MINIMUM DYNAMIC POWER CMOS DESIGN WITH VARIABLE INPUT DELAY LOGIC

BY TEZASWI RAJA

A dissertation submitted to the Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements for the degree of Doctor of Philosophy
Graduate Program in Electrical and Computer Engineering

Written under the direction of Prof. Vishwani D. Agrawal and Prof. Michael L. Bushnell and approved by

________________________________________

________________________________________

________________________________________

________________________________________

New Brunswick, New Jersey
May, 2004
ABSTRACT OF THE DISSERTATION

Minimum Dynamic Power CMOS Design with Variable Input Delay Logic

by Tezaswi Raja

Dissertation Directors: Prof. Vishwani D. Agrawal and Prof. Michael L. Bushnell

For correct functioning of CMOS circuits, every gate in the circuit needs to have an output transition, at most once for every input change. But in reality, the gates transition more than once due to unequal arrival times of signals at their inputs. These extra transitions are called glitches and they consume power. The dynamic power due to glitches in a CMOS circuit can be eliminated by the introduction of delay buffers at specific nets but some extra power would be consumed by these buffers. In this thesis we propose a new methodology of eliminating the glitches without the insertion of delay buffers.

The glitches are suppressed by the manipulation of the input delay of gates creating different delays for different input-output paths within the same gate. The technology constraints for the feasible ranges for length and width of transistors restrict the obtainable I/O delay difference for a gate. This difference is specified by an upper bound $u_b$ in the thesis. A given value of $u_b$ would result in a minimum dynamic power design that is the fastest allowed by that technology.
The circuit is designed by a *linear program* (LP) that determines the component delays to eliminate all glitches. We do not pose the problem as the *conventional transistor sizing* because delays are non-linearly related to transistor size parameters. We propose to solve the problem in three phases. The first phase is a detailed study of a single gate to determine the allowed \( u_b \) for the technology. In the second phase, an LP of size proportional to the circuit size is formulated by treating the gate level input delay variables independent of each other, but constrained by \( u_b \). We describe this as *variable input delay logic design* in the thesis. The third phase consists of the design of gates according to the delay assignment solution of the LP. Hence, the non-linearity in transistor sizing is relegated to the third phase, leaving the variables in the second phase independent. We describe possible ways of implementing the variable input delay logic at the transistor level. The gates can be designed by either *input capacitance manipulation* or by inserting resistances at gate inputs by inserting pass transistors.

This technique is scalable to large circuits because the formulation is linear. The circuits designed by this method are totally glitch-free and may use a minimal number of delay buffers necessary to satisfy the input-output delay specification. Our design of the benchmark circuit \( c7552 \) containing 3827 gates achieved a peak power savings of 68% and an average energy saving of 58% compared to an unoptimized circuit having the same input-output delay specification. This optimized circuit consumed 18% less energy than the circuit designed by inserting buffers at the same delay specification.
Acknowledgements

I would like to thank my professors, Dr. Vishwani Agrawal and Dr. Michael Bushnell who made me realize my true potential. They have been excellent guides and I feel fortunate to have worked with them. I thank Prof. Agrawal for giving me the freedom to explore other areas during my research. I thank Prof. Bushnell for instilling a strong sense of perfection in my writing and presentation style. I thank them both for helping me choose a career path as well.

I would like to thank my parents and family for being supportive and encouraging. I thank my dad for making me always see the bigger picture and I am sure he would have been proud of my achievement. I thank my grand father for his faith in me.

I thank my seniors Bantu, Sathe and Gaur Saab for their help and guidance. I thank Sameer, Siri, Omar, Hari, Krishna, Karthik, Rajesh, Bhaskar, Lalit, Vishal, Kunal, Swathi, Swetha, Sumit, Junwu, Gopi and Rajamani for making my life in the lab more interesting. The discussions we had on various topics were educational as well as a breather from work. I wish them all a great future. I thank my roommates Vijay, Dennis, Vas, Sameer, Manish, Neelesh, Hari and Tambe for bearing with me for this long. I would have lost my sanity many times had it not been for these guys.

I thank my machine faraday.rutgers.edu and the CAIP staff for putting up with all my complaints. I thank the people at Prolific Inc. for providing me with the library generation tool, Progenesis.

A special thanks to Deepti, who has been with me all along. I could not have done this without her support and understanding. I thank her for being the inspiration and motivation behind the whole thing.
Dedication

To all my teachers, from Kindergarten to PhD and beyond.
Table of Contents

Abstract ................................................................. ii
Acknowledgements ................................................. iv
Dedication ............................................................... v

List of Tables ........................................................... xiii
List of Figures .......................................................... xiv

1. Introduction ....................................................... 1
   1.1. Motivation .................................................... 1
       1.1.1. Why Reduce Power in CMOS Circuits? .............. 1
       1.1.2. Why Should Glitches Be Eliminated in a Circuit? .... 2
       1.1.3. Why Are Existing Glitch Reduction Techniques Inadequate? 2
   1.2. Problem Statement ............................................. 3
   1.3. Original Contributions of the Thesis ........................ 3
   1.4. Organization of the Thesis ................................... 4

   2.1. Introduction .................................................. 5
   2.2. Power Dissipation in CMOS Circuits ........................ 6
       2.2.1. Dynamic Power ........................................ 6
       2.2.2. Hazards and Glitch Power ............................ 7
       2.2.3. Short Circuit Power Dissipation ..................... 8
       2.2.4. Leakage Power Dissipation ........................... 10
   2.3. Software Level Low Power Techniques ..................... 13
## Table of Contents

2.3.1. Instruction Profiling .............................................. 13  
2.3.2. Compiler Power Optimization ................................. 15  
2.3.3. Memory Access Reduction .................................... 16  

### 2.4. System Level Power Management Techniques ................ 17  
2.4.1. Power Managed Systems ...................................... 18  
2.4.2. Predictive Techniques ....................................... 20  
2.4.2.1. Static Techniques ....................................... 21  
2.4.2.2. Adaptive Techniques .................................... 22  
2.4.3. Stochastic Control ............................................ 22  
2.4.4. Implementation of DPM ....................................... 23  
2.4.4.1. Clock Gating ............................................. 24  
2.4.4.2. Supply Shutdown ......................................... 24  
2.4.4.3. Multiple and Variable Power Supplies ............... 24  

### 2.5. Architecture-RTL Level Low Power Techniques .............. 26  
2.5.1. Control Data-flow Graphs ................................... 26  
2.5.2. Operation Scheduling ....................................... 27  
2.5.3. Resource Allocation ......................................... 28  
2.5.4. Bus Encoding ................................................ 29  

### 2.6. RTL-Gate Level Low Power Techniques ....................... 31  
2.6.1. Control Logic Synthesis ..................................... 31  
2.6.2. Charge Recovery Techniques .................................. 32  

### 2.7. Gate-Transistor Level Low Power Techniques ............... 33  
2.7.1. Low Power Technologies .................................... 33  
2.7.2. Path Balancing ............................................... 34  
2.7.3. Hazard Filtering ............................................. 37  
2.7.4. Gate Sizing .................................................. 39  
2.7.5. Transistor Sizing ............................................ 40  
2.7.6. Linear Programming (LP) Approach ......................... 41  

### 2.8. Summary .......................................................... 41
3. Prior Work: Techniques for Power Estimation

3.1. Software-Level Power Estimators

3.2. Behavior-Level Power Estimation

3.2.1. Information Theoretic Models

3.2.2. Complexity Based Models

3.2.3. Synthesis Based Models

3.3. RT-Level Power Estimation

3.3.1. Regression Based Models

3.3.2. Sampling-Based Models

3.4. Gate-Level Power Estimation

3.4.1. Simulation-Based Techniques

3.4.1.1. Event-driven Power Estimation

3.4.2. Probabilistic Techniques

3.4.3. Statistical Techniques

3.5. Summary

4. Prior Work: Delay Elements in CMOS Circuits

4.1. Introduction

4.2. Buffers

4.2.1. $m$ Transistor Buffer

4.2.2. Staged Buffers

4.3. Transmission Gates

4.3.1. Transmission Gate Cascaded with Schmitt Trigger

4.4. Voltage Controlled Elements

4.4.1. $n$-Voltage Controlled Elements

4.4.2. $p$-Voltage Controlled Elements

4.4.3. $np$-Voltage Controlled Elements

4.5. Summary
5. Background on Linear Programming Techniques ............. 67
  5.1. Definitions ...................................................... 67
  5.2. Overview of Linear Programming (LP) Approaches ........... 68
    5.2.1. Berkelaar et al.'s Method ................................. 68
      5.2.1.1. Objective Function ....................................... 69
      5.2.1.2. Constraints .............................................. 69
    5.2.2. Agrawal et al.'s Method ................................. 70
      5.2.2.1. Variables .............................................. 71
      5.2.2.2. Objective Function ....................................... 72
      5.2.2.3. Constraints .............................................. 72
  5.3. Raja et al.'s Method ......................................... 74
    5.3.1. The Concept of a Timing Window Per Gate .................. 74
    5.3.2. Linear Program ............................................ 76
      5.3.2.1. Variables .............................................. 76
    5.3.3. Objective Function ......................................... 77
      5.3.3.1. Initial Constraints ....................................... 77
      5.3.3.2. Gate Constraints ........................................... 77
      5.3.3.3. Overall Circuit Delay Constraints ...................... 78
    5.3.4. Validation of the Model .................................... 78
    5.3.5. Why is this Model Superior? .............................. 81
  5.4. Summary of Linear Programming Techniques .................. 82

6. New Variable Input Delay Logic ................................. 84
  6.1. Why Were Buffers Necessary? ................................. 84
  6.2. Variable Input Delay Logic ..................................... 87
    6.2.1. Technology Parameter $u_b$ ................................ 88
  6.3. A New Minimum Dynamic Power LP ............................. 90
    6.3.1. Variables .............................................. 91
    6.3.2. Constraints on Delays ..................................... 91
6.3.3. Glitch Suppression Constraints ..................... 91
6.3.4. Maxdelay Constraints .............................. 92
6.3.5. Objective Function ................................. 92
6.3.6. Feasibility Constraints .............................. 92
6.4. Delay Specification LP ................................. 93
6.5. General Solution LP ................................. 94
6.6. Solution Curves ....................................... 95
6.7. Original Contributions ............................... 97
6.8. Summary ............................................. 97

7. Transistor Level Design of Variable Input Delay Logic .......... 98

7.1. Conventional CMOS Logic Gates .......................... 98
  7.1.1. Components of Gate Delay ......................... 99
7.2. Gate Design by Input Capacitance Manipulation ........ 102
  7.2.1. Calculation of \( u_b \) ............................ 102
  7.2.2. Design Issues .................................. 103
7.3. Gate Design with \( n \)MOS Pass Transistors .......... 105
  7.3.1. Concept of Increasing Resistance ........ ........ 105
  7.3.2. Effect of Input Slope .......................... 106
  7.3.3. Proposed Gate Design ........................... 107
  7.3.4. Calculation of \( u_b \) ............................ 108
  7.3.5. Design Issues .................................. 110
7.4. Gate Design with CMOS Pass Transistors .............. 111
  7.4.1. Calculation of \( u_b \) ............................ 112
  7.4.2. Design Issues .................................. 113
7.5. Technology Mapping ................................ 113
  7.5.1. Place-and-Route Design Flow ..................... 114
  7.5.2. Delay Components ............................... 115
  7.5.3. Search Dimensions .............................. 115
7.5.4. Lookup Table Generation .......................... 116
7.5.5. Fine Tuning the Transistor Size Assignment ........ 119
7.6. Original Contributions ................................. 120
7.7. Summary ............................................. 120

