach with a variation to [19] to deal with large input space. According to Halter and Najm [19], simulation with 460 random vectors will give us a 99% confidence that the lowest leakage vector found has less than 1% of the vectors in the entire population with lower leakage than the vector found from Equation 3.2. Equation 3.2 was derived from rank distribution in [92] which can only be applied to continuous distributions like normal, weibull, etc. Equation 3.2 from [19] is used for an input space with a normal distribution. In the equation, \( \alpha \) represents the confidence level and \( \beta \) represents the error tolerance. Given that the input space is normal even at the grid level, we simulate the circuit with 460 random vectors...
Figure 3.5: Leakage Current Distribution for 16b Adder from 1000 Random Vectors

to identify the low leakage points in each grid and form the low leakage intervals across the entire input space. Given the SPICE-level model of an \(n\)-bit module instance, we perform two successive MC runs. The property of interest is the leakage power.

- Run I - Coarse grained MC run: The input space under consideration is the entire space, i.e., \(2^{2n}\) input vectors. The input space is divided into grids along both x and y axes or just along one axis. For example, if a 256 x 256 space is divided into four grid cells along both axes, we have the following grid cells: [0, 0 - 127, 127], [0, 127 - 127, 255], [127, 0 - 255, 127], [127, 127 - 255, 255]. Appendix A consists of self similarity plot of 8b adder and 8b multiplier for different grids. These plots empirically prove the scalability of characterization.

We sample the input space with uniform probability distribution and then simulate the layout
with 460 samples in each grid to obtain leakage power values. In the result aggregation step, the power values for vectors covered in each grid cell are sorted in ascending order and then the lowest 5% of the values from each grid are used to identify low leakage regions in the input space. The LLV coverage increased with the number of grids. The input space covered in this run is shown in 3.10 and 3.11 for 8b adder and multiplier, respectively. The reduced set of values are marked by inverted delta in 3.10 and 3.11. The minimum leakage points found in grid cell division along x and y axes, or just along one axis, did not differ considerably. The coverage increased with the number of grid cells.

- Run II - Fine grained MC run: In this run, the input space is restricted to the regions identified in Run I. Figures 3.9 and 3.8 show the leakage profiles of functional units. The color
Figure 3.7: Hurst Parameter vs Block Size Plot

Figure 3.8: Leakage Profile for 8b Adder
blue represents the low leakage regions. From the leakage profiles of the functional units, we observed that low leakage points are clustered at specific regions. Hence, the sampling in this run is biased in the neighborhood of reduced space (low leakage vectors) identified previously. This helps concentrate the coverage in low leakage region. The result aggregation step involves merging input samples to create set of low leakage intervals. We usually cover as many points as possible in a reasonable amount of time, such as 1 hour.

Self-similarity is a fractal property, and fractals are scalable. The proposed MC based approach is scalable to larger circuits because of the self-similarity exhibited by the functional units. The simulation time is calculated by multiplying Nanosim[89] spice simulation time for single run with total number of runs (random samples). The simulations were carried out on a SunOS workstation (16 CPUs, 96GB RAM) using Synopsys Nanosim spice simulator [89].
Table 3.1: Time Estimation for Different Bit Widths (in Hours)

<table>
<thead>
<tr>
<th>Bit size</th>
<th>Adder</th>
<th>Multiplier</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>15.8</td>
<td>18.8</td>
</tr>
<tr>
<td>16</td>
<td>31.6</td>
<td>37.6</td>
</tr>
<tr>
<td>32</td>
<td>63.2</td>
<td>75.2</td>
</tr>
<tr>
<td>64</td>
<td>126.4</td>
<td>150.4</td>
</tr>
</tbody>
</table>

### 3.2.4 Run Time Complexity of Monte Carlo Characterization

We would like to estimate the run time complexity of the characterization for a module of size \( n \).

1. Stage I: The total run time of Stage I is \( S \times K \times T(n) \), where \( S \) is the number of sub-spaces resulting from partitioning the entire input space, \( K \) is the minimum number of vectors required for user-given confidence (\( \alpha \)) and error tolerance (\( \beta \)) levels, and \( T(n) \) is the simulation time for one vector. Note that \( K \) is a constant for fixed \( \alpha \) and \( \beta \) values. The user can keep the number of sub-spaces fixed i.e., \( S \) is a constant. \( T(n) \) is proportional to the gate complexity. In case
of ripple carry adder, $T(n) = O(n)$ as gate complexity grows linearly with bit width, while for parallel multiplier, $T(n) = O(n^2)$. Therefore, the complexity of Stage I is $O(n)$ and $O(n^2)$ for adder and multiplier respectively. If the user chooses to linearly scale the number of sub-spaces with the module complexity (i.e., $S = O(n)$), then the run time complexity increases to $O(n^2)$ and $O(n^3)$ for adder and multiplier respectively.

2. Stage II: We perform two steps: (1) local search around the vectors found in Stage I; and (2) then merge the low leakage vectors into low leakage intervals. The run-time complexity of first step is same as that of stage I, as we sample fixed number of vectors in the neighborhood of each vector from Stage I. The worst-case run time complexity of step 2 is same as that of a two-key sorting algorithm, $O(n \log n)$, since, we sort all input vectors and then merge immediate neighbors into intervals. Therefore, the complexity of Stage II in case of an adder is $O(n) + O(n \log n) = O(n \log n)$, while for multiplier it is $O(n^2) + O(n \log n) = O(n^2)$. 

Figure 3.11: Run I Input Space for 8b Multiplier
Table 3.2: Comparison of Monte Carlo Runs

<table>
<thead>
<tr>
<th>Properties</th>
<th>Run I - Coarse Grained</th>
<th>Run II - Fine Grained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input space</td>
<td>Entire input space</td>
<td>Immediate neighboring space of samples identified in Run I</td>
</tr>
<tr>
<td>Probability distribution</td>
<td>Uniform</td>
<td>Uniform</td>
</tr>
<tr>
<td>Property of interest computation</td>
<td>Leakage from Nanosim</td>
<td>Leakage from Nanosim</td>
</tr>
<tr>
<td>Result aggregation</td>
<td>Reduced sample set</td>
<td>Interval sets</td>
</tr>
</tbody>
</table>

Since the two stages are performed sequentially, the overall runtime complexity of MC based leakage interval characterization procedure is $O(n \log n)$ and $O(n^2)$ for $n$ bit adder and multiplier respectively.

### 3.3 Proposed Approach

The approach described in this paper has two types of variations: Top down and Bottom up. In top down technique, we carry out the propagation in two iterations. In first iteration, we
Figure 3.13: Run II Input Space for 8b Multiplier

start with a raw set of low leakage intervals at PIs, and propagate all the way to the bottom where we end up with a set of output intervals at the POs. In second iteration, output intervals are propagated in reverse direction minimizing the input interval set at each intermediate node and end up with a sparser set of low leakage intervals at PIs. This sparser set of intervals is processed with simulated annealing to arrive at the best LLV. In bottom up technique, we start with raw set of low leakage output intervals at POs and propagate them all the way to the PIs, where we end up with a minimized interval set. This minimized set is again processed with simulated annealing to arrive at the best LLV. In [93], a hierarchical reliability analysis scheme was tailored based on a similar top down and bottom up propagation technique.

The raw sets of low leakage intervals at PIs and POs are obtained through characterization presented in detail in Section 3.2. They depend on type of functional unit and tolerance value. Since our approach is a heuristic one, we implemented both top down and bottom up techniques, but
chose the best one in terms of leakage savings and area overhead. We present motivating examples illustrating top down and bottom up techniques.

### 3.3.1 Motivating Examples

1. Example 1: Top Down Approach

   In Figure 3.14, we show an example DFG with two adders (A1, A2) and one multiplier (M1). Let us say the low leakage vector sets are: $\mathcal{L}(+, 8) = \{([2, 4], [6, 8]), ([8,12], [8,12]), ([14, 20], [14,24])\}$ and $\mathcal{L}(*, 8) = \{([3, 4], [5, 6]), ([9, 10], [5, 6]), ([13, 24], [5, 6])\}$

   ![Figure 3.14: Example 1 a - Top Down Interval Propagation - Iteration 1](image)

   We start applying low leakage vectors for A1 and A2 i.e., $([2, 4], [8, 12], [14, 20])$ on the first input and $([6, 8], [8, 12], [14, 24])$ on the second input. Using interval arithmetic, we compute
A1’s and A2’s output range to be ([8, 12], [16, 24], [28, 44]). Thus, the input interval of M1 for input 1 is ([8, 12], [16, 24], [28, 44]) ∩ ([3, 4], [9, 10], [13, 24]) = ([9, 10], [16, 24]) and for input 2 is ([8, 12], [16, 24], [28, 44]) ∩ ([5, 6]) = ∅. Low leakage vectors are derived at the first input of M1. However, the computed input interval for second input of M1, ([5, 6]) does not overlap with the interval in \( L(\ast, 8) \). Therefore, we will have to introduce a control point at the second input of M1 to put M1 in low leakage mode. The control point consists of a multiplexer that can be used to force a low leakage vector in idle state. If the intersection of intervals happen to result in an empty set as above, we just need to insert a control point and start with entire low leakage vector set from that point. To determine the MLV at PIs, a backward propagation of minimized intervals is implemented for the same DFG as shown in Figure 3.15. The intervals available at input 1 of M1 are fed as outputs to A1, and these intervals are propagated to the inputs of A1. This gives a minimized interval set on which the simulated annealing algorithm is applied to find the best LLV.

2. Example 2: Bottom Up Approach

Now consider the same DFG for bottom up propagation technique, however with output low leakage vector sets: \( L(+, 8) = \{[2, 8]\} \) and \( L(\ast, 8) = \{[16, 24]\} \). For a given output low leakage set [16, 24] to M1, the corresponding input leakage sets are ([2, 4], [8, 8]). Similarly, for output set ([2, 4]), input sets are ([1, 2], [2, 2]), and for output set ([8, 8]), the corresponding input sets are ([2, 2], [4, 4]). We apply output low leakage vector intervals on primary outputs (Figure 3.16).

We start applying output low leakage vectors for M1 i.e., [16, 24] at output. It propagates corresponding input vectors [2, 4] to input 1 and [8, 8] to input 2 and M1 is set to low leakage mode. The input intervals at M1 are intersected with \( L(+, 8) \) and \( L(\ast, 8) \). Finally, output
intervals $[2, 4]$ and $[8, 8]$ are fed as input to A1 and A2, respectively. The corresponding input intervals ($[1, 2], [2, 2]$) and ($[2, 2], [4, 4]$) are propagated to inputs of A1 and A2, respectively. An input vector is chosen randomly from the available input intervals and both A1 and A2 are also set in low leakage mode. The best LLV is found by processing the reduced set with simulated annealing algorithm.

As seen from the above two examples, it is possible to achieve low leakage state for the entire design by applying low leakage vectors only at the primary inputs. Our proposed heuristic attempts to maximize the leakage power savings while keeping the control overhead as small as possible.
3.3.2 Low Leakage Vector Determination

A DFG, $G(V, E)$, is a directed graph such that $v_i \in V$ represents an operation and $e = (v_i, v_j) \in E$ represents a data transfer from operation $v_i$ to $v_j$. A low leakage interval set, $L(t, w)$, is the set of all low leakage input intervals of a given module of type $t$ and $w$. A low leakage output interval set, $LO(t, w)$, is the set of corresponding outputs of low leakage input intervals of a given module of type $t$ and $w$. Given the following two inputs: a data flow graph $G(V, E)$ and set of low leakage interval sets, $\bigcup_{t, w} L(t, w)$ and $\bigcup_{t, w} LO(t, w)$, for all distinct operations of type $t$ and width $w$, we need to identify best low leakage primary input vector and a set of control points $C$ such that the objective functions, $\sum_{v_i \in V} P(V, T(v_i), W(v_i))$ and $C$, are minimized. $P(V, t, w)$ is the leakage power function which calculates the total leakage of the filter and $C$ is the set of control points. Figure 3.17 shows the pseudo-code of the proposed heuristic for low leakage vector determination.
It accepts an input DFG (directed acyclic graph) and low leakage vector sets for distinct types and operations obtained from the characterization procedure as described in section 3.2. Breadth First Search is used to cover all the nodes of the graph. Flag is used to determine the direction of propagation (top down or bottom up).