8. Results ................................................... 121
8.1. Experimental Procedure ............................... 121
8.2. Feasible Gate Differential Delay Upper Bound ........ 122
8.3. Example Circuit ....................................... 123
  8.3.1. Energy Consumption in the Example Circuit ....... 125
  8.3.2. Leakage Current Analysis for the Example Circuit .. 126
8.4. Power Savings for Benchmark Circuits Using the Minimum Dynamic Power LP .......................... 128
8.5. Power Savings for Benchmark Circuits Using the Delay Specification LP .................................. 129
8.6. Summary ............................................. 132

9. Physical Design and Implementation of the c7552 Benchmark Circuit ........................................... 134
  9.1. Overall Design Flow .................................. 134
    9.1.1. Logic-level Constraint Generation ................. 135
    9.1.2. Technology Mapping ............................... 136
    9.1.4. Abstract Generator ................................ 137
    9.1.5. Verilog Generator ................................ 137
    9.1.6. Silicon Ensemble ................................ 137
    9.1.7. Extraction ....................................... 138
  9.2. Incorporating Routing Delay into the LP Formulation .... 138
  9.3. Physical Design ..................................... 139
  9.4. Power Consumption Results ............................ 141
9.5. Summary ...................................................... 142

10. Conclusion and Future Work ............................ 145
   10.1. Conclusion ................................................. 145
   10.2. Future Work .............................................. 146
       10.2.1. Glitch Reduction Using a Statistical Delay Model .. 146
       10.2.2. Leakage Current Reduction .......................... 148
       10.2.3. Design of a Low Power Standard Cell Library ...... 148
       10.2.4. Efficient Search Algorithms for Gate Sizing ....... 148
       10.2.5. Variable Input Delay Logic Gate Modeling ......... 149
       10.2.6. New Delay Elements ............................... 149

References ......................................................... 150

Curriculum Vita .................................................. 165
List of Tables

8.1. Dynamic power dissipation in ISCAS’85 benchmark circuits for the proposed (no buffer) design method. We used the minimum *dynamic power* LP and chose $u_b = 10$. The method gives the fastest minimum dynamic power circuit for the chosen $u_b$ value. $Maxdelay$ is normalized to the fastest possible circuit before optimization, i.e., the length of the *critical path* of the circuit. Power measurements are done by using a weighted fanout variable delay simulator [77]. The table shows the optimized power consumption normalized to the power consumption of the unoptimized version of the circuit. Power savings are (1-value shown in the table). ........................................ 132

8.2. Power measurement results of the proposed *delay specification* LP on all ISCAS’85 benchmark circuits. The circuits are designed for two delay specifications given by *maxdelay*, and for $u_b = 10$. The number of buffers inserted by this method are compared with the published results of Raja *et al.* [156, 157]. $Maxdelay$ is normalized to the fastest possible circuit before optimization, i.e., the length of the *critical path* of the circuit. Power measurements are done by using a weighted fanout variable delay simulator [77]. The table shows the optimized power consumption normalized to the power consumption of the unoptimized version of the circuit. Power savings are (1-value shown in the table). ........................................ 133
List of Figures

2.1. An example of a static hazard. ................................. 7
2.2. An example of a dynamic hazard. .............................. 8
2.3. Effect of load capacitance on short circuit power dissipation. 9
2.4. Short circuit energy dissipation versus input rise/fall time. .... 10
2.5. Leakage currents in an inverter. ................................. 11
2.6. Leakage current vs. Gate-to-Source voltage characteristic of MOS-FET shown by the solid curve. The dashed line shows the effect of decreasing threshold voltage and the dotted line shows the effect of increasing temperature. ................................. 12
2.7. Code optimization to reduce the number of memory accesses. ... 16
2.8. Power state machine of a typical microprocessor, showing high and low power states with state transition costs. ............................ 19
2.9. Abstract structure of a system-level power manager. ............. 20
2.10. Illustration of computation of a second-order polynomial using adders and multipliers in a circuit. (a) Unoptimized version and (b) optimization using control data flow graph transformations. ... 27
2.11. Glitching behavior of an ill-structured circuit. .................. 35
2.12. Removal of glitches after restructuring of the circuit in Figure 2.11. 35
2.13. Glitch behavior of a circuit where simple restructuring will not balance paths. ................................................. 36
2.14. Path balancing of the circuit in Figure 2.13 by buffers. ........... 36
2.15. Energy dissipation in an inverter. ............................... 37
2.16. Effect of gate delay on energy dissipation. ....................... 38
3.1. Simulation and probabilistic techniques of power estimation. ... 52
4.1. Buffer delay elements. (a) Simple cascaded inverter pair buffer delay element, (b) \(m\)-cascaded inverter delay element and (c) Staged cascaded inverters. .................................................. 60

4.2. Transmission gate delay elements. (a) Simple CMOS pass transistor as delay element and (b) Transmission gate cascaded with a Schmitt trigger circuit for a steeper output slope. .................. 63

4.3. Voltage controlled delay elements. (a) \(n\)-Voltage controlled delay element, (b) \(p\)-Voltage controlled delay element and (c) \(np\)-Voltage controlled delay element. ................................................. 65

5.1. A gate illustrating paths from PIs. ................................. 70

5.2. A 1-bit adder. .......................................................... 71

5.3. An arbitrary distribution of events at the inputs of a gate. ..... 75

5.4. A combinational circuit block. ..................................... 79

5.5. A combinational array circuit using the block of Figure 5.4. ... 81

5.6. Number of constraints for the new LP model and the old (path enumeration) model. .................................................. 82

6.1. An example circuit with a critical path delay of 3 units. The gate delays and the arrival times of signals at the gate inputs are shown. 85

6.2. The optimized circuit of Figure 6.1. The optimization is done by the insertion of a delay buffer, as proposed by Raja et al. [156, 157]. 85

6.3. The optimized circuit of Figure 6.1 using the proposed technique. This example shows that it is possible to eliminate glitches without the help of delay buffers, if we can design the new \textit{variable input delay gate}. .................................................. 86

6.4. Conventional and proposed gate delay models. (a) Delay model of the conventional gate design showing a single inertial delay, (b) Variable input delay gate and (c) Variable input gate delay model showing the input delay variables that are independent of the inertial delay. .................................................. 87
6.5. A combinational circuit. .................................................. 89
6.6. Delay model for the circuit of Figure 6.5. ............................. 90
6.7. Plot of the power and solution curves of the previous method of Raja et al. [156, 157] and the proposed methods of this thesis. The minimum dynamic power LP solution points show the solutions which are minimum power consuming for a given technology specification $u_b$. The designs given by these points are the fastest minimum power circuits for that $u_b$. The maximum $u_b$ allowed for a technology is given by Max $u_b$. This is the bound of the feasibility region for that technology. For the delay specified circuits, the minimum power design solution can be speeded up by adding extra buffers. The power is slightly increased as shown by the points labeled proposed delay specification LP method. The proposed general LP solution is shown by the dotted line which is the entire power-delay tradeoff curve. Hence, the methods proposed in this thesis effectively exploit the power-delay solution space. .......................... 96
7.1. (a) Schematic of a CMOS inverter. (b) Switch configuration when input applied is 1. (c) Switch configuration when input applied is 0. 99
7.2. Schematic of a CMOS NAND gate and the truth table of input combinations. ................................................................. 99
7.3. Timing diagram illustrating the delay of the NAND gate of Figure 7.2. Input 2 is the cause of the transition at output 3. The interval $d$ is the delay of the gate. ................................................... 100
7.4. The RC components along the charging path. ......................... 101
7.5. The proposed single added nMOSFET variable input delay NAND gate. (a) Transistor Level showing the nMOS transistor added and (b) charging path for transitions along the different paths through the gate. ......................................................... 107
7.6. The logic degradation of the single nMOS transistor addition (a) 
When logic 1 is passed through and (b) When logic 0 is passed 
through the gate. .................................................. 108
7.7. The proposed CMOS added pass transistor variable input delay NAND gate. (a) Transistor Level showing the nMOS and pMOS 
transistors added and (b) Charging path for transitions along the 
different paths through the gate. .................................. 112
7.8. Flowchart showing the standard cell place-and-route. The different 
cells for every gate instance have to be present in the Cell Library. 114
7.9. Flowchart of the lookup table generation algorithm for transistor 
sizing. ................................................................. 118
8.1. Flow chart depicting the experimental procedure for generating the 
results at the logic level. ........................................ 122
8.2. Normalized maxdelay versus $u_b$. The maxdelay is normalized to the 
fastest possible circuit design, i.e., without altering delays along the 
critical path. ......................................................... 123
8.3. An example circuit. ................................................ 124
8.4. Output waveforms for one input transition at the PIs of the unop- 
timized circuit. Glitches are produced at gates 5 and 6. ........ 125
8.5. Optimized example circuit with buffer insertion. ................. 126
8.6. Output waveforms for one input transition at the PIs of the buffer 
optimized circuit. Glitches are eliminated. ...................... 127
8.7. Optimized example circuit with the proposed gate. ............. 128
8.8. Output waveforms for one input transition at the PIs of the opti- 
mized circuit with the new gate design. ......................... 129
8.9. Energy consumption comparisons of the three different circuits in 
fJ. The new gate optimized circuit consumes 62.5% less energy 
than the unoptimized circuit and 45% less energy than the buffer 
optimized circuit. ................................................... 130
8.10. Leakage current analysis for three different circuit configurations in fJ. The circuits considered were: (1) Unoptimized circuit, (2) Optimized circuit with nMOS pass transistors added and (3) Optimized circuit with CMOS pass transistors added. The leakage current is increased for input vector 111 by 0.2% with CMOS pass transistors and by 0.45% for nMOS pass transistors. The leakage current is unaffected for the input vector of 000. .......................... 131

9.1. Flow chart showing the entire design flow for automated design of large circuits. ................................................................. 135

9.2. A combinational circuit with the routing delays of nets. .......... 138

9.3. The layout of the unoptimized circuit c7552 designed using the generic standard cell design library. ................................. 140

9.4. The layout of the optimized circuit c7552 designed using the proposed variable input delay technique. The area increased 14% compared to the unoptimized circuit in Figure 9.3. Both circuits function at the same speed of operation. ................................. 141

9.5. The layout of the optimized circuit c7552 designed using the buffer insertion technique proposed by Raja et al. [156, 157]. The area increased 24% compared to the unoptimized circuit in Figure 9.3. All three circuits function at the same speed of operation. ........ 143

9.6. Instantaneous energy consumption results for benchmark circuit c7552 for 156 vectors. Results of the optimized circuit using the proposed method show a peak power savings of 68% over the unoptimized circuit. ......................................................... 144

9.7. Average energy consumption results for benchmark circuit c7552 for 156 vectors. Results of the optimized circuit using the proposed method show an average energy savings of 58% over the unoptimized circuit. ......................................................... 144
10.1. Hazard filtering condition for a statistical delay model. The old
timing window is stretched to account for the delay distribution
due to process variations. The delay varies as a Gaussian curve
with a deviation $\sigma$ as shown. 

10.2. A new delay element combining the buffer and the pass transistor.
This element does not suffer from the input slope degradation as
much as a single pass transistor as the second inverter stage acts
as a slope shaper circuit.
Chapter 1

Introduction

The contribution of this work is a new design methodology for optimizing CMOS circuits for minimum dynamic power consumption without a loss in speed. This work focuses on the delay assignments for the gates in the circuit for glitch elimination. A new linear program formulation is presented assuming gates that have variable input delays at the logic level. These input delay variables are allowed to vary within a technology-dependent parameter called $u_b$. The solution to the LP is then realized at the transistor level using new variable input delay gate designs. This design methodology eliminates the insertion of extra delay buffers as done by existing techniques. Since no extra elements are inserted, this technique saves more power than those techniques.

1.1 Motivation

In this section, we address several basic issues that motivated this work.