First, the graph is topologically sorted (line 5) to yield a sorted list $L$. A set $C$ that collects the control points, is initialized (line 6). The for loop in lines 8–27 visits each node in the order specified by $L$. If a node is a PI node (i.e., both inputs to the node are primary), then the intervals on both inputs are initialized to the appropriate low leakage vector sets (line 13). Both inputs are added to the set $C$ (line 14). On line 16, we call a function `Interval_Propagate()` that accepts an ordered interval pair and the operation type of the node (i.e., $\mathcal{T}(v_i)$). `Interval_Propagate()` implements the interval arithmetic equations as mentioned in subsection 2.6 and returns an appropriate output interval $I_c$.

In line 21, we call a function `Interval_Intersection()` shown in Figure 3.18 to check if the computed interval is contained in low leakage vector set of the successor $v_j$. If the check succeeds, then we move onto to the next node in the list. If the check fails, then a new control point is inserted by resetting the inputs of node $v_j$ to it’s low leakage vector set (lines 22 –26) and adding the inputs of $v_j$ to control point set. In line 28, a function `Back_Propagate()` further truncates the reduced interval set at primary outputs, $\mathcal{LO}_{\text{reduced}}(t, w)$, by performing similar propagation in backward direction to primary inputs as shown in Figure 3.19. The backward propagation is described in the example from subsection 3.3.1. In case of bottom up propagation, the `Back_Propagate()` function (line 31) is performed on the complete low leakage output interval set, $\mathcal{LO}(t, w)$, to obtain reduced interval set at primary inputs.
Algorithm find_LLV
Inputs: (a) Graph G(V,E); (b) Low Leakage Vector Sets; (c) Flag
Outputs: output_llv and Control Points

begin
L ← Topological_Sort(G) /* L is a sorted list */
C ← ∅ /* internal control points */
if(Flag == TopDown) then
    foreach v_i ∈ L do
        Let a and b denote input edges of v_i
        c the output edge of v_i
        if v_i is a PI node
            I_{a,b} ← L(T(v_i), W(v_i))
            C ← C ∪ {a, b}
        end if
        I_c ← Interval_Propagate(I_{a,b}, T(v_i))
        Let v_j be the successor of v_i
        Let d be the second input of v_j
        /* check for interval intersection */
        contains ← FALSE
        Interval_Intersection(I_c, L(T(v_j), W(v_j)))
        if(contains == FALSE) then
            /* insert a new control point */
            I_{c,d} ← L(T(v_j), W(v_j))
            C ← C ∪ {c, d}
        end if
    end for
    reduced LLV set ← Back_Propagate
    ( LÖreduced(T(v_j), W(v_j)))
else
    reduced LLV set ← Back_Propagate
    ( LÖ(T(v_j), W(v_j)))
end if
output_llv ← Simulated_Annealing_Leakage
(reduced LLV set)
end Algorithm

Figure 3.17: Algorithm to Determine Low Leakage Vector
Algorithm Interval_Intersection
begin
Inputs: Interval set at input/output of $v_i$, $\cap I_c$, $\mathcal{L}(T(v_j))$
Outputs: contains
foreach $L \in \mathcal{L}(T(v_j), W(v_j))$ do
if $L \cap I_c \neq \emptyset$ then
    contains $\leftarrow$ TRUE
    break
end if
end for
end Algorithm

Figure 3.18: Algorithm to Check for Interval Intersection

Algorithm Back_Propagate
begin
Inputs: (a) Graph $G(V,E)$; (b) Low Leakage Output Vector Set
Outputs: reduced LLV set and Control Points
foreach $v_i \in L$ do
    Let a and b denote input edges of $v_i$
    c the output edge of $v_i$
    if $v_i$ is a PO node then
        $I_c \leftarrow \mathcal{L}O(T(v_i), W(v_i))$
        $C \leftarrow C \cup \{c\}$
    end if
    $I_{a,b} \leftarrow$ Interval_Propagate($I_c, T(v_i)$)
    Let $v_j$ be the predecessor of $v_i$
    Let a be the output of $v_j$
    /* check for interval intersection */
    contains $\leftarrow$ FALSE
    $\text{Interval\_Intersection}(I_a, \mathcal{L}O(T(v_j), W(v_j)))$
    if(contains == FALSE) then
        /* insert a new control point */
        $I_a \leftarrow \mathcal{L}(T(v_j), W(v_j))$
        $C \leftarrow C \cup \{a\}$
    end if
end for
end Algorithm

Figure 3.19: Algorithm for Back Propagation
Algorithm Simulated_Annealing_Leakage

Inputs: (a) Reduced Low Leakage Vector Sets

Outputs: Best low leakage vector outputllv

begin

Temp ← 100 ; gamma ← 0.99
bestllv ← random(reduced LLV set)
currllv ← bestllv
bestleak ← leakage(currllv)

while Temp >0 do

foreach iteration in 500*Temp do

currllv ← currllv + random(reduced LLV set in Tempval window)
currleak ← leakage(currllv)
if currleak <bestleak or
random(0,1) <= pow(e,-(bestleak - currleak)/Temp )
then
if currleak <bestleak
then
outputllv ← currllv
end if
bestllv ← currllv
bestleak ← leakage(currllv)
end if

Temp ← Temp * gamma
end while

bestleak ← Est_Leakage(outputllv)

end Algorithm

Figure 3.20: Simulated Annealing Algorithm to Find the Best LLV

In line 34, we call a function Simulated_Annealing_Leakage() shown in Figure 3.20. This function is applied on the reduced LLV set to find the best LLV. The variable for SA is an LLV and the cooling function is the corresponding leakage value for a given LLV. The initial temperature value Temp is set to 100 and cooling coefficient γ to 0.99. The number of iterations for each temperature value varies with the temperature value itself (500 x Temp). An LLV is initially chosen at random from the reduced interval set. In each iteration, another random LLV is chosen from the neighborhood of the previous LLV. The neighborhood window size is also a function of temperature value itself (Temp). Est_Leakage() function calculates the leakage of the DFG for
any given input vector. *output_llv* is the best LLV that sets the circuit in low leakage state. In the above algorithm, we assume only one successor for each node. This assumption is made to simplify the presentation of the algorithm. It is straightforward to extend the algorithm to multiple successors.

### 3.3.3 Run Time Complexity of LLV Determination Algorithm

In the first step of *find_LLV()* function, topological sort has a time complexity of $O(|V| + |E|)$. The maximum number of edges in a DAG is $(|V|)(|V| - 1)/2$. Hence the time complexity of topological sort is $O(|V|^2)$. The for loop (lines 8-27) runs $|V|$ times. If the node is PI, initial interval set is initialized at the node. This initialization takes a constant time. We also have the *Interval_Propagate()* and *Interval_Intersection()* which take time proportional to $I^2$, where $I$ is the number of low leakage intervals. The number of intervals $I$ is a constant obtained from characterization as described in subsection 3.2.3. *Back_Propagate()* function on lines 28 and 31 has the same complexity as that of the code on lines 8-27. Finally, simulated annealing algorithm takes a constant time based on the quality of solution desired. Hence, the run time complexity of the algorithm is $O(|V|^2)$.

### 3.4 Experimental Results

We report the experiment results obtained by applying the proposed input vector control technique on five datapath-intensive benchmarks, namely, IIR, FIR, Elliptic, Lattice, and Differential Equation Solver. As described in section 3.2 the library is characterized *a priori* and the low leakage vectors sets saved. The DFG of each filter is then processed to identify a low leakage vector. The leakage power values are measured at the layout level using Synopsys NANOSIM[89] version 56.
F-2011.09-SP2. We employ the Predictive Technology Models for 16nm technology node generated by the online model generation tool available on the ASU PTM website[90]. We simulate each layout with 1000 random vectors and measure the dynamic and subthreshold leakage currents. Resource sharing is incorporated to the approach from preliminary work [85] and results are tabulated in the subsection 3.4.1. Comparison of our work with [6] in presented in the subsection 3.4.2. In subsection 3.4.3, we present results from top down propagation approach. The bottom up propagation technique did not help optimize the leakage power, so its corresponding results are not reported.

3.4.1 Resource Sharing with Propagation

Table 3.3 reports the leakage savings and area overhead without resource sharing [85]. We can observe that both leakage and dynamic components are significantly reduced. The savings in dynamic power are the side-effect of holding the inputs stable to the circuit. Further, resource sharing (resource optimization technique during high level synthesis) is incorporated. The results for resource shared filters are reported in Table 3.4. In this case, the resource count is assumed to be two both for adder and multiplier. The average leakage savings with and without resource sharing 42.5% and 74.4%, respectively with 0.98% and 1.9% area overhead, respectively.

Tables 3.3 and 3.4 report the area penalty and the number of control points in addition to leakage savings. The area penalty is measured in terms of the transistor count. The fifth column reports the number of control points inserted in the intermediate nodes. The number of control points is directly proportional to the percentage of input space in consideration to create the low leakage intervals. As we can observe, the area overhead (sixth column) in Tables 3.3 and 3.4 are in the range of 0.48% to 3.93% and 0% to 1.8%, respectively with an average of 1.9% and 1.0%, respectively. The dynamic power in case of resource sharing is drastically reduced because the
Table 3.3: Power Savings, Control Points and Area Overhead without Resource Sharing

<table>
<thead>
<tr>
<th>Design</th>
<th>Leakage Savings (%)</th>
<th>Dynamic Savings (%)</th>
<th>Total Savings (%)</th>
<th>Control Points</th>
<th>Area Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffeq (2+, 5*)</td>
<td>77.29</td>
<td>52.79</td>
<td>70.51</td>
<td>1</td>
<td>0.48</td>
</tr>
<tr>
<td>EWF (26+, 8*)</td>
<td>68.13</td>
<td>65.11</td>
<td>66.79</td>
<td>8</td>
<td>2.16</td>
</tr>
<tr>
<td>FIR (4+, 5*)</td>
<td>77.57</td>
<td>62.52</td>
<td>71.81</td>
<td>4</td>
<td>1.77</td>
</tr>
<tr>
<td>IIR (4+, 5*)</td>
<td>79.83</td>
<td>61.47</td>
<td>74.31</td>
<td>3</td>
<td>1.33</td>
</tr>
<tr>
<td>Lattice (8+, 5*)</td>
<td>69.46</td>
<td>54.07</td>
<td>65.26</td>
<td>9</td>
<td>3.93</td>
</tr>
</tbody>
</table>

Table 3.4: Power Savings, Control Points and Area Overhead with Resource Sharing

<table>
<thead>
<tr>
<th>Design</th>
<th>Leakage Savings (%)</th>
<th>Dynamic Savings (%)</th>
<th>Total Savings (%)</th>
<th>Control Points</th>
<th>Area Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffeq (2+, 5*)</td>
<td>47.98</td>
<td>90.73</td>
<td>83.72</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>EWF (26+, 8*)</td>
<td>54.78</td>
<td>89.74</td>
<td>83.70</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>FIR (4+, 5*)</td>
<td>45.85</td>
<td>91.41</td>
<td>84.11</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>IIR (4+, 5*)</td>
<td>15.88</td>
<td>96.90</td>
<td>83.70</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>Lattice (8+, 5*)</td>
<td>47.85</td>
<td>89.15</td>
<td>81.75</td>
<td>1</td>
<td>0.8</td>
</tr>
</tbody>
</table>

The number of resource count is reduced as well. This explains the increase in dynamic power savings in Table 3.4.

3.4.2 Comparison with Existing Work

Our technique is implemented at register transfer level of abstraction. To the best of our knowledge, there is no work done at RTL that uses input vector control for leakage reduction. We, therefore, compared our method with a technique from prior work at the gate level. Out of the two relevant works from literature [11] and [6], [6] is an improvement over the technique used in [11]. Therefore, we compare our work with [6] which proposes a gate replacement technique.

The essence of gate replacement is to replace a logic gate that is at its worst leakage state (WLS) by another library gate [6]. The new gate is selected in such a way that it has one extra input
bit which is connected to SLEEP signal, retains the operation during active mode when SLEEP=0 and is put in low leakage mode when SLEEP=1. If the replacement changes output of the gate in sleep mode, leakage change in fanout gates is also taken into consideration and a gate replacement decision is made based on resultant increase or reduction of total leakage power. In [6], experiments are carried out by applying an MLV at primary inputs and further implementing gate replacement algorithm. We followed the same approach for comparison.

We synthesized gate level circuits for our benchmarks only with the specific set of gates used in [6]. The algorithm is implemented in C. Commercial gate level synthesis tools are not opted due to the gate level processing involved in the gate replacement algorithm implementation. We conducted two experiments using the gate replacement algorithm.

1. Experiment 1: In this experiment, MLV is provided to each module instance that receives primary inputs in the circuit followed by the gate replacement algorithm. Table 3.5 shows the comparison of leakage savings and area overhead in both the techniques. Column 2 shows the improvement by gate replacement algorithm over a circuit where MLV is applied at primary inputs. Column 3 shows the improvement by application of MLV to primary inputs and gate replacement algorithm over a circuit where a random input vector is applied to primary inputs. Gate replacement algorithm from [6] incurred 6.64% area overhead while our technique incurred no area overhead.
Table 3.6: Experiment 2 - MLV at all DFG Nodes. Comparison with [6].

<table>
<thead>
<tr>
<th>Design</th>
<th>Leakage Savings (%)</th>
<th>Area Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffeq</td>
<td>20.88</td>
<td>32.24</td>
</tr>
<tr>
<td>EWF</td>
<td>22.43</td>
<td>33.80</td>
</tr>
<tr>
<td>FIR</td>
<td>21.14</td>
<td>32.51</td>
</tr>
<tr>
<td>IIR</td>
<td>21.14</td>
<td>32.51</td>
</tr>
<tr>
<td>Lattice</td>
<td>21.39</td>
<td>32.75</td>
</tr>
</tbody>
</table>

2. Experiment 2: In this experiment, MLV is applied to every module instance in the circuit, and gate replacement algorithm from [6] is applied. Leakage savings and area overhead are reported in Table 3.6.

From these experiments, we demonstrate that the proposed algorithm yields better leakage savings over the gate-replacement algorithm [6] with no area overhead.

3.4.3 Top Down and Bottom Up Propagation

To obtain the experimental results, we initially vary the value of $\varepsilon$ and determine the optimum (lowest) value for maximum leakage savings (i.e., leakage increase in any module up to $\varepsilon\%$ is tolerated). Figures 3.21 and 3.22 shows the variation of leakage savings with tolerance in different designs for top down and bottom up techniques, respectively. From Figure 3.21, it can be observed that the optimum tolerance value for top down technique is 10% (i.e., leakage increase in any module up to 10% is tolerated). The minimum leakage vector is captured within 10% tolerance itself. Even if the tolerance is increased above 10%, it is observed that leakage savings do not increase any further. Hence, no area overhead is incurred. In case of bottom up as shown in Figure 3.22, optimum tolerance is observed to be 15%. Only one value of optimum tolerance is chosen for all benchmarks. This helps use a common module characterization for all the benchmark circuits.
as discussed in Subsection 3.2.3. Tables 3.7 and 3.8 reports the results for top down and bottom up interval propagation techniques, respectively, with simulated annealing. Column 3 presents the leakage value obtained from interval propagation with simulated annealing. We can see a significant improvement compared to the random case in column 2. Leakage savings are presented in column 4. The top down technique achieved average leakage savings of 93.6% for 10% tolerance and no area overhead. On the other hand, bottom up technique achieved average leakage savings of 89.2% for 15% tolerance and no area overhead. Hence, top down technique has better leakage savings and is preferred over bottom up technique.

As a reduced set of LLVs at primary inputs are formed using our technique, it beats the pure simulated annealing technique in execution time. Table 3.9 reports the simulation speed up obtained when we use simulated annealing on reduced set of intervals from top down propagation as opposed to pure simulated annealing. Columns 2 and 4 present the leakage values. Execution time is reported in columns 3 and 5 in Table 3.9. It shows that simulated annealing with top down propagation is much faster than simulated annealing alone to obtain a similar quality of solution. The difference in solution quality is also presented in column 6 of Table 3.9. For EWF benchmark, a solution that is much better than pure simulated annealing solution is obtained with interval propagation and simulated annealing combined. For the rest of the benchmarks, the difference between solutions is minimal. Execution speed up is reported in column 7.

Table 3.10 compares the quality of leakage obtained by pure simulated annealing(SA) for 10 minutes, 1 hour, 2 hours, and 3 hours with leakage obtained by top down propagation with simulated annealing which takes an average of 10 minutes as shown in Table 3.9. Columns 2, 3, 4, and 5 report the leakage obtained by pure simulated annealing. Column 6 reports the leakage
obtained from top down interval propagation incorporated with simulated annealing. It is observed that the leakage found by interval propagation with SA in 10 minutes is better than leakage found by pure SA in 3 hours. These results demonstrate the effectiveness of interval propagation technique.

![Figure 3.21: Leakage vs Tolerance in Top Down Propagation](image1)

![Figure 3.22: Leakage vs Tolerance in Bottom Up Propagation](image2)

3.5 Chapter Summary

In this chapter, a self-similarity based Monte Carlo characterization of RTL modules is introduced. Low leakage vector determination approach is presented along with motivational examples. Experimental results are reported for the leakage minimization technique and a comparison of our technique is done with gate replacement technique discussed in the literature.
Table 3.7: Percentage Power Savings in Top Down Propagation with Simulated Annealing

<table>
<thead>
<tr>
<th>Design</th>
<th>Leakage (µA)</th>
<th>Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>random</td>
<td>Top down + SA</td>
</tr>
<tr>
<td>DiffEq (2+, 5*)</td>
<td>197.9</td>
<td>16.42</td>
</tr>
<tr>
<td>EWF (26+, 8*)</td>
<td>654</td>
<td>64.13</td>
</tr>
<tr>
<td>FIR (4+, 5*)</td>
<td>220.9</td>
<td>9.95</td>
</tr>
<tr>
<td>IIR (4+, 5*)</td>
<td>266.8</td>
<td>11.94</td>
</tr>
<tr>
<td>Lattice (8+, 5*)</td>
<td>226.4</td>
<td>11.01</td>
</tr>
</tbody>
</table>

Table 3.8: Percentage Power Savings in Bottom Up Propagation with Simulated Annealing

<table>
<thead>
<tr>
<th>Design</th>
<th>Leakage (µA)</th>
<th>Savings (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>random</td>
<td>Bottom up + SA</td>
</tr>
<tr>
<td>DiffEq (2+, 5*)</td>
<td>197.9</td>
<td>21.6</td>
</tr>
<tr>
<td>EWF (26+, 8*)</td>
<td>654</td>
<td>72.8</td>
</tr>
<tr>
<td>FIR (4+, 5*)</td>
<td>220.9</td>
<td>25.8</td>
</tr>
<tr>
<td>IIR (4+, 5*)</td>
<td>266.8</td>
<td>25.3</td>
</tr>
<tr>
<td>Lattice (8+, 5*)</td>
<td>226.4</td>
<td>23.9</td>
</tr>
</tbody>
</table>

Table 3.9: Speed Up with Interval Propagation

<table>
<thead>
<tr>
<th>Design</th>
<th>Interval + SA</th>
<th>SA</th>
<th>Leakage Improvement</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Leakage (µA)</td>
<td>Time (min)</td>
<td>Leakage (µA)</td>
<td>Time (min)</td>
</tr>
<tr>
<td>DiffEq</td>
<td>20.60</td>
<td>12</td>
<td>18.75</td>
<td>170</td>
</tr>
<tr>
<td>EWF</td>
<td>55.24</td>
<td>13</td>
<td>71.62</td>
<td>812</td>
</tr>
<tr>
<td>FIR</td>
<td>16.50</td>
<td>2</td>
<td>16.30</td>
<td>247</td>
</tr>
<tr>
<td>IIR</td>
<td>16.55</td>
<td>18</td>
<td>14.46</td>
<td>244</td>
</tr>
<tr>
<td>Lattice</td>
<td>23.00</td>
<td>6</td>
<td>21.83</td>
<td>440</td>
</tr>
</tbody>
</table>

Table 3.10: Leakage Comparison of Pure SA with Interval Propagation + SA

<table>
<thead>
<tr>
<th>Design</th>
<th>Leakage(µA) with SA</th>
<th>Interval + SA leakage (time)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10min</td>
<td>1hr</td>
</tr>
<tr>
<td>DiffEq</td>
<td>32.08</td>
<td>27.65</td>
</tr>
<tr>
<td>EWF</td>
<td>99.12</td>
<td>83.50</td>
</tr>
<tr>
<td>FIR</td>
<td>34.74</td>
<td>31.80</td>
</tr>
<tr>
<td>IIR</td>
<td>45.53</td>
<td>27.56</td>
</tr>
<tr>
<td>Lattice</td>
<td>41.24</td>
<td>36.64</td>
</tr>
</tbody>
</table>
CHAPTER 4 : NBTI AND LEAKAGE CO-OPTIMIZATION WITH VECTOR CYCLING

The effects of NBTI and leakage can be opposing based on the stacking effect in a circuit. It is therefore essential not to ignore NBTI during leakage optimization. In this chapter, we introduce a simulated annealing based NBTI and leakage co-optimization and a back tracking algorithm.

4.1 Motivational Example

Assume an NBTI - leakage co-optimized input vector for circuit c17 (from ISCAS85 benchmark suite) is 11111. The critical path consists of nodes N6, N11 and N16 (gates G2, G4 and G5). For the sake of simplicity, we focus only on worst critical path in this example. In other experiments, near critical paths along with worst critical path are considered. In Figure 4.1, the

![Figure 4.1: C17 with Co-optimized Vector](image)
critical path is shown in gray and the gates in critical path are shaded. Each node corresponds to a specific PMOS transistor. Let the nodes N6, N11, N16 of transistors p6, p11, p16, respectively. When 11111 is applied to primary inputs, N6=1, N11=0, and N16=1 as shown in Figure 4.1. Hence, p6 and p16 are in recovery mode as their gate values are at logic '1' while p11 is in stress mode. Using backtracking algorithm, p11 should be recovered so logical value 1 is back propagated from node N11 to primary inputs which returns an input vector (alternative vector) 11101. In Figure 4.2, when 11101 is applied to primary inputs, then N6=0, N11=1, N16=0. Hence, p6 and p16 are in stress mode and p11 is in recovery mode. The transistor p11 is mutually exclusive to transistors p6 and p16 as shown in Figures 4.1 and 4.2. with different shading. When both vectors are cycled during idle time, overall stress of p6, p11 and p16 is reduced thereby alleviating NBTI degradation.