1.1.1 Why Reduce Power in CMOS Circuits?

The continuing decrease in the feature size and the corresponding increases in chip density and operating frequency have made power consumption a major concern in VLSI design. Excessive power dissipation in integrated circuits discourages their use in portable systems. It also causes overheating, which degrades the performance and reduces chip lifetime. To control their temperature levels, the chips need specialized and costly packaging and cooling arrangements, which result in
further escalation of the system cost. The growing need for portable communication devices and computing systems has increased the need for optimization of power consumption in a chip. Overall, low power design is a critical technology needed in the semiconductor industry today. Simultaneously, we also need to speed up the critical paths of the circuit, while reducing its power consumption.

1.1.2 Why Should Glitches Be Eliminated in a Circuit?

For the correct functioning of the circuit, a gate output needs to transition at most once per every vector transition [2]. But in reality, the gates switch more than once due to differing signal arrival times at the inputs of the gate. These extra transitions are known as glitches. Every transition at a gate output consumes energy and since glitches are unnecessary transitions, they waste energy. Glitches need to be eliminated to prevent this wastage of energy.

1.1.3 Why Are Existing Glitch Reduction Techniques Inadequate?

There have been many different techniques proposed for glitch elimination in circuits. A total elimination of glitches requires differential path delays to satisfy certain conditions at every gate. This can be done by balancing the paths by inserting buffers along slow paths or slowing down selected gates [1, 146]. Gate delay can be a non-linear function of the sizes of transistors in the gate. Hence, there have been techniques that treat the gate as an equivalent inverter whose sizes are treated as variables (gate sizing) or the transistor sizes in the entire circuit are treated as variables (transistor sizing) [20, 56, 179]. But these techniques suffer from non-linearity of the models and cannot optimize large circuits efficiently. Another set of techniques called linear programming (LP) techniques model the system as a linear program with constraints set up to obtain the desired conditions at every gate of the circuit, and to limit the overall delay [2, 156, 157]. A solution finds the delays of all gates and buffers and at the same time minimizes the
number of buffers inserted. Linear programming is preferred as it guarantees a
global optimum and hence the result would produce the maximum power savings.

The LP techniques are efficient for large circuit optimizations as well, but do
so by inserting extra buffers in the circuit. The power savings are thus reduced
by the energy consumed in the added circuitry. For example, the benchmark
circuit c7552 containing 3827 gates, when optimized by the technique proposed
by Raja et al., needed 366 buffers to be inserted, an increase of 9.5% for no loss in
speed [156, 157]. The average power reduction was 38%, which could be further
increased if the glitch elimination could be achieved without buffers.

1.2 Problem Statement

The problem solved in this work is: Find a new glitch reduction technique that
eliminates glitches without inserting extra buffers in the circuit and without any
reduction in speed. The technique should be scalable for large circuits and should
be realizable at the physical level of design.

1.3 Original Contributions of the Thesis

In this work we propose a new methodology for glitch-free design of digital cir-
cuits. At the logic level, we propose a new LP formulation that uses a technology
dependent parameter $u_b$ to generate the feasible delays of gate variables. At the
transistor level, we propose three new ways of implementing the variable input
delay logic gate that can be used for realizing the output gate delays of the LP.
We propose ways of quantifying the value of $u_b$ for each of these designs and for
different technologies making the overall technique technology compatible. This
is a technique that eliminates glitches without loss in speed and without inserting
delay buffers. Most importantly, this technique optimizes large circuits, keeping
the formulation linear. Since no extra gates or buffers are added to the circuit, the
power savings achieved are the maximum possible. For example, the benchmark
circuit c7552 designed with this new technique at the same speed, consumed 58% less average energy and 68% less peak power than the unoptimized circuit. These measurements were 18% less than the circuit optimized with the buffer insertion technique [156, 157].

1.4 Organization of the Thesis

In Chapter 2, we present a survey of various low power techniques, organized according to the stage in the design process in which they are effective. In Chapter 3, we present a survey of power estimation techniques at different levels of design. In Chapter 4, we present a discussion on various delay elements used in CMOS digital circuits that are relevant to the present work. A background of the prior work on glitch reduction using linear programming (LP) techniques to adjust delays is presented in Chapter 5. A new LP formulation based on a logic level analysis is presented in Chapter 6. The proposed gate designs for realizing variable input delays are presented in Chapter 7 along with a method to calculate the technology parameter $u_b$. Logic level power analysis and results are presented in Chapter 8. The entire design methodology for the design of benchmark circuit c7552 is presented in Chapter 9 along with the power measurement results at the physical level. We conclude in Chapter 10 with some pointers to the future work.
Chapter 2

Prior Work: Techniques for Low Power Design

2.1 Introduction

In this chapter, we review the basic mechanisms of power dissipation in CMOS circuits. We describe the existing low-power techniques at every level of the design process. We concentrate on the techniques used for optimizing digital circuits. Our classification is based on the degree of freedom allowed to the designer in creating a low power design. At the software level the target architectures are fixed and, hence, the techniques are specific to the microprocessor architecture underneath. These techniques optimize power by rescheduling software over an existing hardware and are called software level low power techniques.

A system is comprised of a set of functional units whose description is behavioral. The designer has the flexibility to assign extra capabilities to these individual blocks. Certain decisions can be taken to coordinate the different units to shut them off when idle. These techniques are known as system level power management techniques. Once the system behavior is decided, the architecture of individual components presents opportunities to save power, such as bus encoding, resource allocation in a microprocessor, etc. We categorize such techniques as architecture level optimization techniques. Typically the architectural level analyses are done with blocks in the register transfer (RT) level. These blocks are synthesized into a gate level netlist using automatic synthesis tools. Power can also be saved at this level of the design process and we describe these as the RT-gate level low power techniques. Once the gate level netlist is decided, the designer is left with very little freedom to change the basic functionality. But
the implementation of the logic functions can still be achieved by manipulation of
the transistor sizes. The techniques that are targeted in the design stage of going
from logic to transistor level of design are categorized as logic-transistor level low
power techniques.

2.2 Power Dissipation in CMOS Circuits

There are three main sources of power dissipation in digital CMOS circuits:

- Dynamic power
- Short circuit power
- Leakage current power.

2.2.1 Dynamic Power

The dynamic power dissipation in a CMOS gate is due to the charging and discharging of load capacitance driven by the gate. This capacitance consists of the
internal capacitances of the gate, wire capacitances of the fanout net and the capacitances of the gate terminals of the transistors being controlled by the fanout net. This power dissipation can be calculated by the following equation [54, 152]:

\[ P_{dyn} = \frac{1}{2} C_{load} V_{dd}^2 f D \]  

(2.1)

where

- \( P_{dyn} \): dynamic power dissipation of gate
- \( C_{load} \): load capacitance of the gate
- \( f \): clock frequency
- \( V_{dd} \): supply voltage
- \( D \): transition density of the output of the gate
Figure 2.1: An example of a static hazard.

The transition density is the average number of transitions during a clock cycle [134]. The dynamic power dissipated is thus proportional to the number of transitions occurring at a gate. In prior CMOS technologies, dynamic power accounted for most of the power used by the circuits [153]. But with the advent of deep sub-micron technology, the leakage current component of power consumption is also becoming significant. However, several low power techniques have concentrated on minimizing dynamic power, as will be explained later. Dynamic power can be classified into necessary switching activity for the correct functioning of the circuit and unnecessary transitions due to unbalanced paths in the circuit. The latter component of dynamic power dissipation is the glitching power and is elaborated in the next section.

2.2.2 Hazards and Glitch Power

Before signals of a digital circuit reach steady state, gates can have multiple transitions. Since the power consumed is directly proportional to the number of transitions, these unnecessary transitions increase the power consumption. These transitions are called glitches or hazards. Glitches are produced in a circuit due to the difference in signal arrival times at the inputs of a gate. Power consumed by glitches is called glitch power and it typically amounts to at least 20% of the overall power consumption of the digital circuitry in a chip and as high as 70% in circuits such as a combinational adder [166]. In this section we familiarize the reader with the terminology of hazards and also the issues involved before we discuss the techniques for glitch suppression in the next section.

Consider the example of Figure 2.1 with each gate having one unit of delay.
Figure 2.2: An example of a dynamic hazard.

Due to the difference in arrival times of signals at the inputs of the AND gate the output produces a pulse (called a *glitch*) of 1 unit width, which equals the inverter delay. This is known as a *static hazard*. In Figure 2.2 the OR gate produces a static hazard of 1 unit. The transient at the output consists of three edges, two rising and 1 falling. This is a *dynamic hazard* with a combined width of 2 units.

The number of edges in transients at the output of a gate may equal the number of arriving signals at the gate. The maximum difference in arrival time of signals at the inputs of a gate is called the *differential path delay*. It is the maximum width of the transient period at the circuit output. A *hazard producing gate* has more than one input and has a non-zero differential path delay. Every gate has an *inertial delay* due to the finite switching speed of the transistors. It is the time a device takes to switch the output, after the cause for the change has occurred at the input [1]. Inertial delay plays a major role in distorting the glitches produced by the gate. In Section 2.7, we present ways of eliminating glitches by either making the differential path delay zero or by increasing the inertial gate delay.

### 2.2.3 Short Circuit Power Dissipation

Short-circuit power dissipation occurs when a gate switches. During the transition there is a short time when both the nMOS and pMOS transistors conduct. This is equivalent to shorting the supply and ground rails for a brief interval of time. The current through these transistors dissipates power.

The value of the short circuit current greatly depends on the capacitive load
on the output of the gate. Consider the example in Figure 2.3. For a high load capacitance $C_L$, the output fall time is significantly larger than the input rise time. It follows that the short circuit current is very close to zero in this case. Conversely, for a low capacitive load, the output fall time is substantially smaller than the input rise time. It follows that the short circuit current is at its maximum. Thus, the short circuit power optimization techniques are aimed at making the output rise/fall times larger than the input rise/fall times. Thus, the amount of short-circuit power depends on the switching speed of the gate [197]. Average short circuit power can be estimated using the average current flow through the inverter. For a pair of rising and falling input transitions seperated by a period of $T$ at the input of an inverter, the short-circuit power dissipated is given by [197]:

$$P_{\text{short-circuit}} = \frac{\beta}{12} (V_{dd} - 2V_T)^3 \frac{\tau}{T}$$

(2.2)

where

- $P_{\text{short-circuit}}$: short circuit dissipation
- $\beta$: gain factor of the inverter
- $V_{dd}$: supply voltage
- $V_T$: absolute threshold voltage of nMOS and pMOS transistors (assumed equal)
- $\tau$: rise or fall time of the input signal
- $T$: period of the signal.

![Graph showing short circuit power versus rise/fall time](image)

**Figure 2.4:** Short circuit energy dissipation versus input rise/fall time.

Figure 2.4 is a plot of short circuit energy dissipation versus the ratio of input rise/fall times to output rise/fall times [195]. As can be seen, the longer the input rise/fall time, the longer the short circuit current will flow, and the average short circuit power increases. For most ICs, the short circuit dissipation is approximately $5 - 10\%$ of the total power dissipated.

### 2.2.4 Leakage Power Dissipation

There are two types of leakage currents: *reverse biased leakage* through the parasitic diodes between the source and the bulk, and *sub-threshold leakage through the channel of a device* [26]. The magnitude of those currents is set predominantly
by the processing technology. However, there are some things that a designer can do to minimize their contribution.

Diode leakage current occurs when the transistor is turned off, and another active device charges up/down the drain with respect to the former’s bulk potential. Consider the inverter in Figure 2.5 when it is given a low voltage as input. The pMOS transistor is turned on and the output charges to $V_{dd}$. The parasitic diode from output to the nMOS bulk is reverse biased with voltage $-V_{dd}$. This diode has a current to the substrate given by:

$$I_{leakage} = i_s (e^{qV/kT} - 1)$$

where $i_s$ is the reverse saturation current in the diode, $V$ is the reverse bias voltage ($V_{dd}$ in this case), $q$ is the electronic charge constant ($1.602 \times 10^{-19} C$), $k$ is the Boltzmann’s constant and $T$ is the temperature in Kelvin [152, 205]. The leakage current is of the order of $0.1 - 0.5 nA$ per device at room temperature. The leakage current can increase dramatically at high temperatures.