4.2 Proposed Approach

A primary input vector effects both leakage and NBTI of a CMOS circuit. In idle mode, leakage or NBTI friendly vector can be chosen to mitigate either phenomenon. We can also choose a
vector from the input space that optimizes both leakage and NBTI. This vector might result in suboptimal leakage reduction, but it optimizes NBTI degradation as well. In this work, we start with simulated annealing optimization for three different cases: (a) leakage only; (b) NBTI degradation only; and (c) both leakage and NBTI degradation. In the third case, we co-optimize NBTI and leakage based on the weights assigned to each objective during co-optimization. For example, if the weight on NBTI is 0.5 and the weight on leakage is 0.5, optimization works equally effectively on both NBTI and leakage. If the weight on NBTI is 0.1 and the weight on leakage is 0.9, optimization is concentrated on leakage rather than on NBTI. A back tracking algorithm is introduced post co-optimization to further optimize NBTI effect.

4.3 Vector Cycling

An active device under BTI stress will recover substantially (as much as 75% depending on duty cycle and input patterns) when turned off. Under static BTI condition, devices under stress must be given an opportunity to recover periodically. Further, in order to minimize area/power overhead, such relief should be provided primarily to gates on critical path(s) of the design. Vector cycling switches the primary input vector of a digital circuit between two vectors during idle time. Critical path stress transistor sets of both vectors should have minimum intersection.

For illustration purposes, 4.3 shows a simple combinational circuit with G1, G2, and G3 on critical path. Let us say, we have two static vectors V1 and V2. Above the gate-level diagram, for each vector, we identify the state of PMOS devices in each of these gates (for the sake of clarity, we only show PMOS devices in this example). Under V1, both PMOS devices of G1, and one PMOS device of G2 are under stress. If we switch to V2, then these devices enter relaxation mode; the
other PMOS device of G2 and one PMOS device of G3 are now under stress. By alternating between the two vectors, transistors on the critical path get a chance to recover from BTI stress. Of course, this idea can be extended to multiple vectors. We propose to cycle through carefully chosen vectors to periodically relax critical path transistors leading to enhancing the reliability and design lifetime. The choice of vectors and cycling frequency will lead to various area, delay, and power overheads.

4.4 Simulated Annealing Optimization

A simulated annealing approach requires an input variable and a cost function. Input variable is the primary input vector. Cost function can be solely NBTI or leakage of the circuit. Cost function used for co-optimization is:

\[
Cost = W_{\text{leak}} \times (\text{leakage}/\text{MAX\_leakage}) + W_{\text{NBTI}} \times (\text{NBTI\_delay}/\text{MAX\_NBTI\_delay})
\]  

(4.1)
where \( MAX_{leakage} \) and \( MAX_{NBTI\_delay} \) are the maximum leakage power and NBTI delay obtained from simulated annealing for leakage and NBTI degradation as shown in Figures 4.4 and 4.5. \( W_{leak} \) and \( W_{NBTI} \) represent the weight of each objective contributing to the cost function.

In the literature, most works used random samples to find the minimum leakage vector or minimum NBTI vector of a circuit. Simulated annealing is a technique that can yield better quality results compared to random sampling due to its ability to climb out of local minima. Simulated annealing code and parameters used in this work are experimented with a sample function and the code is verified for correctness. Initial temperature of the optimization is set to 100 \((Temp)\). The annealing co-efficient of simulated annealing is

\[
\gamma = e^{(\log(2) - \log(Temp)) / \text{num\_iterations}}
\]

(4.2)

where

1. \text{num\_iterations} is \text{total\_exec\_time} \times 60 / \text{(single\_func\_time} \times \text{total\_iterations}).
2. \text{total\_exec\_time} is the total time taken for simulated annealing optimization.
3. \text{single\_func\_time} is time taken by a single iteration.
4. \text{total\_iterations} is the number of iterations at each temperature value.

Annealing function is 0.95 \times PI\_size \times Temp / 200 where \( PI\_size \) is the number of bits in primary input vector. Annealing move in each iteration is bit flipping in a window across primary input vector defined by annealing function. Solution in each iteration is the primary input vector. We experiment with simulated annealing with different cost functions.
4.4.1 Simulated Annealing for Leakage Only Optimization

This simulated annealing experiment considers leakage only as cost function. The vector obtained from this experiment is the minimum leakage vector. The minimum leakage value estimated from this experiment is used to derive leakage overhead during co-optimization. Figure 4.4 shows the simulated annealing-based flow for leakage as cost function. In leakage framework, input vector generator generates random vectors by flipping bits in a given window which is a function of current temperature. Synopsys HSPICE tool is used to measure leakage.

4.4.2 Simulated Annealing for NBTI Only Optimization

This simulated annealing experiment considers NBTI degradation only as cost function. The output vector is the minimum NBTI vector. The minimum delay value from this experiment
represents the best case NBTI delay degradation. It is used to calculate percentage delay improve-
ment that needs to be compromised during co-optimization. NBTI degradation model used in our
work is adopted from [39] and is presented in Chapter 2. The primary input vectors obtained in
these experiments can be applied to a circuit at idle time to optimize leakage and NBTI effects,
respectively. Figure 4.5 show the simulated annealing based flow NBTI delay as cost function.

NBTI framework incorporates NBTI degradation model in Verilog-A compact transistor
models which are used with HSPICE tool. Input vector generator works the same as in case of
leakage framework. Threshold voltage degradation values of degraded transistors are obtained with
HSPICE simulation. A degraded transistor model is created for each transistor. NBTI degraded
spice circuit is developed from degraded transistor models using an in-house script. Synopsys NANOTIME STA tool is used to measure degraded delay from NBTI degraded spice circuit from which delay degradation is calculated.

4.4.3 Simulated Annealing for NBTI and Leakage Co-optimization

This simulated annealing experiment incorporates both leakage and NBTI effects in its cost function. The leakage and NBTI degradation corresponding to input vector obtained in this experiment are suboptimal. Figure 4.6 shows the simulated annealing framework for co-optimization of NBTI degradation and leakage. It combines the tools and models used in leakage and NBTI frameworks. The primary input vector obtained in this experiment is used in the backtracking algorithm to further optimize NBTI degradation.

The overall implementation framework is shown in Figure 4.7. Gate level netlist is converted to SPICE netlist using Synopsys NETTRAN tool. Transistor level static timing analysis is carried out on the SPICE netlist to identify critical path and near critical path nodes with their respective delay contributions. Simulated annealing is used in conjunction with backtracking algorithm to obtain inputs vectors to be cycled in idle time.

4.5 Back Tracking Algorithm

When NBTI-leakage co-optimized vector is applied at primary inputs, a set of PMOS transistors on critical and near critical paths, say $S_1$, are under stress. We need to obtain an alternate vector such that most of the PMOS transistors in $S_1$ are put in recovery mode and a different set of PMOS transistors, say $S_2$, are stressed. Ideally, sets $S_1$ and $S_2$ should be mutually exclusive, but however, this is not possible. Therefore, we obtain a vector such that minimum number of transis-
Figure 4.6: Simulated Annealing for NBTI-Leakage Co-optimization
Figure 4.7: Proposed Co-optimization Flow
tors are stressed by both vectors. In other words, intersection of sets $S_1$ and $S_2$ is to be minimized. A back-tracking algorithm is devised to obtain the alternate vector. Figure 4.8 shows the pseudo code of back tracking algorithm. The benchmark circuit and set of critical path nodes are given as inputs and an input vector is obtained as output. The algorithm starts with levelizing the circuit in line 5. Critical nodes of each level are sorted in decreasing order of delay contribution. Individual delay contribution of nodes is obtained from Synopsys Nanotime timing report. At each level, critical nodes are set to 1 (recovery mode) and propagated backwards to the previous level as shown in line 12 and 15. This step is repeated until primary inputs are reached. It may not be possible to set all the critical and near critical path transistors in recovery mode. However, the transistors that contribute to higher delay are preferred over the others to recover. NBTI-leakage co-optimized vector and alternate vector are cycled in idle time to obtain NBTI optimization. Switching intervals can usually be as long as several days to months depending on type of the application. The switching intervals can be scheduled such that negligible dynamic power overhead is incurred.

4.6 Experimental Results

We report the experiment results obtained by applying two vectors with minimum intersection of stress transistors sets in critical path. Experiments are conducted on five ISCAS85 benchmark circuits, namely, c432, c499, c880, c1908, and c2670. Leakage overhead in each case is reported as well.

NBTI device degradation model from [39] is modeled with Verilog-A and incorporated in HSPICE simulation to obtain threshold voltage degradation of all PMOS transistors. An in-house script is developed to create an NBTI degraded spice level circuit from the threshold voltage degra-
Algorithm back_tracking
Inputs: (a) Circuit Ckt(n); (b) critical_nodes N(n,c)
Outputs: primary_input_vector
begin
L ← Levelize(Ckt) /* L is a sorted list */
N ← Critical_Nodes_Sort(Ckt)
Start at primary outputs
foreach Lᵢ ∈ L do
   foreach Nᵢ ∈ N do
      if Nᵢ is a critical node
      then
         Propagate_1_output_to_input(Nᵢ)
      else
         if Nᵢ has no valid logic
         Propagate_N_i_output_to_input(Nᵢ)
      end
   end
end
end Algorithm

Figure 4.8: Back Tracking Algorithm to Determine Alternate Vector

dation. The Verilog-A code of the NBTI degradation model is verified with silicon results from [94]. Synopsys Nanotime transistor level static timing analysis tool is employed to measure the delay after degradation and also to find the critical and near critical paths of the circuit before degradation.

Figures 4.9 and 4.10 report the simulated annealing plots for leakage and NBTI delay, respectively of circuit c432. Figures 4.11–4.19 report simulated annealing plots for leakage and NBTI co-optimization of circuit c432. Figures 4.20 and 4.21 report the simulated annealing plots for leakage and NBTI delay, respectively of circuit c499. Figures 4.22–4.30 report simulated annealing plots for leakage and NBTI co-optimization of circuit c499. Figures 4.31 and 4.32 report the simulated annealing plots for leakage and NBTI delay, respectively of circuit c880. Figures 4.33–4.41 report simulated annealing plots for leakage and NBTI co-optimization of circuit c880. Figures 4.42 and
Table 4.1: Leakage and Delay Range with Simulated Annealing

<table>
<thead>
<tr>
<th>Design</th>
<th>NBTI Delay (ns)</th>
<th>NBTI Degradation (%)</th>
<th>Leakage (µW)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Max</td>
<td>Min</td>
<td>Max</td>
</tr>
<tr>
<td>c432</td>
<td>2.83</td>
<td>2.53</td>
<td>27.55</td>
</tr>
<tr>
<td>c499</td>
<td>2.47</td>
<td>2.17</td>
<td>40.90</td>
</tr>
<tr>
<td>c880</td>
<td>1.86</td>
<td>1.69</td>
<td>18.43</td>
</tr>
<tr>
<td>c1908</td>
<td>2.33</td>
<td>2.13</td>
<td>17.06</td>
</tr>
<tr>
<td>c2670</td>
<td>2.01</td>
<td>1.75</td>
<td>27.32</td>
</tr>
</tbody>
</table>

Table 4.2: MDV Leakage and MLV Delay

<table>
<thead>
<tr>
<th>Design</th>
<th>MLV Delay (ns)</th>
<th>MLV Degradation (%)</th>
<th>MDV Leakage (µW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>2.65</td>
<td>19.31</td>
<td>1.22</td>
</tr>
<tr>
<td>c499</td>
<td>2.43</td>
<td>38.6</td>
<td>1.85</td>
</tr>
<tr>
<td>c880</td>
<td>1.87</td>
<td>18.94</td>
<td>2.31</td>
</tr>
<tr>
<td>c1908</td>
<td>2.20</td>
<td>10.64</td>
<td>5.20</td>
</tr>
<tr>
<td>c2670</td>
<td>1.85</td>
<td>17.20</td>
<td>424.80</td>
</tr>
</tbody>
</table>

4.43 report the simulated annealing plots for leakage and NBTI delay, respectively of circuit c1908. Figures 4.44–4.52 report simulated annealing plots for leakage and NBTI co-optimization of circuit c1908. Figures 4.53 and 4.54 report the simulated annealing plots for leakage and NBTI delay, respectively of circuit c2670. Figures 4.55–4.63 report simulated annealing plots for leakage and NBTI co-optimization of circuit c2670.

Tables 4.1 and 4.2 report leakage, delay, and NBTI degradation numbers of simulated annealing on leakage and NBTI degradation. Columns 6 and 7 report minimum and maximum leakage from simulated annealing on leakage alone. Columns 2 and 3 report minimum and maximum NBTI delay from simulated annealing on NBTI. Columns 4 and 5 report corresponding NBTI degradation numbers.

Columns 2 and 3 of Table 4.2 report NBTI delay and degradation corresponding to minimum leakage vector. Column 4 of Table 4.2 reports leakage corresponding to minimum NBTI vector. We
Table 4.3: Leakage and NBTI Degradation - Co-optimized Vector

<table>
<thead>
<tr>
<th>Design</th>
<th>Delay (ns)</th>
<th>Co-optimized Vector</th>
<th>Leakage (µW)</th>
<th>Delay (ns)</th>
<th>Degradation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>2.22</td>
<td>1.04</td>
<td>2.62</td>
<td>17.90</td>
<td></td>
</tr>
<tr>
<td>c499</td>
<td>1.75</td>
<td>1.83</td>
<td>2.19</td>
<td>25.00</td>
<td></td>
</tr>
<tr>
<td>c880</td>
<td>1.57</td>
<td>2.33</td>
<td>1.73</td>
<td>9.90</td>
<td></td>
</tr>
<tr>
<td>c1908</td>
<td>1.99</td>
<td>5.08</td>
<td>2.16</td>
<td>8.3</td>
<td></td>
</tr>
<tr>
<td>c2670</td>
<td>1.58</td>
<td>325.80</td>
<td>1.86</td>
<td>17.60</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Leakage and NBTI Degradation - Vector Cycling

<table>
<thead>
<tr>
<th>Design</th>
<th>Co-optimized Vector with Back Tracking</th>
<th>Leakage (µW)</th>
<th>Delay (ns)</th>
<th>Degradation (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>1.09</td>
<td>2.47</td>
<td>11.30</td>
<td></td>
</tr>
<tr>
<td>c499</td>
<td>1.83</td>
<td>2.04</td>
<td>16.60</td>
<td></td>
</tr>
<tr>
<td>c880</td>
<td>2.35</td>
<td>1.67</td>
<td>6.20</td>
<td></td>
</tr>
<tr>
<td>c1908</td>
<td>5.11</td>
<td>2.11</td>
<td>6.00</td>
<td></td>
</tr>
<tr>
<td>c2670</td>
<td>350.7</td>
<td>1.77</td>
<td>11.80</td>
<td></td>
</tr>
</tbody>
</table>

observe that the delay corresponding to minimum leakage vector in column 2 of Table 4.2 is close to worst case delay in column 2 of Table 4.1. This observation reiterates the opposing effect of stacking on both phenomena and the need for NBTI and leakage co-optimization.

Table 4.3 reports leakage and NBTI degradation for co-optimized vector at primary input. Column 2 reports original delay of the circuit without NBTI degradation. Columns 3, 4, and 5 report leakage, NBTI delay and degradation over original delay when co-optimized vector is applied at primary inputs in idle mode. Columns 1, 2, and 3 of Table 4.4 report leakage, NBTI delay, and degradation with vector cycling.

Table 4.5 reports NBTI improvement and leakage overhead of vector cycling with co-optimized vector and alternative vector with respect to co-optimized vector alone. Column 2 reports the NBTI improvement and Column 3 reports leakage overhead incurred. Average of 5.3% NBTI improvement is achieved for 3.3% leakage overhead. In Table 4.6, Column 2 reports the NBTI im-
Table 4.5: Vector Cycling Compared to Co-optimized Vector Only. NBTI Improvement and Leakage Overhead presented.

<table>
<thead>
<tr>
<th>Design</th>
<th>Improvement (%)</th>
<th>Leakage Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>6.60</td>
<td>5.60</td>
</tr>
<tr>
<td>c499</td>
<td>8.40</td>
<td>2.20</td>
</tr>
<tr>
<td>c880</td>
<td>3.70</td>
<td>0.60</td>
</tr>
<tr>
<td>c1908</td>
<td>2.30</td>
<td>0.60</td>
</tr>
<tr>
<td>c2670</td>
<td>5.80</td>
<td>7.60</td>
</tr>
<tr>
<td>Average</td>
<td>5.30</td>
<td>3.30</td>
</tr>
</tbody>
</table>

Table 4.6: Vector Cycling Compared to MLV Only. NBTI Improvement and Leakage Overhead presented.

<table>
<thead>
<tr>
<th>Design</th>
<th>Improvement (%)</th>
<th>Leakage Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>8.01</td>
<td>14.10</td>
</tr>
<tr>
<td>c499</td>
<td>22.00</td>
<td>4.00</td>
</tr>
<tr>
<td>c880</td>
<td>12.74</td>
<td>13.70</td>
</tr>
<tr>
<td>c1908</td>
<td>4.64</td>
<td>4.70</td>
</tr>
<tr>
<td>c2670</td>
<td>5.40</td>
<td>53.70</td>
</tr>
<tr>
<td>Average</td>
<td>10.50</td>
<td>18.00</td>
</tr>
</tbody>
</table>

Improvement with respect to degradation corresponding to minimum leakage vector at primary inputs and column 3 reports leakage overhead incurred. Average of 10.5% NBTI improvement is achieved for 18% leakage overhead. In Table 4.7, column 2 and 3 report NBTI improvement and leakage overhead when compared to a circuit with minimum NBTI vector. An average of 2.13% NBTI improvement is obtained. The negative numbers in column 7 show that there is a leakage improvement instead of overhead. Minimum NBTI vector is not leakage friendly, so it has worse leakage when

Table 4.7: Vector Cycling Compared to MDV Only. NBTI Improvement and Leakage Overhead presented.

<table>
<thead>
<tr>
<th>Design</th>
<th>Improvement (%)</th>
<th>Leakage Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>2.52</td>
<td>-11.30</td>
</tr>
<tr>
<td>c499</td>
<td>7.40</td>
<td>-1.31</td>
</tr>
<tr>
<td>c880</td>
<td>1.04</td>
<td>1.53</td>
</tr>
<tr>
<td>c1908</td>
<td>0.87</td>
<td>-1.64</td>
</tr>
<tr>
<td>c2670</td>
<td>-1.18</td>
<td>-21.12</td>
</tr>
<tr>
<td>Average</td>
<td>2.13</td>
<td>-6.77</td>
</tr>
</tbody>
</table>
compared to co-optimized vector with back tracking. An average of 6.77% leakage improvement is obtained. An improvement over minimum NBTI vector reinforces the NBTI optimization solely attributed to back tracking algorithm. The ratio of active to stand by time is set to be 1:9. In all simulated annealing experiments, five simulations are run and the best value is chosen.

Figures 4.64–4.68 report NBTI delay and leakage variation when weight of each phenomenon is controlled during the co-optimization. X axis shows the optimization weight on NBTI phenomenon while Y axis shows the leakage and NBTI degradation. The total weight of NBTI and leakage phenomena is always 100%. NBTI and leakage effects depend on the transistor organization of the gates. If all gate inputs are 1 in the case of NOR gates, both NBTI and leakage are minimized. In case of NAND gates, NBTI is minimized, but leakage worsens. If all gates are NAND, as NBTI weight increases, a decrease in NBTI degradation and an increase in leakage is expected. However, the circuits consist of different gate types with varying stack effect. As a result, the minimum cost at each NBTI weight fluctuates.

4.7 Chapter Summary

In this chapter, vector cycling approach is proposed to co-optimize NBTI and leakage in gate level combinational circuits. Simulated annealing is the primary optimization heuristic employed for co-optimization. A back tracking algorithm is introduced to obtain two efficient vectors to be used in vector cycling such that disjoint set of transistors are stressed. Experimental results are reported and substantial NBTI optimization is obtained along with the leakage reduction.
Figure 4.9: Simulated Annealing for Leakage - c432

Figure 4.10: Simulated Annealing for NBTI Delay - c432

Figure 4.11: Simulated Annealing for 50% Leakage and 50% NBTI Delay - c432
Figure 4.12: Simulated Annealing for 10% Leakage and 90% NBTI Delay - c432

Figure 4.13: Simulated Annealing for 20% Leakage and 80% NBTI Delay - c432

Figure 4.14: Simulated Annealing for 30% Leakage and 70% NBTI Delay - c432

Figure 4.15: Simulated Annealing for 40% Leakage and 60% NBTI Delay - c432
Figure 4.16: Simulated Annealing for 60% Leakage and 40% NBTI Delay - c432

Figure 4.17: Simulated Annealing for 70% Leakage and 30% NBTI Delay - c432

Figure 4.18: Simulated Annealing for 80% Leakage and 20% NBTI Delay - c432

Figure 4.19: Simulated Annealing for 90% Leakage and 10% NBTI Delay - c432
Figure 4.20: Simulated Annealing for Leakage - c499

Figure 4.21: Simulated Annealing for NBTI Delay - c499

Figure 4.22: Simulated Annealing for 50% Leakage and 50% NBTI Delay - c499
Figure 4.23: Simulated Annealing for 10% Leakage and 90% NBTI Delay - c499

Figure 4.24: Simulated Annealing for 20% Leakage and 80% NBTI Delay - c499

Figure 4.25: Simulated Annealing for 30% Leakage and 70% NBTI Delay - c499

Figure 4.26: Simulated Annealing for 40% Leakage and 60% NBTI Delay - c499
Figure 4.27: Simulated Annealing for 60% Leakage and 40% NBTI Delay - c499

Figure 4.28: Simulated Annealing for 70% Leakage and 30% NBTI Delay - c499

Figure 4.29: Simulated Annealing for 80% Leakage and 20% NBTI Delay - c499

Figure 4.30: Simulated Annealing for 90% Leakage and 10% NBTI Delay - c499
Figure 4.31: Simulated Annealing for Leakage - c880

Figure 4.32: Simulated Annealing for NBTI Delay - c880

Figure 4.33: Simulated Annealing for 50% Leakage and 50% NBTI Delay - c880
Figure 4.34: Simulated Annealing for 10% Leakage and 90% NBTI Delay - c880

Figure 4.35: Simulated Annealing for 20% Leakage and 80% NBTI Delay - c880

Figure 4.36: Simulated Annealing for 30% Leakage and 70% NBTI Delay - c880

Figure 4.37: Simulated Annealing for 40% Leakage and 60% NBTI Delay - c880
Figure 4.38: Simulated Annealing for 60% Leakage and 40% NBTI Delay - c880

Figure 4.39: Simulated Annealing for 70% Leakage and 30% NBTI Delay - c880

Figure 4.40: Simulated Annealing for 80% Leakage and 20% NBTI Delay - c880

Figure 4.41: Simulated Annealing for 90% Leakage and 10% NBTI Delay - c880
Figure 4.42: Simulated Annealing for Leakage - c1908