The other component of leakage current is the sub-threshold leakage. In the inverter shown in the Figure 2.5, even when the nMOS transistor is turned OFF, there is a current flowing through its channel known as sub-threshold current $I_d$. 

Figure 2.5: Leakage currents in an inverter.
Figure 2.6: Leakage current vs. Gate-to-Source voltage characteristic of MOSFET shown by the solid curve. The dashed line shows the effect of decreasing threshold voltage and the dotted line shows the effect of increasing temperature.

This current is dependent on the gate voltage $V_{gs}$, drain voltage $V_{ds}$ and temperature. A typical plot of $I_d$ vs $V_{gs}$, as shown in Figure 2.6, has an exponential relation in the sub-threshold region. The magnitude of the sub-threshold current is a function of the process, device sizing and supply voltage. The process parameter that affects the current value is the threshold voltage $V_t$. Reducing $V_t$ exponentially increases the sub-threshold current. For supply voltages of 1V, the $V_t$ can be reduced up to 0.4V in deep-submicron enhancement-mode MOS devices [96]. The sub-threshold current is also proportional to the device size (width/length). Thus, the current can be minimized by reducing the transistor sizes, and by reducing the supply voltage. Voltage scaling is one of the techniques that reduces leakage power drastically and is described in the architecture level techniques for power optimization.
In the following sections we look at the various low power techniques at different levels of design.

2.3 Software Level Low Power Techniques

The software low power techniques are targeted to a specific existing architecture. The main idea is to use the resources available in the architecture in an orderly fashion that reduces overall power consumption of the system while executing a program [184]. The generation of a software optimizer requires the construction of a profile of power consumption of each microprocessor instruction and the adequate selection of compiling techniques, which can reduce the energy consumed by the program. Since the architecture of every microprocessor is different, the modeling and profiling is architecture specific. This makes the software techniques architecture dependent. First we will look at some instruction profiling techniques. The optimization techniques are classified into two categories, where the optimization is done by:

- Instruction rescheduling and
- Memory access reduction.

2.3.1 Instruction Profiling

Profiling is the first step in estimating the power consumed by a program. The generation of an instruction profile requires the use of a system that can measure the power consumed by each instruction and keep track of certain special cases during the execution of the program such as stalls and cache misses.

Two main methods have been reported to achieve this objective. The first method by Tiwari et al. measures the current drawn by the processor during the execution of every instruction [185, 187]. Thus, by adding the power contributions of each instruction in a program, the total power consumption can be estimated. The current drawn by the processor can also be measured using an
oscilloscope with a shunt resistor connected in series with the supply voltage pin of the microprocessor [73]. This measurement can is also possible by a simple ammeter for high frequency processor [186]. The average power and energy can be calculated as

\[ P_{average} = V_c \times I_{average} \]

\[ Energy = V_c \times I_{average} \times T_{clock} \]

where \( V_c \) is the supply voltage of the microprocessor, \( I_{average} \) is the average measured current of the supply and \( T_{clock} \) is the clock period of the microprocessor. This method is applicable to pipelined processors, but there are effects that are not accounted for in this estimation procedure. As an example, the energy cost of an instruction is also dependent on the current state of the system, which cannot be measured from this procedure. Another effect is the cache miss ratio, which leads to additional execution time and energy penalty. A solution to this problem can be achieved by calculating the cost for a pair of instructions instead of the standalone instructions [185].

The second method is based on the simulation of the microprocessor and the effect of the instruction set in the microprocessor model [42, 84]. Mehta et al. suggest that lower level simulation can provide an estimate of the current drawn to calculate power of each instruction [125, 126].

Chakrabarti et al. built a model of each module of a microcontroller using a hardware description language [30]. Black box models have also been tried for hierarchical power estimation [42]. Once the modules are characterized, the overall power can be estimated by emulating the program execution. A drawback of this method is that detailed information about the CPU must be available. More details about the power estimation techniques are provided in Chapter 3.
2.3.2 Compiler Power Optimization

A single high-level language program can be compiled into a variety of micro-instruction sequences [3]. Once the micro-instructions are profiled for their relative power consumptions, the next step is to schedule them within the program such that the overall power consumption is minimized. This can be achieved during code compilation when the compiler is aware of the relative weights in the instruction set and orders the instructions accordingly. Knowledge of the architecture is imperative in this step. We discuss a few prominent compiler optimizations for low power in this section.

The switching activity in a circuit is a function of the present inputs and the previous state of the circuit. Thus, an appropriate reordering of the instructions in a program results in a lower use of energy. Su et al. describe a technique known as cold scheduling for scheduling instructions on an experimental RISC microprocessor in such a way that the switching on the control path is minimized [177]. Their paper describes a reduction of 20-30% in overall power.

The instruction selection module of modern compilers is often generated automatically by programs called code-generator generators. IBURG is such a program that is used to generate code for lcc, a retargetable ANSI C compiler [59]. These programs accept a specification of the instruction set of the targeted architecture. The specification defines a set of instruction patterns that can be included in the generated code and assigns a cost for each pattern. A code generator accepts an intermediate representation such as a directed acyclic graph (DAG) of each basic block in the source code. It uses pattern matching and dynamic programming to find a cover for the DAG in terms of specified instruction patterns such that the overall cost is minimized [184]. The cost function used in most compilers is the number of executing cycles, but it can be changed to energy costs for a code generator that targets reduced energy consumption.

A more articulated methodology for code generation and optimization was
for(i=0;i<n;i++)
    b[i] = f(a[i]);
for(i=0;i<n;i++)
    c[i] = g(b[i]);

(a) 

for(i=0;i<n;i++) {  
    b[i] = f(a[i]);
    c[i] = g(b[i]);
}

(b) 

Figure 2.7: Code optimization to reduce the number of memory accesses.

proposed by Lee et al. [102]. In this solution, techniques such as instruction packing, minimization of circuit state effects and operand swapping are exploited [187].

A loop unrolling technique was successfully applied to DSP processors by Vishal et al. [167]. In this approach the main sources of power dissipation were arithmetic logic units (ALUs) and memories. The objective was to reduce the total number of comparisons in a given program as each comparison consumes power. The results reported a low use of the ALU (<20%) with the drawback of increased code size (>10%). Predictive models have also been used to pre-estimate the power and optimize the operations of DSP cores statically [63].

Some processor architectures support dedicated instructions for power minimization. These micro-instructions could be additions to an existing architecture or to an entirely new one [72, 73, 164].

2.3.3 Memory Access Reduction

A major component of power consumption is the memory accesses in program execution [44]. This can be minimized by reducing the total number of memory accesses in a single program. In this section, we discuss some of the techniques used for reduction of memory accesses.

Consider the example code shown in Figure 2.7(a). If we assume the size of the array b to be too large to fit in the registers in the CPU, a total of 2n read/write accesses to the memory are needed for the intermediate array b during the execution of the program. By transforming the code to what is shown in Figure 2.7(b), the required element array b can be kept in a register of the processor.
Therefore, only register accesses are necessary to store and load the data. This reduces the total number of memory accesses and also the energy consumed by the program.

Panda et al. focus on the reduction of power dissipated in off-chip drivers and memory decoding logic, by reducing the number of address bus transitions [144, 145]. The goal is reached through a memory mapping scheme that allows one to place large arrays of data in the main memory, for which the access patterns can be extracted from a program source code at the compile time. Additional contributions to the problem of finding data allocations that minimize power in the memory-processor interface are available in the literature [112, 201]. These techniques have the same objectives as the bus encoding techniques described in later sections.

Memory hierarchy can be effectively exploited to reduce power consumption. The higher levels of memory consume less power but are limited in capacity. Power can be reduced by organizing the data in a way to increase the higher level accesses and reduce the amount of lower level accesses [121]. Catthoor et al. used this principle for reducing power in DSP and video applications [49, 202]. They present a formalized methodology for the choice of the proper memory hierarchy to be adopted in the design of data intensive applications.

2.4 System Level Power Management Techniques

The activity of several components in a system is event driven. For example, the activity of the display servers and user interface functions is triggered by external events and it is often interleaved with long periods of quiescence. An intuitive way of reducing power consists of shutting down the resources during their periods of inactivity. This technique is referred to as dynamic power management (DPM) [12].
Dynamic power management (DPM) is a design methodology for dynamically reconfiguring systems to provide the requested services and performance levels with a minimum number of active components or a minimum load on such components [108].

In this section we first describe the components that can utilize DPM and their applicability to different scenarios. Then, we describe the predictive and stochastic management policies and the implementation strategies.

2.4.1 Power Managed Systems

A power manageable component (PMC) is an atomic block in a complete system. It depends on the abstraction level of the system. It can be as simple as a functional unit or as complicated as a printed circuit board (PCB). A power manageable system must have PMCs that have multiple modes of operation that span the power-performance trade off. Hence, these PMCs can switch within modes of operation depending upon the requirement and the power budget. An ideal PMC has a continuous variation in the power-performance trade off curve and has an infinite number of operating modes that it can switch between instantaneously. In reality, however, the number of modes of operation is actually quite small due to the added design complexity and hardware overhead for supporting power management. Another factor to consider is that the transitions between operation modes also involve cost in terms of power and time. For example, the transition cost to return from a low power state involves time and energy for turning on and stabilizing the power supply and clock, reinitializing the system and restoring the context. Transition cost depends on the implementation strategy considered. Excessive transition cost can reduce the power savings achieved by PMCs.

Once the PMCs are decided in the system, a power state machine (PSM) is designed. This PSM consists of the states that are the various modes of operation of the PMC with differing power and latency. High power states usually
Figure 2.8: Power state machine of a typical microprocessor, showing high and low power states with state transition costs.

have lesser latency than the low power states. The transition latency from low power states to high power states is also higher than the inverse transition. A typical PSM for a StrongARM SA-1100 microprocessor is shown in Figure 2.8 [5]. This abstraction model is valid not only for many on-chip components such as microprocessors and memories, but also disk drives, wireless network interfaces and displays [70, 176].

A set of interacting PMCs constitute a power managed system. The system consists of a central control that can decide the mode of operation of each PMC in the system based on the workload. The general control is done in software by the operating system. This overall controller is referred to as the power manager. A simple power managed system is shown in Figure 2.9. An observer keeps a tab on the amount of workload and the criticality of the tasks involved. It passes this information to the controller which then predicts if any of the PMCs can be relegated to a low power mode of operation and the workload can still be handled. It makes the decision based on the minimization of the overall cost of the system and shuts down parts of the system. The power manager can be implemented as a hardware block or a software routine or as a hybrid of both [107]. The overall system functions according to the DPM scheme selected.

The choice and realization of the DPM scheme requires both the modeling of the power/performance of each PMC as well as that of the workload. The former
Figure 2.9: Abstract structure of a system-level power manager.

is achieved by using the PSM as described earlier. The prediction of the workload is the heart of a successful DPM scheme. The models for workload prediction can vary in complexity and an inefficient prediction could ruin the overall performance of the system. The prediction schemes can be classified into two broad classes:

- Predictive and
- Stochastic.

We will discuss the two classes and their merits in the following sections.

2.4.2 Predictive Techniques

In most real world systems there is little knowledge of future events and DPM decisions have to be taken based on uncertain predictions. The rationale behind all predictive techniques is the exploitation of correlation between the past history of the workload and its near future in order to make effective predictions. Overprediction is a predicted idle time longer than the actual one and underprediction is an idle time shorter than the actual one. An overprediction results in a penalty in speed whereas underpredictions waste power. Good predictors should minimize
the number of mispredictions. Predictive techniques can be further classified into static and dynamic.

2.4.2.1 Static Techniques

*Fixed timeout* is the most common predictive policy which uses elapsed time as the observed quantity. If the elapsed time is greater than a predetermined value $T_{TO}$, then the system shuts off or goes into *idle* mode. The system remains off until a request is made by the environment that signals the end of the idle period. The critical design decision, however, is to select the value of $T_{TO}$ [111]. Timeouts have two main advantages: they are general and their safety can be improved by just increasing the timeout intervals. The major disadvantage is that they trade off efficiency for safety. Large timeouts represent large underpredictions which mean lost savings in power.

The first disadvantage of timeouts can be improved by using a technique called *predictive shutdown* [65]. Srivastava *et al.* propose a non-linear regression equation obtained from past history to make prediction of the next idle time [174]. The format of the nonlinear regression is decided heuristically and the coefficients are determined by standard curve fitting methods. The main limitations of this approach are (1) the lack of automatic way to decide the type of regression equation and (2) offline data collection and analysis are required to construct and fit the regression model.

Another method proposed by Srivastava *et al.* is based on a *threshold*. The duration of the busy period immediately preceding the current idle period is observed. If the idle period is greater than a threshold, the system shuts down. This is applicable to systems with bursts of busy periods and long idle periods [174].

Hwang *et al.* address the second major limitation of the simple timeout policy, namely the performance penalty that is always paid on wakeup [82]. To achieve lesser performance penalty at wakeup, the PM performs a wake-up when the predicted idle time expires even if no requests are being sent. This decreases the delay of the first incoming request upon wakeup but suffers from power wastage
if the threshold is under-predicted.

2.4.2.2 Adaptive Techniques

Static predictive techniques assume a certain workload and hence are ineffective in systems with varying workload leading the way to adaptive techniques. Adaptive prediction techniques make use of recent history of the workload and dynamically change the wakeup/shutoff time. In the work by Krishnan et al., a set of timeout values is maintained and each timeout is associated with an index indicating how successful it would have been [92]. The policy chooses, at each idle time, the timeout that would have performed best among the set of available ones.

Helmbold et al. present a policy that keeps a set of candidate timeouts and assigns a weight to each timeout, based on how well it would have performed on an optimum offline strategy for past requests [76]. The actual timeout is achieved by a weighted average of all of the candidates.

Douglis et al. introduced a technique that keeps just one timeout value but increases it if there are too many shutdowns [51]. The timeout is decreased if some shutdowns can be tolerated. Workload prediction accuracies can be increased by customizing the policy to a specific class of workload. Adaptive shutdown policies tuned for hard disk drives produce significant power savings [109, 110].

2.4.3 Stochastic Control

Policy optimization is an optimization problem under uncertainty. Predictive approaches address workload uncertainty but they assume a two-state model with deterministic response and transition times for the system. For systems with PMCs of more than two states, stochastic techniques have been proposed. Policy optimizations should determine not only when to transition, but also which state to transition to. Power management optimization has been studied within the framework of controlled Markov processes. In this flavor of stochastic optimization the system is modeled as a Markov chain. The model consists of the following:
• A service requestor (SR) having a state set $R$, which models the arrival of service requests for the system (i.e., the workload).

• A service provider (SP), which is a controlled Markov chain with $S$ states that models the system as a whole. The states represent operating modes and their transitions are probabilistic. These probabilities are controlled by the power manager.

• A Power manager (PM), which implements a function $f : S \times R \to A$ from the state sets of SR and SP to a set of possible functions $A$. Such a function is an abstract representation of the decision process. The PM observes the system and the workload, takes a decision and issues a command to control the future state of the system.

• Cost metrics, which associate power and performance values with each system command pair in $S \times R \times A$.

Paleology et al. describe a Markov model specialized by assuming a finite state set and discrete time. Continuous Markov models have also been studied [169, 170]. There are static as well as adaptive stochastic control techniques. The static techniques model the system with precharacterized workloads and state sets $S$ and $R$. The adaptive techniques use learning algorithms to adapt to the probabilities during the execution of the system.

2.4.4 Implementation of DPM

In this section we address how different DPM schemes can be implemented in circuits and systems. Power consumption in clocked digital components (microprocessors, controllers, etc.) is roughly proportional to the clock frequency and the square of the supply voltage. Power can be saved by reducing the clock frequency or by reducing the supply voltage or both. We will look at them separately.
2.4.4.1 Clock Gating

This is the most common technique for power management. If any components are identified as idle by the power manager, then the clock to these components is shut down by clock gating logic. When the clock is not active, the components do not switch and hence save power. This is very useful for registers that are mostly idle. Clock gating has been implemented in many processors [45, 67, 161]. Clock gating has very small performance overhead. The clock is stopped by asserting a signal known as the clock freezing signal. The clock can be restarted by simply deasserting the clock freezing signal. There have been many CAD tools developed that design the idleness detection hardware that generates the clock freezing signal [4, 11, 142, 162, 183]. These circuits detect idleness based on certain schemes and shut off the clock for one or more components.

The main challenges in clock gating are: (1) to construct an idleness detecting circuit which is small and accurate and (2) to design gated clock distribution circuitry that introduces minimum routing overhead [141].

2.4.4.2 Supply Shutdown

Clock gating does not totally eliminate power consumption. Leakage power is dissipated even when the clock is disabled and can be a significant power wastage. Optimal power shutoff for idle components can be achieved by shutting off the power supply. An advantage of this approach is the applicability to all kinds of components such as digital, analog, sensors, etc. A major disadvantage is the added wake-up time, which is higher than the clock gating technique.

2.4.4.3 Multiple and Variable Power Supplies

DPM is also applicable not only to components that are idle but also to those components whose performance needs vary with time. The implementation can be based on a slowdown of the non-critical components at runtime. The slowdown is achieved by lowering the supply voltage of the non-critical components.
This dynamic variance of voltage based on performance requirements is known as dynamic voltage scaling. Varying the voltage continuously is a design issue and hence most of the implementations have components varying between two fixed supply voltages [194]. Nielsen et al. designed self-timed circuits with variable input voltage [140]. Similar approaches for synchronous circuits have also been implemented [32, 181, 193].

Chandrakasan et al. [31, 34, 35, 36] propose a first-order theory to estimate the dependencies of circuit styles and architectures on voltage scaling. They propose architecture driven voltage scaling, which decreases the power consumption but essentially keeps the overall delay of the system constant. They achieve this by introducing parallelism into the architecture. Consider the example of a datapath with a tolerable worst-case (lowest) frequency $f$ Hz and a supply voltage $V$ Volts. The dynamic power consumed would amount to:

$$P_{\text{dynamic}} = CV^2 f$$

Now if we duplicate the datapath into two separate datapaths with the same combined throughput, each of them could perform at frequency $f/2$. By their first order theory we can reduce the supply voltage to $V/2$ and since the capacitance $C$ is doubled by duplication, we have the new expression for power as:

$$P_{\text{dynamic}} = 2C \frac{V^2}{4} \frac{f}{2}$$
$$P_{\text{dynamic}} = 0.25 CV^2 f$$

Thus effectively the power has been reduced to 25% of its original value using an architecture driven technique. Increase in chip area is the price of this technique.

Voltage scaling increases the device reliability as it effectively reduces the ambient temperature of the chip and prevents devices from crossing their upper threshold temperatures [96]. Since the decrease in supply voltage decreases the driving capacity of the gate, switching time is increased. This also increases the short-circuit and leakage power dissipation of the circuit [96, 165, 171].
2.5 Architecture-RTL Level Low Power Techniques

The architecture level low power techniques are those implemented after the overall architecture of the system is fixed but the individual component architectures are not yet decided. The components are typically microprocessors or controllers whose functionality is decided by the system and software design levels. The implementation strategy is decided at the architecture design level. There are a number of techniques that can be used to make power-conscious decisions at this stage. We describe a few in this section.

2.5.1 Control Data-flow Graphs

The behavior of a system can be described as a control data flow graph (CDFG) with the nodes as the resources and the edges as operations. Certain transformations can be made to reduce power before these designs are implemented in the RTL or logic level. To be applicable such transformations must only modify the computational structure of the selected algorithm, while they preserve its original input/output behavior and to some extent its latency.

Chandrakasan et al. propose two distinct ways of optimization using behavioral transformations [33]. The first one consists of enabling the reduction of supply voltage through application of speedup transformations, such as retiming, pipelining, algebraic manipulations and loop restructuring. The second approach has a target of minimizing the effective capacitance through transformations that increase the utilization of system resources. Fewer and smaller computing elements provide better power performance. The easiest way to reduce the total switching capacitance consists of reducing the number of operations in the CDFG. Unfortunately, reducing the number of operations adversely affects the system performance.

Consider the example CDFG shown in Figure 2.10. This CDFG implements a second order polynomial. The one on the left is a more straightforward implementation. It requires the use of two multipliers and two adders. The transformed
Figure 2.10: Illustration of computation of a second-order polynomial using adders and multipliers in a circuit. (a) Unoptimized version and (b) optimization using control data flow graph transformations.

The implementation shown in Figure 2.10(b) is obtained by a simple algebraic transformation. The new CDFG requires only one multiplier and two adders. Thus the simple transformations can be used to reduce the overall power consumption of the system.

Multiplications usually take more time than additions and hence *strength transformations* are used to substitute adders for multipliers wherever possible [33]. But this technique has a drawback of increasing the overall critical path length. Conversion of multipliers to combinations of shifters and adders is also a useful transformation. The impact of various transformations depends on the specific situation to which they are targeted. Hence, a fully automatic procedure that drives the optimization process is infeasible.

### 2.5.2 Operation Scheduling

The goal of an operation scheduling algorithm is to associate each primitive operation appearing in the CDFG with the time interval in which the operation is to be performed to satisfy the design constraints. Several attempts have been made to modify traditional scheduling algorithms for power minimization. Musoll and Cortadella have proposed a measure of switching activity occurring at the inputs of the functional units [131]. They select certain nodes from the CDFG where the input operands do not change for consecutive operations. Placing such nodes as close as possible to the scheduling hardware can significantly reduce the
switching capacitance. They have also proposed a set of CDFG transformations such as loop interchange, operand reordering and operand sharing that may help in minimizing the activity at the inputs of a functional unit [130].

A power conscious loop folding technique has been proposed by Kim and Choi [87]. This transformation enables the detection of common input operands hidden inside loops of CDFGs. Monteiro et al. propose the idea of scheduling at a lower level of abstraction [128]. The scheduling attempts to assign the operations involved in determining and controlling the flow of data to the earliest possible time intervals. This allows one to establish the functional units that are absolutely necessary for a computation. The other functional units can be disabled or run at a slower speed.

2.5.3 Resource Allocation

Once the scheduling is complete, a resource allocation procedure assigns registers and functional units to variables and operations in the CDFG. This step also specifies the interconnects and buses to be used for the operations. If the registers, functional units and interconnects are allocated one after the other it is called serial allocation. Such allocating algorithms do not take into account the switching at the inputs of the various functional units and hence may waste energy.

Chang et al. proposed graph based algorithms for register allocation in non-pipelined designs and module allocation for functionally pipelined designs [37, 38]. This technique relies on accurate computation of the probability density functions at the inputs of various resources, given the probability distributions at the inputs of the system.

Serial allocation may result in sub-optimal solutions requiring more interconnections than available. In such cases, simultaneous allocation can be used. Raghunathan et al. describe a combined minimization of the total circuit capacitance and switching activity at the inputs of the registers and the functional
units [154]. The first objective is reached by minimizing the total number of resources in the final design implementation and by keeping under control the required amount of steering logic and interconnect. The minimization of input activity is obtained through exploitation of the correlation that may exist between the data words traveling and being stored within the circuit.