Figure 4.43: Simulated Annealing for NBTI Delay - c1908

Figure 4.44: Simulated Annealing for 50% Leakage and 50% NBTI Delay - c1908
Figure 4.45: Simulated Annealing for 10% Leakage and 90% NBTI Delay - c1908

Figure 4.46: Simulated Annealing for 20% Leakage and 80% NBTI Delay - c1908

Figure 4.47: Simulated Annealing for 30% Leakage and 70% NBTI Delay - c1908

Figure 4.48: Simulated Annealing for 40% Leakage and 60% NBTI Delay - c1908
Figure 4.49: Simulated Annealing for 60% Leakage and 40% NBTI Delay - c1908

Figure 4.50: Simulated Annealing for 70% Leakage and 30% NBTI Delay - c1908

Figure 4.51: Simulated Annealing for 80% Leakage and 20% NBTI Delay - c1908

Figure 4.52: Simulated Annealing for 90% Leakage and 10% NBTI Delay - c1908
Figure 4.53: Simulated Annealing for Leakage - c2670

Figure 4.54: Simulated Annealing for NBTI Delay - c2670

Figure 4.55: Simulated Annealing for 50% Leakage and 50% NBTI Delay - c2670
Figure 4.56: Simulated Annealing for 10% Leakage and 90% NBTI Delay - c2670

Figure 4.57: Simulated Annealing for 20% Leakage and 80% NBTI Delay - c2670

Figure 4.58: Simulated Annealing for 30% Leakage and 70% NBTI Delay - c2670

Figure 4.59: Simulated Annealing for 40% Leakage and 60% NBTI Delay - c2670
Figure 4.60: Simulated Annealing for 60% Leakage and 40% NBTI Delay - c2670

Figure 4.61: Simulated Annealing for 70% Leakage and 30% NBTI Delay - c2670

Figure 4.62: Simulated Annealing for 80% Leakage and 20% NBTI Delay - c2670

Figure 4.63: Simulated Annealing for 90% Leakage and 10% NBTI Delay - c2670
Figure 4.64: NBTI Delay and Leakage with Varying Weights - c432

Figure 4.65: NBTI Delay and Leakage with Varying Weights - c499
Figure 4.66: NBTI Delay and Leakage with Varying Weights - c880

Figure 4.67: NBTI Delay and Leakage with Varying Weights - c1908

96
Figure 4.68: NBTI Delay and Leakage with Varying Weights - c2670
CHAPTER 5 : STATE ENCODING BASED NBTI OPTIMIZATION IN FINITE STATE MACHINES

NBTI degradation is a serious problem in FSM circuits as it might lead to several timing closure issues. It can be controlled using proposed state encoding. In this chapter, we introduce a state encoding based NBTI optimization technique for FSMs and report experimental results.

5.1 Proposed Approach

We propose a run time NBTI optimization technique for finite state machines. FSMs operate in various states at different points of time. Output of state registers drive a number of PMOS transistors in a circuit, some of which might be on the critical path. The number of critical path transistors under stress depends on state code. The extent of NBTI degradation in these transistors is controlled by state probability. For a given application, states can be sorted based on how long the FSM works in specific states, or based on state probability. Since NBTI depends on the ON time of PMOS transistors, high probability states have worse NBTI degradation. The transistors that are ON in a high probability state are under stress for a longer period of time, and hence more prone to NBTI degradation. High probability states have a great scope to influence the NBTI degradation of the circuit. Given a previous state code of a circuit, one can determine the state code of current state with minimum NBTI effect. Such optimized state codes should be assigned.

3This chapter is a part of S. Pendyala and S. Katkoori, “State Encoding based NBTI optimization of Finite State Machines,” Manuscript submitted to 2016 17th International Symposium on Quality of Electronic Design (ISQED).
to the high probability states so that overall NBTI degradation of the circuit can be minimized. State codes of low probability states influence NBTI degradation as well. But given their shorter duration, their effect on degradation is much lesser when compared to the high probability states.

The relation between ON time of transistors and NBTI degradation is not straightforward. It can be found in the NBTI model [22] used in this paper. Hence, a heuristic approach is used instead of an analytical approach to achieve NBTI optimization. The technique used in this work could easily be extended to counter PBTI effect as well. A compact model for PBTI DC stress can be developed in a similar way as NBTI [22]. According to Lin et al. [62] and Wang et al. [61], PBTI is much lower than NBTI in 45nm technology, so NBTI effect is solely considered. PBTI is comparable to NBTI in high-k metal process technology. We would like to include PBTI effect in our future work with lower technology nodes. In conventional synthesis, reducing dynamic power of FSM is the major concern. In synthesis for NBTI minimization, delay degradation is optimized at the risk of slight dynamic power overhead.

5.1.1 Motivational Example 1

Consider an example FSM benchmark circuit sample.kiss2 as shown in Table 5.1 with two states $S_0$ and $S_1$. Figures 5.1 and 5.2 show the critical paths with different state codes for $S_0$ and $S_1$. In Figure 5.1, the critical path consists of 6 PMOS transistors, with 2 PMOS in FF2 and 1
PMOS each in g1, g2, g18, and FF1. In Figure 5.2, the critical path consists of 4 PMOS transistors, with 3 PMOS in FF1 and 1 PMOS in g11. This example illustrates how PMOS transistors on the critical path vary with state encoding.

Figure 5.1: Sample FSM with $S_0 = 0, S_1 = 1$

Figure 5.2: Sample FSM with $S_0 = 1, S_1 = 0$
5.1.2 Motivational Example 2

Consider an FSM benchmark circuit mc.kiss2 from LGSYNTH93 benchmark suite as shown in Table 5.2. It has 4 states (\(S_1, S_2, S_3, S_4\)), 3 bit input, and 5 bit output. For a sample input sequence, \(S_3\) and \(S_4\) are states with high probability. State codes for \(S_3\) and \(S_4\) should be chosen to achieve minimized NBTI optimization. The initial state codes for \(S_3\) and \(S_4\) are 10 and 11, respectively. The number of PMOS transistors on the critical path is six. By applying the proposed NBTI optimization technique, final state codes for \(S_3\) and \(S_4\) are 01 and 00, respectively. The number of PMOS transistors on the critical path is reduced to four and 4% NBTI optimization is achieved using optimized codes. Worst case state code assignment is \(S_3=00\) and \(S_4=10\) and it results in 24% NBTI degradation. This example demonstrates the influence of state code assignment on NBTI degradation.

Table 5.2: MC.kiss2 State Transition Table

<table>
<thead>
<tr>
<th>Input</th>
<th>Current state</th>
<th>Next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-</td>
<td>S1</td>
<td>S1</td>
<td>0010</td>
</tr>
<tr>
<td>-0-</td>
<td>S1</td>
<td>S1</td>
<td>00010</td>
</tr>
<tr>
<td>11-</td>
<td>S1</td>
<td>S2</td>
<td>10010</td>
</tr>
<tr>
<td>-0</td>
<td>S2</td>
<td>S2</td>
<td>00110</td>
</tr>
<tr>
<td>-1</td>
<td>S2</td>
<td>S3</td>
<td>10110</td>
</tr>
<tr>
<td>10-</td>
<td>S3</td>
<td>S3</td>
<td>01000</td>
</tr>
<tr>
<td>0-</td>
<td>S3</td>
<td>S4</td>
<td>11000</td>
</tr>
<tr>
<td>-1</td>
<td>S3</td>
<td>S4</td>
<td>11000</td>
</tr>
<tr>
<td>-0</td>
<td>S4</td>
<td>S4</td>
<td>01001</td>
</tr>
<tr>
<td>-1</td>
<td>S4</td>
<td>S1</td>
<td>11001</td>
</tr>
</tbody>
</table>

Figure 5.3 shows the implementation framework. For a given FSM benchmark in .kiss2 format, an in-house FSM to behavioral verilog converter is developed. Cadence BUILDGATES is used to do gate level synthesis. The gate level verilog netlist is converted into SPICE format using Synopsys NETTRAN tool. An in-house NBTI degradation tool is developed to achieve \(V_{th}\)
Figure 5.3: Framework for NBTI Optimization
Algorithm Simulated_Annealing_NBTI

2 Inputs: (a) High Probability States \((H)\)
3 (b) Used State codes \((U)\)
4 (c) Unused State codes \((UU)\)
5 (d) \(algo\_type \in \text{generic, state\_prob\_based}\)
6 Outputs: Optimized State codes \((NBTI\_codes)\)
7
8 \begin{verbatim}
9 Temp \leftarrow 100
10 \gamma \leftarrow \text{annealing co-efficient}
11 \text{best\_codes} \leftarrow U
12 \text{curr\_codes} \leftarrow \text{best\_codes}
13 \text{best\_delay} \leftarrow NBTI\_delay(\text{curr\_codes})
14 \textbf{while} Temp > 0 \textbf{do}
15 \hspace{1em} \textbf{foreach} iteration in \text{iteration\_count} \textbf{do}
16 \hspace{2em} \textbf{if} algo\_type == \text{state\_prob\_based} \textbf{then}
17 \hspace{3em} window \leftarrow Temp \ast \text{no\_of\_high\_states}/2
18 \hspace{3em} \text{curr\_codes} \leftarrow \text{Swap(random}(H) \text{ in the window,}
19 \hspace{3em} \text{random}(U) \text{ or random}(UU))
20 \hspace{2em} \textbf{else}
21 \hspace{3em} window \leftarrow Temp \ast \text{no\_of\_used\_states}/2
22 \hspace{3em} \text{curr\_codes} \leftarrow \text{Swap(random}(U) \text{ in the window,}
23 \hspace{3em} \text{random}(UU))
24 \hspace{1em} \textbf{end if}
25 \hspace{1em} \text{curr\_delay} \leftarrow NBTI\_delay(\text{curr\_codes})
26 \hspace{1em} \textbf{if} curr\_delay < best\_delay \parallel \text{random}(0,1) \leq \text{pow(e, -(best\_delay - curr\_delay)/Temp)} \textbf{then}
27 \hspace{2em} \textbf{if} curr\_delay < best\_delay \textbf{then}
28 \hspace{3em} NBTI\_codes \leftarrow \text{curr\_codes}
29 \hspace{2em} \textbf{end if}
30 \hspace{2em} \text{best\_codes} \leftarrow \text{curr\_codes}
31 \hspace{2em} \text{best\_delay} \leftarrow NBTI\_delay(\text{curr\_codes})
32 \hspace{1em} \textbf{end if}
33 \hspace{1em} Temp \leftarrow Temp \ast \gamma
34 \textbf{end while}
35 \text{best\_delay} \leftarrow NBTI\_delay(\text{NBTI\_codes})
36 \end{verbatim}
8
9 Figure 5.4: Simulated Annealing Algorithm for NBTI Optimization
degradation models for all degraded transistors in the SPICE circuit. This tool used NBTI long term degradation (DC stress) model for random input sequence [22] to calculate Vth degradation of the PMOS transistors. Activity factor is one of the inputs to the NBTI degradation model. For any given input sequence, activity factor for all PMOS transistors is derived using Synopsys VCS and Synopsys PRIMETIME tools. NBTI degraded SPICE circuit is developed from degraded PMOS transistor models using an in-house script. Synopsys NANOTIME static timing analysis tool is used to measure degraded delay from NBTI degraded SPICE circuit from which delay degradation is calculated.

Time complexity of brute force search on the NBTI aware state encoding problem is \(O(t^u)\), where \(t\) represents total possible state codes and \(u\) represents the number of states required in the FSM. The time complexity to find an optimal solution is exponential. Hence, we adopt heuristic technique to find NBTI optimized state encoding. Figure 5.4 shows the simulated annealing (SA) algorithms for NBTI optimization. The two simulated annealing algorithms are implemented: (a) Generic SA; (b) State probability based SA.

Generic SA technique considers of all the used states for optimization. For example, say an FSM has 3 bit state encoding and only five states (S1, S2, S3, S4, S5). Since state encoding has three bits, there are total of eight state codes to choose from. Initial state assignment as given in the benchmark circuit consists of five states with state codes (S1=000, S2=001, S3=010, S4=011, S5=100). The five states of FSM are sorted based on duration of active time during total lifetime (10 years). Let us assume the states in the order of high to low state probability are S3, S1, S5, S2, S4. In the NBTI optimization algorithm, first half of the states are chosen as high probability states (S3, S1).
Inputs of the algorithm on line 2 include used state codes \((U)\), unused state codes \((UU)\), high probability states \((H)\), complete state set \((S)\), and \(high\_state\_flag\). For above example, \(U \rightarrow (000, 001, 010, 011, 100)\), \(UU \rightarrow (101, 110, 111)\), \(H \rightarrow (S3, S1)\), \(algo\_type\) is \(state\_prob\_based\) for state probability based SA and \(generic\) for generic SA. Output is the Optimized state code set \((NBTI\_codes)\) on line 6. Temperature parameter of simulated annealing is set to 100 on line 8. Simulated annealing coefficient \(\gamma\) on line 9 is a variable that is a function of total simulation time as defined in 4.2. In every SA iteration, a state in a window is chosen and its corresponding state code is swapped with a state code outside the window as shown in lines 18, 19, 22, and 23.

In case of generic SA, the window consists of complete state set \(S\). Window for state probability based SA consists of high probability states \(H\). Depending on \(algo\_type\) in line 15, window size and swap operation are executed. The size of the window varies with temperature parameter of simulated annealing. Figure 5.5 shows the window range during SA iterations. In generic SA, window size starts with complete state set \(S\) and gradually reduces at low temperatures.

In state probability based SA, window size starts with high probability states \(H\) and gradually reduces as well. Simulated annealing condition is tested in lines 26 and 27. Temperature parameter is updated in line 36. Optimized state codes are assigned to \(NBTI\_codes\). Cost function is the NBTI delay degradation \((NBTI\_delay)\), and is used to calculate cost of a given solution in lines 12, 25, 34, and 37. Depending on the simulation time that can be afforded, quality of the solution improves. If the probability numbers of high probability states are large, state probability based SA tend to achieve a better quality solution since the input space is pruned and limited to high probability states. This is because high probability states cause major stress due to long ON time. The scope of NBTI minimization by assigning optimized state code is substantial. The
simulated annealing can also be extended to NBTI and leakage co-optimization by changing the
cost function to include leakage as well.

<table>
<thead>
<tr>
<th>Design</th>
<th>Input (bits)</th>
<th>Output (bits)</th>
<th>Number of states</th>
<th>Number of state table entries</th>
<th>Simulation time 1 SA run (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s8</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>20</td>
<td>67</td>
</tr>
<tr>
<td>ex6</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>34</td>
<td>77</td>
</tr>
<tr>
<td>dk17</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td>32</td>
<td>85</td>
</tr>
<tr>
<td>s386</td>
<td>7</td>
<td>7</td>
<td>13</td>
<td>64</td>
<td>94</td>
</tr>
<tr>
<td>s1</td>
<td>8</td>
<td>6</td>
<td>20</td>
<td>107</td>
<td>176</td>
</tr>
<tr>
<td>s1494</td>
<td>8</td>
<td>19</td>
<td>48</td>
<td>250</td>
<td>1500</td>
</tr>
<tr>
<td>s1488</td>
<td>8</td>
<td>19</td>
<td>48</td>
<td>251</td>
<td>1500</td>
</tr>
<tr>
<td>s208</td>
<td>11</td>
<td>2</td>
<td>18</td>
<td>153</td>
<td>180</td>
</tr>
</tbody>
</table>

We report the experimental results for two FSM NBTI optimization techniques: (a) Generic SA; and (b) State probability based SA. Eight FSM benchmarks from LGSYNTH93 suite are used in 45nm technology node at temperature 105°C. Relevant data about the benchmark circuits is summarized in Table 5.3. Total optimization time for each benchmark is five hours except for s1488 and s1494 which takes eight hours each. It can be varied by the user to determine the
solution quality. The Verilog-A code of the NBTI degradation model is verified with silicon results from [94]. In the framework, for each set of state codes, the circuit undergoes gate level synthesis. This might lead to different circuit with more area than the circuit with initial state encoding. Hamming distance is also changed when state encoding is modified. Re-synthesis might cause area overhead and change in hamming distance might cause dynamic power overhead. Area and power overheads are reported along with NBTI optimization for the FSM benchmarks.

Time taken for generation of SPICE level circuit and net probabilities takes an average of 70 seconds. Time taken to generate degraded circuit from in-house Vth degradation calculation tool depends on the size of the benchmark. This tool occupies a large part of simulation time. We experimented with Cadence Relxpert but did not achieve correct results. With a quality industry standard tool, the time taken by Vth degradation calculation can be reduced in the future. This can make the technique scalable to larger designs. Figures 5.6–5.21 show the cost versus temperature plots for generic and state probability based optimization for eight benchmarks. The best solution so far is the cost on y-axis.

Table 5.4 reports NBTI improvement obtained using state probability based simulated annealing along with area and power overheads. NBTI improvement is reported in column 2. In columns 3, 4, and 5, area, dynamic power, and leakage power overheads are reported, respectively. Several overhead numbers are observed to be negative (i.e. improvement). For instance, NBTI optimization on ex6 showed an improvement in NBTI degradation, area, leakage and dynamic power. For the eight benchmarks, an average of 18.8% NBTI improvement. It also achieved an average area improvement of 5.5%, dynamic power improvement of 4.6% and leakage improvement of 4.1%. Hence, the overhead is optimized. In Tables 5.4 and 5.5, NBTI improvement reported is mea-
sured between NBTI degradation of optimized state encoding and NBTI degradation of initial state encoding provided in the benchmarks.

Table 5.5 reports NBTI improvement obtained using generic simulated annealing along with area and power overhead. NBTI improvement is reported in column 2 and area, dynamic power and leakage power overheads are reported in columns 3, 4 and 5, respectively. Several overhead numbers are observed to be negative in this technique as well which means that there is an improvement instead of overhead incurred. An average of 17.5% NBTI improvement is reported for an average area overhead of 3.3%, dynamic power overhead of 9.6% and leakage overhead of 2.0%.

The experimental results of both NBTI optimization techniques achieve substantial NBTI improvement. However, further improvement in area, leakage and dynamic power parameters overhead is observed in case of state probability based simulated annealing.

5.3 Chapter Summary

In this chapter, a framework for state encoding based NBTI optimization was detailed. State probability based and generic optimization techniques were implemented. Experimental results were reported for both of the techniques, and NBTI improvement looks promising.
Figure 5.6: Generic Simulated Annealing for NBTI Delay - s8

Figure 5.7: Generic Simulated Annealing for NBTI Delay - ex6

Figure 5.8: Generic Simulated Annealing for NBTI Delay - dk17
Figure 5.9: Generic Simulated Annealing for NBTI Delay - s386

Figure 5.10: Generic Simulated Annealing for NBTI Delay - s1

Figure 5.11: Generic Simulated Annealing for NBTI Delay - s1494
Figure 5.12: Generic Simulated Annealing for NBTI Delay - s1488

Figure 5.13: Generic Simulated Annealing for NBTI Delay - s208

Figure 5.14: State Probability Based Simulated Annealing for NBTI Delay - s8
Figure 5.15: State Probability Based Simulated Annealing for NBTI Delay - ex6

Figure 5.16: State Probability Based Simulated Annealing for NBTI Delay - dk17

Figure 5.17: State Probability Based Simulated Annealing for NBTI Delay - s386
Figure 5.18: State Probability Based Simulated Annealing for NBTI Delay - s1

Figure 5.19: State Probability Based Simulated Annealing for NBTI Delay - s1494

Figure 5.20: State Probability Based Simulated Annealing for NBTI Delay - s1488
Figure 5.21: State Probability Based Simulated Annealing for NBTI Delay - s208

Table 5.4: Experimental Results - State Probability Based Simulated Annealing

<table>
<thead>
<tr>
<th>Design</th>
<th>NBTI Improvement (%)</th>
<th>Area (%)</th>
<th>Dynamic Power (%)</th>
<th>Leakage Power (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s8</td>
<td>10.3</td>
<td>5.4</td>
<td>28.0</td>
<td>10.3</td>
</tr>
<tr>
<td>ex6</td>
<td>37.9</td>
<td>-2.0</td>
<td>-3.9</td>
<td>-1.3</td>
</tr>
<tr>
<td>dk17</td>
<td>3.8</td>
<td>-1.9</td>
<td>-37.0</td>
<td>-13.2</td>
</tr>
<tr>
<td>s386</td>
<td>14.5</td>
<td>-2.8</td>
<td>-1.3</td>
<td>-8.1</td>
</tr>
<tr>
<td>s1</td>
<td>23.1</td>
<td>7.7</td>
<td>5.0</td>
<td>8.4</td>
</tr>
<tr>
<td>s1494</td>
<td>20.8</td>
<td>0.9</td>
<td>0.8</td>
<td>3.7</td>
</tr>
<tr>
<td>s1488</td>
<td>9.6</td>
<td>-23.2</td>
<td>-0.2</td>
<td>-2.7</td>
</tr>
<tr>
<td>s208</td>
<td>30.8</td>
<td>-28.5</td>
<td>-28.0</td>
<td>-30.0</td>
</tr>
<tr>
<td>Average</td>
<td>18.8</td>
<td>-5.5</td>
<td>-4.6</td>
<td>-4.1</td>
</tr>
</tbody>
</table>

Table 5.5: Experimental Results - Generic Simulated Annealing

<table>
<thead>
<tr>
<th>Design</th>
<th>NBTI Improvement (%)</th>
<th>Area (%)</th>
<th>Dynamic Power (%)</th>
<th>Leakage Power (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s8</td>
<td>3.8</td>
<td>4.6</td>
<td>19.4</td>
<td>8.6</td>
</tr>
<tr>
<td>ex6</td>
<td>35.0</td>
<td>2.2</td>
<td>1.4</td>
<td>4.0</td>
</tr>
<tr>
<td>dk17</td>
<td>0.4</td>
<td>3.2</td>
<td>23.8</td>
<td>-5.0</td>
</tr>
<tr>
<td>s386</td>
<td>19.1</td>
<td>1.2</td>
<td>15.7</td>
<td>-3.3</td>
</tr>
<tr>
<td>s1</td>
<td>30.8</td>
<td>11.3</td>
<td>9.2</td>
<td>12.4</td>
</tr>
<tr>
<td>s1494</td>
<td>13.4</td>
<td>-0.1</td>
<td>-2.4</td>
<td>-0.3</td>
</tr>
<tr>
<td>s1488</td>
<td>16.6</td>
<td>-2.4</td>
<td>-0.5</td>
<td>-1.0</td>
</tr>
<tr>
<td>s208</td>
<td>21.0</td>
<td>-3.2</td>
<td>10.1</td>
<td>0.4</td>
</tr>
<tr>
<td>Average</td>
<td>17.5</td>
<td>3.3</td>
<td>9.6</td>
<td>2.0</td>
</tr>
</tbody>
</table>
CHAPTER 6 : CONCLUSIONS AND FUTURE WORK

Leakage reduction is a crucial problem in deep sub micron regime. In chapter 3, we have formulated the low leakage vector identification technique based on interval propagation at register transfer level. Using this technique, significant subthreshold leakage savings with no area overhead are achieved successfully. We also proposed a self similarity based module characterization which makes this technique scalable to more complex datapaths. We conclude that our low leakage vector determination technique achieves significant leakage optimization in RTL datapaths of any level of complexity if one could invest in a few hours of pre-characterization time. As part of future work, a correlation between a circuit and self similarity property of the input space could be derived. The origin of self similarity property in VLSI circuits can be examined based on the structure of the circuits. The difference of self similarity in bit sliced designs and random designs can be researched. Interval propagation technique can be adopted to identify minimum NBTI vector to achieve NBTI minimization at high level. Input design space exploration can be conducted with self similarity based characterization to acquire NBTI and leakage co-optimized vector.