Bhatia et al. propose the concept of a compatibility graph (CG) [23]. The CG is an undirected weighted graph that has as many nodes as there are variables and operations in the CDFG. Edges in the CG connect pairs of compatible nodes, i.e., nodes that can be mapped onto the same resources. Edge weights reflect the potential savings that could be achieved in the architectural implementation if the pairs of variables or operations connected by the edges were assigned to the same hardware resources. After building the CG, the allocation program groups the compatible nodes starting with the nodes with higher global weights. The allocation and binding algorithm does not guarantee a global optimum but achieves a power savings of up to 33%.

Other low power allocation strategies have been proposed in the recent literature [62, 66, 122]. In the strategies discussed above operation scheduling has to precedes the resource allocation step. Novel approaches where the resource allocation is done before the scheduling have been proposed [46, 61]. Techniques where the scheduling and resource allocation are done simultaneously have also been proposed [155].

2.5.4 Bus Encoding

It is known that the bus capacitances are several orders of magnitude higher than the internal switching capacitances. Consequently, a considerable amount of power can be saved by using efficient bus encoding algorithms. The bus invert code is a simple encoding scheme [175]. The Hamming distance of two consecutive patterns is computed. If the Hamming distance is greater than $N/2$ for the current $N$-bit pattern, then the bit is transmitted with inverted polarity. Obviously, a
redundant invert line is needed in the bus to signal the receiving end of the polarity of the signal.

Su et al. have proposed a Gray code encoding strategy. This code achieves its asymptotic best performance of a single transition per emitted address when infinite streams of consecutive addresses are considered and it is optimum only in the class of irredundant codes [124, 178]. If some redundancy is allowed then another encoding known as zero activity encoding or T0 coding is more effective [14, 15]. This technique requires an extra line called INC, to signal when a pair of consecutive addresses is written to the bus. When INC is high, the current value is frozen and the new address is computed directly by the receiver. When INC is low the bus operates normally.

Musoll et al. propose a working zone code based on the observation that many programs access multiple data arrays [131]. The accesses to each array are mainly in sequence, but are often interleaved. This sequentiality is destroyed in the bus. The working zone scheme restores sequentiality by storing the reference address of each working zone on the receiver side and by sending only the highly sequential offsets. Whenever the data access moves to a new working zone, this information is communicated to the receiver. The receiver then changes the reference address and the offset transmission can resume. Although this scheme is more flexible, it relies on the assumption that the data access is array based.

Benini et al. propose the beach code, which relies on temporal correlations between the patterns that are being transmitted [13]. The bus lines are grouped into clusters according to their correlations. An encoding scheme is generated for each cluster and each bit in each cluster is changed into the new bit configuration. The output of the transformation is an encoded stream for which the average number of bus line transitions between two successive patterns is minimized.

The architecture level techniques have been taken to a new level by the Crusoe microprocessor chip of Transmeta Corporation [64]. The idea behind the chip was to design an architecture with the compiler’s specification in mind. One offloads most of the functioning of the microprocessor to software, which allows
a streamlined hardware with about half the number of transistors as that of an
X86 microprocessor chip. Fewer transistors mean lower power. Another technique
used to reduce hardware by Transmeta was the use of virtual devices. They work
by using code morphing software to monitor the input and output instructions
sent to the device, then send those instructions to the virtual device instead.
Detailed description of both code morphing and virtual devices is beyond the
scope of this document.

2.6 RTL-Gate Level Low Power Techniques

Once the architecture and system level parameters have been decided, the hard-
ware blocks are generally synthesized using electronic design automation (EDA)
tools. There are a variety of algorithms proposed that can synthesize the design
with an objective of minimizing power. In this section, we will look at some of
the techniques used to reduce the power through control logic synthesis.

2.6.1 Control Logic Synthesis

High level synthesis produces a combined description of the data-path and the
control logic. The latter is in the form of a translation structure, whose most
familiar representation is a finite state machine (FSM) or a collection of FSMs.
The translation of such an FSM into a structural description presents an oppor-
tunity for reducing power. At this level of design a gate-level netlist is generated
from a state transition graph (STG). For designs with many latches, the STG can
be huge and the entire STG is not feasible to construct. Hence, symbolic tech-
niques such as binary decision diagrams (BDD) have been used [24]. The BDD
can be subjected to transformations that reduce power. Among the modifications
are decomposition and restructuring. Decomposition techniques produce smaller
interconnected FSMs from a large FSM. The techniques can be based on algebraic theory or other procedures that identify sub-routines or coroutines [48, 71].
Shutdowns can be used for some FSMs that are not in use at that time [16].
Algorithms are available for state minimization of completely specified FSMs [105]. These algorithms can be used with an objective function of power minimization [95, 97, 98, 99]. The problem of encoding the STG has also received considerable attention. The idea is to use the transition probability of a given edge in the STG as a measure. The problem is then translated into a hypercube of suitable dimension so that arcs of high cost connect states of low Hamming distance. Many standard search techniques have been used for this optimization [10, 68, 143, 159, 180, 191].

2.6.2 Charge Recovery Techniques

In CMOS circuits, for every rising transition, the load capacitance $C_L$ is charged to $V_{dd}$ and the energy drawn from the supply is $C_L V_{dd}^2$. Only half of this energy is stored on the load capacitor and the remaining $0.5C_L V_{dd}^2$ is dissipated as heat in the resistor (i.e., the switching transistor). During a falling transition, the $0.5C_L V_{dd}^2$ stored on the capacitor is dissipated as heat. Thus, in a single cycle the entire energy of $C_L V_{dd}^2$ is dissipated as heat. Some techniques try to recover some of this energy. These techniques are called charge recovery or adiabatic techniques.

The main idea for these techniques is that the energy dissipated as heat is a function of the difference in the voltage of the supply and the signal nets. In conventional CMOS technologies the supply is constant and, hence, the difference is maximum during a signal transition. If we can make the supply follow the signal nets, the difference can be reduced. Thus, these techniques have a pulsating supply voltage, which is varied as a slow ramp [60]. The signals transition during the upward ramp of the signal. Since the voltage difference is reduced during switching, the circuits consume less power.

The problem with a single pulsating voltage supply is that all signals in the circuit need to transition at the same time. This does not allow gates to be cascaded. Roy et al. propose a technique where two complementary supplies are
used [160]. The concept is that the two signals are used as the supply and ground for a conventional CMOS gate. The operation is divided into precharge and evaluation phases. This technique is known as quasi-adiabatic design of circuits. This technique is shown to save up to 70% of the energy. The problem with these circuits is the low speed of operation. For greater power the ramp has to be slow, which slows down the circuit speed. Charge recovery techniques are useful for very low activity circuits where energy is more critical than speed.

Manne et al. propose a novel technique that recycles the charge in the circuit using a charge pump [114]. The idea is to move the slower adiabatic components away from the critical path of the circuit, thereby increasing the power of operation of the circuit. This is achieved by overlapping the adiabatic charge pump delays with the computing path logic delays. The ground-bound charge is collected in a large capacitor. This charge is then pumped into the virtual power supplies of the subsequent stages using a charge pump. Pylarinos et al. analyzed the performance of the charge pump circuits when multi-phase clocks are used in the circuit [149]. They show that a small mismatch in parasitic capacitance causes a large ripple in voltage levels.

2.7 Gate-Transistor Level Low Power Techniques

The gate-level netlist can be altered to reduce the switching capacitance. A variety of algorithms have been proposed of which we describe a few in this section.

2.7.1 Low Power Technologies

The gate-level netlist is mapped to transistor-level implementations. The logic gates can be realized by combining the transistors in different ways based on the application and the design specifications. Some of the popular implementations are:
• *Complementary Metal Oxide Semiconductor* (CMOS) technology, where the gates have a \( p \)-tree and an \( n \)-tree for each gate [86, 168, 182, 196]. The rising transition at a gate output is due to the switching of the transistors in the \( p \)-tree to create a path to the supply and the falling transition is due to the switching of transistors in the \( n \)-tree to create a path to the ground. This technology has very little short-circuit and leakage power consumption compared to other techniques and hence it is preferred in low power applications.

• *Domino Logic* is a variant of the CMOS technology, where only the \( n \)-tree is responsible for the switching [197]. The \( p \)-tree consists of a single \( p \)FET which is precharged to the supply voltage and the precharge circuit is followed by an inverter [91]. During the *evaluation* phase, the charge at the output is either discharged through the \( n \)-tree or left unaltered based on the logic value of the gate. The precharge voltage is stored at the output of the inverter, such that when gates are cascaded, the inputs of the subsequent stages do not change during evaluation phase. This technology also consumes low power as it has all the advantages of the CMOS technology and also has less leakage and less area due to fewer transistors in the \( p \)-tree.

There are other implementation technologies such as *Zipper CMOS*, *pass transistor logic*, etc., which have advantages in speed, but are not always low power [152, 197]. Once the technology is decided, the gates can be sized accordingly to target other power wastages such as *glitches*. Various techniques for reducing the glitch power are discussed in the following sections.

### 2.7.2 Path Balancing

As stated earlier, glitch power is a major component of overall power consumption and there are many techniques to reduce it. One of the earliest is *path balancing*. The glitches are produced due to a difference in arrival times of signals at gate inputs. The idea behind this technique is to prevent glitches from occurring
by balancing the delays of paths such that at any given gate the signals arrive at the input terminal at the same time. Consider the example circuit shown in Figure 2.11. The signals at the inputs of a gate arrive at different times, which would probably result in glitches. This behavior can be suppressed by restructuring the circuit as shown in Figure 2.12 [127]. However, as the next example illustrates, a simple restructuring of gates may not be sufficient for some circuits.

Wong et al., in their papers [198, 199] on wave pipelining, have described a method of balancing delays for circuits by inserting buffers at selected inputs of the gates. The addition of buffers is done such that it does not increase the critical delay of the circuit, but effectively eliminates spurious transitions. The buffers are inserted only in the fast paths of the circuit and since the slowest paths determine the speed of the chip, they are untouched. For example, the
Figure 2.13: Glitch behavior of a circuit where simple restructuring will not balance paths.

Figure 2.14: Path balancing of the circuit in Figure 2.13 by buffers.

circuit in Figure 2.13 cannot be balanced without the introduction of buffers as shown in Figure 2.14. However, the addition of buffers increases the switching activity of the circuit, which may offset the reduction in switching activity due to elimination of glitches.

In CMOS circuits the fanout load significantly affects the gate delay. To account for this, Kim et al. [90, 89] developed a delay buffer insertion method using a chain of buffers, thus eliminating the need for merging common buffers. The problem of delay buffer insertion on a multiple fanout net is simplified by assuming that only one size of delay buffer is used. This method permits a quick estimation of the number of buffers needed to balance the circuit and can therefore be used in heuristics for logic restructuring [27].

Kim et al. calculate the minimum and maximum arrival times at the inputs of the gate for efficient path balancing [88]. Gate sizing is done without inserting buffers. Then, buffers are inserted using integer linear programming (ILP) locally but not globally. Some glitches are still left in the circuit.

Path balancing has been used for some earlier multiplier architectures to increase computational speed by reducing functional activity and also eliminate
hazards [146]. Another method is to use self-timed design in which a combinational element would not compute until all signals have stabilized. This requires additional circuitry to generate the timing or enable completion signals for various stages of logic [103, 151, 173]. The added circuitry reduces the power saving.

2.7.3 Hazard Filtering

This method eliminates glitches in the circuit using inertial delays. Inertial delay is the interval that elapses after an input change before a gate can produce an output change. When the time between successive edges in the input signal is less than the inertial delay of the gate, the effects of these edges on the output are suppressed. This is known as the filtering effect of the gates [1]. All MOS gates have a non-zero delay called the inertial delay and it suppresses signal pulses that are of smaller width than the inertial delay from passing through the gate. Instead of balancing the delays of the gate inputs to be exactly equal (as in delay balancing), this method evaluates the differential delay of inputs and increases the inertial delay of the gate to exceed the differential delay such that the glitch will be suppressed within the gate.
Figure 2.16: Effect of gate delay on energy dissipation.

Consider the example inverter shown in Figure 2.15 with the total capacitance at the output node as $C$ and the short circuit impedance $R_p$. When an input pulse as shown is incident at the input, a charging current $i(t)$ flows through the upper transistor charging the capacitor $C$. This raises the output voltage until time $t = \tau$ when the rising edge occurs at the input. After that $C$ discharges through another short circuit impedance $R_n$. Assuming $R_p = R_n = R$, the energy consumed by the gate is [197]:

$$E(\tau) = CV_{dd}^2 (1 - e^{-\frac{\tau}{RC}}) \quad (2.3)$$

For hazard filtering, we increase the time constant $RC$ of the gate and thus increase the charging time needed for the capacitor to reach $V_{dd}$. When the time constant is large, there is practically no energy dissipation in the gate due to this short glitch of length $\tau$. Energy dissipation for various values of the time constant with respect to the width of the pulse $\tau$ is shown in Figure 2.16 [1].

A simulated example of a full adder circuit shows that the method reduced power by 42% [1]. The glitch-free circuit may have gates that have been slowed down compared to their original speed. Unless the slowed down gates are in the
critical path, there is very little reduction in the overall speed of the circuit.

2.7.4 Gate Sizing

Gate sizing is defined as assigning load drive capabilities to the gates of a Boolean network, such that a given delay limit is obeyed, and the total cost in terms of power consumption is minimal [19]. Similar to the hazard filtering technique, this method also uses the filtering effect of the gates but achieves the increase in inertial delay by resizing the gates in the circuit. The delay model frequently used for solving this problem is [19]:

\[ \tau_{gate} = \tau_{int} + c \cdot C_{load} \] \hspace{1cm} (2.4)

\[ C_{load} = C_{wire} + C_{in} \] \hspace{1cm} (2.5)

where

- $\tau_{gate}$: delay of the gate
- $\tau_{int}$: internal delay of the gate
- $C_{load}$: capacitive loading experienced by the gate
- $c$: a constant
- $C_{wire}$: routing capacitance of the output net of the gate
- $C_{in}$: capacitive loading due to gates in the fanout cone.

Berkelaar and Jacobs [20] use a parameter called the speed constant to reformulate the delay model but this converts the problem into the non-linear domain. Berkelaar et al. [19] have tried to solve it as an LP problem by using a piecewise linear simulator and have published results for all ISCAS benchmark circuits. This method was found to be faster than the LP method for circuits with less than 1000 gates but was not very useful for larger circuits. Berkelaar and Jacobs [20] have tried to formulate the gate sizing problem as an LP problem but
the non-linear nature of their delay model posed problems in finding the global optimum solution. The details of this are dealt with in the next chapter.

Berkelaar and Jacobs [21] also tried gate sizing under a statistical delay model [17, 18] but the resulting problem was again non-linear and took the solver program considerable amounts of time and resources to solve. That method could not be used for circuits with more than 1000 gates. Hashimoto et al. have used the statistical approach for gate sizing [75]. They estimate the gate transition densities based on probabilistic techniques [104]. The glitches are separated into *generated glitches* and *propagated glitches*. They use a greedy approach to optimize the gates that produce the glitches rather than the ones that simply propagate them. They extend the work to cell-based designs where the allowable gate delays are fixed [74].

### 2.7.5 Transistor Sizing

Transistor sizing [22, 57] is similar to gate sizing but the essential difference is that in gate sizing all the transistors of a gate are sized equally but in transistor sizing they can have different sizes. Traditionally, transistor sizing is done to reduce the area and delay of a VLSI chip [172].

Datta et al. [43] have considered transistor sizing for low-power and high-performance of static CMOS circuits. The transistors on the critical paths of the circuit are sized to obtain a better power and delay performance. To improve the switching speed and the output transition characteristics of a particular circuit block on the critical path, one may seek to increase the widths of the transistors in the block. This results in an increased current drive and better output transition time. Faster input/output transition times imply lower rush-through current and hence smaller short-circuit power dissipation. Even though the delays of a particular block and its succeeding block are reduced, an increase in transistor widths increases the capacitive loading of the preceding block and may severely affect its power and delay. Thus, the issues regarding delay and power dissipation
are fairly interlinked [160]. The algorithm described [43] minimizes the delay, the area and the power dissipation of a circuit by optimizing the sizes of the gates on the critical paths of the circuit. However, the constraint set for this model becomes non-linear and hence the solution for large circuits becomes tedious and complex.

2.7.6 Linear Programming (LP) Approach

A linear program determines a set of variables such that an objective is minimized under given constraints [58]. To eliminate the glitching power from a circuit the inertial delay of the gate has to be altered as dictated by the hazard filtering technique. But the altering has to be done without affecting the critical path of the circuit and also taking into account the change in delay of the gates in the fanout cone whose delay will effect the delay of the gate in question. There can be an infinite number of solutions but we are only interested in the global optimum for obvious reasons. Hence, the problem now becomes an optimization problem. The linear programming model guarantees the global optimum for every feasible solution of the problem.

A variety of LP models have been investigated to express the glitch removal problem (see chapter 3). Agrawal et al. [2] have described a gate level formulation for the problem. The variables consist of inertial delays of gates and delays of buffers that may need to be inserted in the circuit for complete glitch removal. The constraints are written by path enumeration from the gate inputs to the primary inputs (PIs). Ideally, the number of buffers should be minimized holding the overall circuit delay below specified limit. In practice, to keep the expressions linear the total delay of the buffers in minimized.

2.8 Summary

This chapter has introduced the field of low power design. We described various components of power dissipation in a circuit. There have been techniques to
reduce power consumption of circuits at every level of design. These techniques can be grouped according to the level of design they are effective at. Software-level low power techniques try to schedule the instructions on a processor to minimize the power. System-level power management techniques try to reduce power by shutting off or slowing down unused or non-critical components in a system. Architecture-level techniques address the power wastage involved in scheduling and allocation of resources. Power can be reduced by intelligent synthesis of the control logic. Gate-level techniques reduce the overall switching activity of the circuit by resizing and efficient delay scheduling. Our proposed technique falls into this final category and will be described in detail later.
Chapter 3
Prior Work: Techniques for Power Estimation

Power estimation is the problem of estimating the average and peak power dissipation in a digital circuit [134, 146]. Accurate and efficient power estimation during the design phase is required in order to meet the power specifications of the chip. Hence the power estimation is done at various levels of design. In this chapter we describe power estimation techniques at various levels of design. We start with the software-level power estimation, followed by some behavioral-level techniques. We move on to RT-level techniques and then finally to the gate and lower levels.

3.1 Software-Level Power Estimators

At this stage of design little is known about the actual capacitances and switching events of the eventual design of the CPU. It is advantageous to have an estimate of the various instructions that would be executed on the processor. This overall cost of each microinstruction is determined by power macromodeling. Sato et al. characterize the power cost of a CPU by estimating the average capacitance that would be charged or discharged when the CPU block is activated [163]. Su et al. use the switching activity on buses to estimate the power consumption of the microprocessor [177].

Tiwari et al. estimate the power consumption based on actual current measurements [184]. The total power of the microprocessor is given by:

\[ \text{Energy} = \sum_i (BC_i N_i) + \sum_{i,j} (SC_{i,j} N_{i,j}) + \sum_k OC_k \]

where \( \text{Energy} \) is the total energy dissipation of the program. The first term
is the summation of the base energy cost of each instruction ($BC_i$ is the base energy cost and $N_i$ is the number of times instruction $i$ is executed). The second term accounts for the circuit state ($SC_{i,j}$ is the energy cost when instruction $i$ is followed by $j$ during program execution). The third term accounts for the energy contribution $OC_k$ of other instruction effects such as stalls and cache misses during the program.

Hsieh et al. present an approach called profile driven program synthesis to perform power estimation of high performance CPUs [79]. This approach consists of the following steps.

- Perform architectural simulation of the target microprocessor under the instruction trace of typical application programs.
- Extract a characteristic profile including parameters such as the instruction mix, instruction/data cache miss rates, branch prediction miss rate, etc., for the microprocessor.
- Use mixed integer linear programming and heuristic rules to gradually transform a generic program template into a fully functional program.
- Perform RT-level simulation of the target microprocessor under the instruction trace of the new synthesized program.

This approach reduces the estimation time of the RT-level power estimation by three orders of magnitude over accurate simulation of the flattened netlist.

### 3.2 Behavior-Level Power Estimation

At the behavior level no information is available about the gate-level structure of the design components and, hence, one must resort to abstract notions of physical capacitance and switching activity to predict power dissipation in the design. These techniques can be classified into three broad categories.
3.2.1 Information Theoretic Models

These techniques depend on measures of activity to obtain quick power estimates [115, 139]. Entropy of a sequence of applied vectors characterizes the randomness or uncertainty present and thus is intuitively related to switching activity per vector transition. If the switching activity is high, it is likely that the sequence is random and the entropy is high. It has been shown that the average switching activity per vector transition of a bit is upper bounded by one half of its entropy. The power dissipation is given by:

\[ \text{Power} = 0.5 V^2 f C_{\text{tot}} E_{\text{avg}} \]

where \( C_{\text{tot}} \) is the total capacitance of the logic module and \( E_{\text{avg}} \) is the average activity of each line in the circuit, which is, in turn, approximated by one half of its average entropy \( h_{\text{avg}} \). The average entropy is computed by abstracting information from the gate level implementation. It is assumed that the word-level entropy per logic level reduces quadratically from circuit inputs to circuit outputs. Based on these assumptions two models can be obtained.

Marculescu et al. derived a closed form expression for the average line entropy for the case of a linear gate distribution, i.e., when the number of gates in the circuit block scales linearly with the sum of the number of circuit inputs \( n \) and outputs \( m \) [115]. The expression gives \( h_{\text{avg}} \) as a function of the entropies \( h_{\text{in}} \) and \( h_{\text{out}} \), which are the bit-level entropies of the input and output vectors, respectively.

Nemani et al. obtain another model by using word-level entropies [139]. The total capacitance \( C_{\text{tot}} \) can be obtained by traversing the circuit netlist and summing the gate loadings. Wire capacitances are estimated by wire-load models. Cheng et al. propose an estimation model where the capacitance \( C_{\text{tot}} \) is a function of the difference of the input and output entropies of the circuit block [39]. This tends to be too pessimistic for circuits with large numbers of PIs. Ferrandi et al. propose an alternative estimate based on the ordered binary decision diagrams representing the circuit [55]. The coefficients are obtained empirically by doing linear regression analysis on the total capacitance values for a large number
of synthesized circuits.

Entropy models for the control circuitry are proposed by Tyagi et al. [192]. The circuit is divided into \( S \) state sets with \( T \) states in each set. The technique provides three entropy bounds on the average Hamming distance for vectors applied to each state. These bounds are used to estimate the overall power.

### 3.2.2 Complexity Based Models

These models relate the power dissipation to circuit complexity. Parameters such as the number and types of arithmetic operators, the number of states in the controller description, etc., are measures of the circuit complexity. Most of the proposed algorithms work on the assumption that the circuit complexity can be estimated by the number of equivalent gates. This information is generated by using analytical predictors or by pre-characterized high-level libraries. Muller-Glaser et al. propose a chip estimation system that computes the average power of a logic module as:

\[
\text{Power} = f N (\text{Energy}_{\text{gate}} + 0.5V^2C_{\text{load}})E_{\text{gate}}
\]

where \( f \) is the clock frequency, \( N \) is the equivalent gate count for the component, \( \text{Energy}_{\text{gate}} \) is the average energy of a single gate per logic transition, \( C_{\text{load}} \) is the average capacitive load of an equivalent gate and \( E_{\text{gate}} \) is the average output activity for an equivalent gate per cycle [129]. \( C_{\text{load}} \) is estimated statistically based on the average fanout count in the circuit and custom wire-load models. \( E_{\text{gate}} \) is dependent on the functionality of the module. The data is precharacterized and stored in a library and is independent of the implementation style. This is an implementation-independent and data-dependent power estimation model.