Input vector control is a common leakage reduction technique which reduces the leakage by controlling transistor stacking with a given input vector. Since NBTI is effected by transistor stacking as well, co-optimization of both the objectives is considered. In chapter 4, an alternative vector cycling technique is developed with simulated annealing and a back tracking algorithm for NBTI and leakage co-optimization. The technique is implemented in gate level combinational
circuits. The periodic recovery for critical path transistors under stress alleviates NBTI degradation. This technique successfully achieved significant NBTI improvement with minimum leakage overhead. We conclude that our technique optimizes NBTI and leakage effects on gate level circuits. The technique can be further extended to high-k circuits where PBTI phenomenon is dominant as well. NBTI, PBTI and leakage can be co-optimized.

State encoding in FSMs has a strong control on circuit area and performance. It can be adopted to optimize NBTI degradation as well. In chapter 5, a simulated annealing algorithm is devised to obtain NBTI optimization in finite state machines at gate level. State probability parameter is used to reduce the input space. Significant NBTI improvement is achieved for minimum area and power overhead. The same technique can be extended to PBTI optimization as well. As a part of future work, NBTI aware synthesis as described in [48] can be performed on the benchmarks in addition to NBTI optimization. This might alleviate NBTI further.
REFERENCES


APPENDIX A: SELF SIMILARITY PLOTS FOR 8b ADDER AND 8b MULTIPLIER

Figures A.1–A.5 present the Hurst parameters and grid level leakage distribution of 8b multiplier when the input space is divided into cells along both axes (2 x 2 grid). Figures A.6–A.10 present the Hurst parameters and grid level leakage distribution of 8b adder when the input space is divided into cells along both axes (2 x 2 grid).

If the same input space is divided into 4 grid cells along one axis, we have the following grid cells: [0, 0 - 63, 255], [63, 0 - 127, 255], [127, 0 - 191, 255], and [191, 0 - 255, 255]. Figures A.11–A.15 present the grid level distribution of a multiplier when the input space is divided along one axis (4 x 1 grid). Figures A.16–A.20 present the grid level distribution of a multiplier when the input space is divided along one axis (4 x 1 grid). Figures A.21–A.37 present the grid level distribution of 8b multiplier when the input space is divided along both axes (4 x 4 grid). Figures A.38–A.54 present the grid level distribution of 8b adder when the input space is divided along both axes (4 x 4 grid). If the bit-width of the module is doubled, the number of cells on each axis is doubled as well. Hence, the characterization is scalable.

Figure A.1: Hurst Parameter in Multiplier with 4 Cells (2 x 2 Grid)
Figure A.2: Multiplier Distribution in Grid Cell 1 (2 x 2 Grid)

Figure A.3: Multiplier Distribution in Grid Cell 2 (2 x 2 Grid)
Figure A.4: Multiplier Distribution in Grid Cell 3 (2 x 2 Grid)

Figure A.5: Multiplier Distribution in Grid Cell 4 (2 x 2 Grid)
Figure A.6: Hurst Parameter in Adder with 4 Cells (2 x 2 Grid)

Figure A.7: Adder Distribution in Grid Cell 1 (2 x 2 Grid)
Figure A.8: Adder Distribution in Grid Cell 2 (2 x 2 Grid)

Figure A.9: Adder Distribution in Grid Cell 3 (2 x 2 Grid)
Figure A.10: Adder Distribution in Grid Cell 4 (2 x 2 Grid)

Figure A.11: Hurst Parameter in Multiplier with 4 Cells (4 x 1 Grid)
Figure A.12: Multiplier Distribution in Grid Cell 1 (4 x 1 Grid)

Figure A.13: Multiplier Distribution in Grid Cell 2 (4 x 1 Grid)
Figure A.14: Multiplier Distribution in Grid Cell 3 (4 x 1 Grid)

Figure A.15: Multiplier Distribution in Grid Cell 4 (4 x 1 Grid)
Figure A.16: Hurst Parameter in Adder with 4 Cells (4 x 1 Grid)

Figure A.17: Adder Distribution in Grid Cell 1 (4 x 1 Grid)
Figure A.18: Adder Distribution in Grid Cell 2 (4 x 1 Grid)
Figure A.19: Adder Distribution in Grid Cell 3 (4 x 1 Grid)

Figure A.20: Adder Distribution in Grid Cell 4 (4 x 1 Grid)
Figure A.21: Hurst Parameter in Multiplier with 16 Cells (4 x 4 Grid)

\[
\begin{array}{cccc}
0.70 & 0.71 & 0.70 & 0.71 \\
0.77 & 0.73 & 0.71 & 0.69 \\
0.75 & 0.71 & 0.68 & 0.67 \\
0.81 & 0.81 & 0.81 & 0.82 \\
\end{array}
\]

Figure A.22: Multiplier Distribution in Grid Cell 1 (4 x 4 Grid)
Figure A.23: Multiplier Distribution in Grid Cell 2 (4 x 4 Grid)

Figure A.24: Multiplier Distribution in Grid Cell 3 (4 x 4 Grid)
Figure A.25: Multiplier Distribution in Grid Cell 4 (4 x 4 Grid)

Figure A.26: Multiplier Distribution in Grid Cell 5 (4 x 4 Grid)
Figure A.27: Multiplier Distribution in Grid Cell 6 (4 x 4 Grid)

Figure A.28: Multiplier Distribution in Grid Cell 7 (4 x 4 Grid)
Figure A.29: Multiplier Distribution in Grid Cell 8 (4 x 4 Grid)

Figure A.30: Multiplier Distribution in Grid Cell 9 (4 x 4 Grid)
Figure A.31: Multiplier Distribution in Grid Cell 10 (4 x 4 Grid)

Figure A.32: Multiplier Distribution in Grid Cell 11 (4 x 4 Grid)
Figure A.33: Multiplier Distribution in Grid Cell 12 (4 x 4 Grid)

Figure A.34: Multiplier Distribution in Grid Cell 13 (4 x 4 Grid)
Figure A.35: Multiplier Distribution in Grid Cell 14 (4 x 4 Grid)

Figure A.36: Multiplier Distribution in Grid Cell 15 (4 x 4 Grid)
Figure A.37: Multiplier Distribution in Grid Cell 16 (4 x 4 Grid)

Figure A.38: Hurst Parameter in Adder with 16 Cells (4 x 4 Grid)
Figure A.39: Adder Distribution in Grid Cell 1 (4 x 4 Grid)

Figure A.40: Adder Distribution in Grid Cell 2 (4 x 4 Grid)
Figure A.41: Adder Distribution in Grid Cell 3 (4 x 4 Grid)

Figure A.42: Adder Distribution in Grid Cell 4 (4 x 4 Grid)
Figure A.43: Adder Distribution in Grid Cell 5 (4 x 4 Grid)

Figure A.44: Adder Distribution in Grid Cell 6 (4 x 4 Grid)
Figure A.45: Adder Distribution in Grid Cell 7 (4 x 4 Grid)

Figure A.46: Adder Distribution in Grid Cell 8 (4 x 4 Grid)
Figure A.47: Adder Distribution in Grid Cell 9 (4 x 4 Grid)

Figure A.48: Adder Distribution in Grid Cell 10 (4 x 4 Grid)
Figure A.49: Adder Distribution in Grid Cell 11 (4 x 4 Grid)

Figure A.50: Adder Distribution in Grid Cell 12 (4 x 4 Grid)
Figure A.51: Adder Distribution in Grid Cell 13 (4 x 4 Grid)

Figure A.52: Adder Distribution in Grid Cell 14 (4 x 4 Grid)
Figure A.53: Adder Distribution in Grid Cell 15 (4 x 4 Grid)
Figure A.54: Adder Distribution in Grid Cell 16 (4 x 4 Grid)
APPENDIX B: COPYRIGHT CLEARANCE FORMS

Below is permission for the use of Figure 1.1.
Below are the permissions for the use of material in Chapter 3.

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any section (e.g., figures, tables, text, or textual material) of an IEEE copyrighted paper in a thesis:

1. In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © (year of original publication) IEEE. (Please note: Users cannot make any removals of, or additions to, IEEE copyright lines from any printed copy of an IEEE copyrighted paper. Users can, however, make any removals of, or additions to, IEEE copyright lines from any electronic copy of an IEEE copyrighted paper when posting the paper or your thesis online.)

2. The following IEEE copyright notice should be placed prominently in the references: © (year of original publication) IEEE, reprinted, with permission, from (author names, paper title, IEEE publication title, and month/year of publication).

3. If an electronic copy of an IEEE copyrighted paper can be used when posting the paper or your thesis online.

4. In sharing the thesis on the author’s university website, please display the following message in a prominent place on the website: “Reference to IEEE copyrighted material is used with permission. Further use of this material is permitted, if referenced in...”
This is a License Agreement between Shilpa Pendyala ("You") and Springer ("Springer") provided by Copyright Clearance Center ("CCC"). The license consists of your order details, the terms and conditions provided by Springer, and the payment terms and conditions.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number 3746360332770
License date Nov 12, 2015
Licensed content publisher Springer
Licensed content publication Springer eBook
Licensed content title Interval Arithmetic and Self Similarity Based Subthreshold Leakage Optimization in RTL Datapaths
Licensed content author Shilpa Pendyala
Licensed content date Jan 1, 2015
Type of Use Book/Textbook
Requestor type Publisher
Publisher University of South Florida
Portion Full text
Format Electronic
Will you be translating? No
Print run 99999
Author of this Springer article Yes and you are the sole author of the new work
Order reference number None
Title of new book University of South Florida Dissertation
Publisher University of South Florida
Author of new book Shilpa Pendyala
Expected publication date of new book Dec 2015
Estimated size of new book (pages) 200
Total 0.00 USD

Terms and Conditions

Introduction
The publisher for this copyrighted material is Springer Science + Business Media. By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time

https://s100.copyright.com/CustomerAdmin/PLF.jsp?ref=8f2b9ac-b51ae-48b2-9d02-fc0f1bb3fa5e
ABOUT THE AUTHOR

Shilpa Pendyala received her Bachelors of Technology in Electrical and Electronics Engineering from Nagarjuna University, Guntur, Andhra Pradesh, India in 2007. She did her Masters in Computer Engineering from Missouri University of Science and Technology, Rolla, MO, USA in 2008. She then joined department of Computer Science and Engineering at University of South Florida, Tampa for a PhD. She interned with Intel Labs, Hillsboro, OR, USA in the summer of 2012. During her doctoral studies, she has published several peer reviewed conference papers. At VLSI SOC 2014, her work got nominated for the best paper award and further invited to submit the extended version as a book chapter. She got awarded a scholarship to attend WD2AT workshop and DAC summer school in 2011. She taught an undergraduate level course, Computer Organization, at USF in summer of 2011. She served as a secretary for IEEE CS student chapter.