Nemani et al. present a high-level estimation model for predicting the area of an optimized single output function [138]. The model is based on the assumption that the area complexity of a Boolean function \( f \) is related to the on-set and offset of the function. Based on the area measure they come up with a capacitance
estimate, which is used for power estimation. This work was extended to area estimation of multiple output functions [137].

Complexity based prediction algorithms have been proposed for controller circuitry [101]. These techniques provide quick estimation of the power dissipation in a control unit based on the knowledge of its target implementation style, number of inputs, number of states and so on. The estimates are derived from curve-fitting by least square error analysis on real data.

### 3.2.3 Synthesis Based Models

One approach to behavior-level estimation is to assume some RT-level template and produce estimates based on that assumption. This approach requires development of a quick synthesis capability, which makes some behavioral choices. Important behavioral choices include type of I/O, memory organization, pipelining issues, synchronization scheme, bus architecture and controller design. After the RT-level structure is decided, the power is estimated by using any of the RT-level techniques described later. The other relevant information such as the number of operations of a given type, bus accesses and memory accesses are captured by static profiling based on stochastic analysis of the behavioral description [37]. Dynamic profiling can also be used. It is based on direct simulation of the behavior under a typical input stream [93, 120].

### 3.3 RT-Level Power Estimation

Most RT-level power estimators use regression based, switching capacitance models for circuit modules. The technique of characterizing a circuit block for power for different input configurations is called power macromodeling. These techniques can be broadly classified into two categories based on the methodology of accumulating the data. In this section we will look at both techniques.
3.3.1 Regression Based Models

A typical RT-level power estimation flow consists of the following steps:

1. Characterize every component in the high-level design library by simulating it under pseudorandom data and fitting the multivariate regression curve to the power dissipation results using a least squares fit [7].

2. Extract the variable values for the macromodel equation either from static analysis of the circuit structure and functionality or by performing a behavior simulation of the circuit.

3. Evaluate the power macromodeling equations for high-level design components, which are found in the library, by plugging the parameter values into the corresponding macromodel equations.

4. Estimate the power dissipation for random logic of these components at the gate level. This can be done by simulation, probabilistic analysis or by statistical methods [40, 50, 80, 81, 116, 117, 118, 119, 133, 188, 189, 190, 206]. These methods are described in greater detail in Section 3.4.

The macromodel for the component may be parameterized in terms of the input bit width, the internal architecture and the supply voltage level. This method works only if the internal structure of the components is known. If the gate level netlist is not finalized by this stage, Step (1) can be replaced by an analysis of past design models for the component followed by appropriate process technology scaling [9]. The macromodel equation could also be replaced by a lookup table with necessary interpolation equations [8].

Now let us look at some of the macromodels. The simplest macromodel is called the constant type model used in a power factorization technique [147]. This technique uses an experimentally determined weighting factor to model the average power consumed by a component per input change. The power dissipation of an $n \times n$ bit integer multiplier can be written as:

$$Power = 0.5V^2n^2Cf_{activ}$$
where $V$ is the supply voltage, $C$ is the capacitive regression coefficient and $f_{\text{activ}}$ is the activation frequency of the module. The weakness of this technique is that it does not account for the data dependency of the power dissipation. For example, if one of the inputs of the multiplier is always one, we expect the power to decrease but this model cannot account for that. It considers both inputs to be equally random.

Landman and Rabaey propose a stochastic power analysis technique, which is based on an activity sensitive macro-model known as a data-bit type model [100]. The higher-order bits in a data stream have temporal correlation whereas the lower-order bits behave more randomly. The module is thus characterized by its capacitance models in the sign and white noise bit regions. Extending this idea, one can use a bitwise data model and the macromodel becomes:

$$\	ext{Power} = 0.5 V^2 f \sum C_i E_i$$

where $n$ is the number of inputs to the module, $C_i$ is the regression capacitance for input pin $i$ and $E_i$ is the switching activity for the $i$th pin of the module. This equation can produce accurate results by taking spatio-temporal correlation coefficients into account. This will, however, increase the number of variables in the macromodel equation.

Liu and Svensson describe a parametric power model where the power is expressed as a function of the implementation styles [106]. For example, the power dissipation of a memory unit consisting of $2^{n-k}$ rows and $2^k$ columns can be split into following components:

- Power consumed by the $2^k$ cells during one precharge or evaluation.
- Power consumed by the row decoder
- Power needed for driving the selected row
- Power consumed by the column select part
- Power dissipated in the sense amplifier and the readout inverter.
These parameters can be obtained from previous designs or by simulation. These will be used in the macromodel for the entire memory characterization. The techniques described thus far estimate the average power. But it might be useful to estimate the peak power for purposes such as reliability testing, package determination, etc. In such cases cycle accurate power estimators are required.

Mehta et al. propose a pattern accurate estimator using an assumption that closely related patterns have similar power dissipation [123]. Each input pattern is mapped into a cluster and then a table lookup is performed to obtain the corresponding power estimates from precalculated power characterization data for the cluster. The weakness of this approach is that a large number of clusters increase the simulation time. The assumption of correlation between Hamming distance of the vectors applied at the PIs and power consumption may not be valid in the presence of mode changing bits, i.e., when a bit change occurs the functionality of the block changes drastically.

Addressing these problems, Wu et al. describe a cycle accurate macromodel generator based on statistical sampling [200]. The regression analysis is combined with appropriate statistical tests for macromodel variable selection. The most power critical values are added to the variable set, enabling the prediction of the power value as well as a confidence measure for the predicted value. Qiu et al. extend this work to capture important first-order temporal and spatial correlations at the circuit inputs [150]. The equation forms and models used are model specific and cannot be used globally. Power estimates using this method are within 5-10% of the actual values at the end of the design process.

3.3.2 Sampling-Based Models

RT-level power estimators can be implemented in the form of a power cosimulator for standard RT-level simulators. The cosimulator is responsible for collecting input statistics from the output of the behavioral simulator and producing the power value at the end. If the cosimulator is invoked by the RT-level simulator
every simulation cycle to collect activity information in the circuit, it is called census macromodeling. It is like a subroutine call from the main simulator for every component access.

Evaluating the macromodel equation each cycle during simulation is actually a census survey. The overhead of data collection and macromodel evaluation can be high. To reduce the run time overhead, Hsieh et al. use simple random sampling to select a sample set and calculate the macromodel equation for the vector pairs in the sample set [78]. The sample size is determined before simulation. The sampler macromodeling randomly selects $n$ cycles and marks those cycles. When the behavioral simulator reaches the marked cycle, the macromodeling invokes the behavioral simulator for the current input vector and previous input vectors for each module. The input vectors are collected only in these marked cycles. In this manner, the sampling overhead of the census every clock cycle is reduced. These techniques estimate the power within a 1% error.

3.4 Gate-Level Power Estimation

As noted in Section 2.2.1, the dynamic power dissipation of a gate can be calculated from [197]:

$$P_{dyn} = \frac{1}{2} C_{load} V_{dd}^2 f D$$

(3.1)

where

- $P_{dyn}$: dynamic power dissipation in the gate
- $C_{load}$: load capacity of the gate
- $f$: clock frequency
- $V_{dd}$: supply voltage
- $D$: transition density or activity of the output of the gate.

Transition density of a gate $x$ is the average number of transitions per second at gate $x$ and is denoted by $D(x)$. For a given chip all quantities except the
A Large Set of Input Patterns

![Diagram showing a circuit simulator connected to a large set of output waveforms, which in turn are connected to an average tool, followed by power estimation.] 

Simulation Techniques

Probabilistic Techniques

Figure 3.1: Simulation and probabilistic techniques of power estimation.

transition density are fixed. It is the estimation of transition density that has resulted in various techniques of power estimation. This section tries to familiarize the reader with a few of these techniques.

A lot of work has been done recently on the estimation and reduction of leakage power in a circuit and an interested reader is directed to the literature [52, 83, 94].

3.4.1 Simulation-Based Techniques

The simplest and most direct power estimation can be done by circuit simulation and monitoring the power supply current waveform as shown in Figure 3.1. The transition density can be directly gotten by keeping track of the overall transitions of each gate during simulation and can easily be averaged out over the given set of input patterns. This transition density can be used for estimating the power consumption for any vector transition at the PIs, even those that were not a part of the initial vector set. The advantages of this technique lie in its accuracy and generality. It can be used to estimate the power of any circuit, technology or design style. But the simulation results are directly related to the input patterns given to the PIs of the chip. Thus, this method is known as strongly pattern
dependent [85, 204]. This disadvantage is serious as most of the estimation is done in the early design stages when the input patterns may not be available. The power estimate would be erroneous if some of the patterns for which the estimate has been done would never occur in the circuit. Hence, the problem is finding a set of input patterns that are typical of the circuit during its normal mode of operation. If these typical patterns cannot be assessed for the circuit, then the designer would have to turn to more complex techniques of estimation, which follow.

There are circuit-level simulator power estimators available as commercial tools. The most important ones are the circuit simulators, SPICE and PowerMill. SPICE [132] operates by creating a large matrix of nodal currents using Kirchhoff's current law. This matrix is then solved for different time instances for obtaining the current at every node. Each component is modeled as a set of equations relating the inputs to the outputs. It accurately takes into account the non-linear relations between components, which cannot be captured by high-level tools [6, 54, 205]. Also, it can accurately measure short-circuit and leakage currents. Power measurement in SPICE is not direct. However, accurate methods for power measurement using SPICE have been reported [168]. It is a pattern-dependent power analysis tool.

The main drawback of SPICE is that it cannot be used for large circuits or chips due to the time-consuming nature of the simulator. PowerMill [47] applies an event-driven simulation algorithm to increase computational speed by two or three orders of magnitude over SPICE. It uses table lookup to determine the terminal current of the device [6]. PowerMill can also identify the hot spots (areas that consume more dynamic power) and trouble spots (which consume unexpectedly large amounts of leakage power). This allows the designer to resize the circuit to reduce power in these hot spots.
3.4.1.1 Event-driven Power Estimation

To decrease the computational time by sacrificing the accuracy, a logic level event-driven technique can be used. Hsiao *et al.* describe a power estimation technique that uses event-driven simulation to estimate the power consumption in a circuit [77]. The transitions at every gate are counted and weighted with the number of fanouts of the gate. The power consumption is given by:

\[
Power = 0.5V_{dd}^2 C_f \sum_{\text{All gates}} T_i \times F_i
\]

where \(V_{dd}\) is the supply voltage, \(T_i\) is the total transitions at gate \(i\), \(F_i\) is the number of fanouts at gate \(i\) and \(C_f\) is the nominal capacitance per fanout. The idea is that the load capacitance offered at each gate output is proportional to the number of fanouts at the gate output. This method is a fast approximation of the overall power of the circuit. However, the load capacitance is not exactly proportional to the number of fanouts as different sized cells might be used for the gates in the layout. This method does not account for the routing capacitance as well. These disadvantages can be eliminated by weighting the number of transitions using the actual extracted capacitances from the layout. Once the circuit is designed at the layout level, the gate capacitance \(C_i\) at every gate \(i\) and the routing capacitance at every signal net is extracted. Using this information the load capacitance \(C_{Li}\) at every gate output \(i\) is calculated. The power can then be estimated by:

\[
Power = 0.5V_{dd}^2 \sum_{\text{All gates}} T_i \times C_{Li}
\]

This method does not account for the leakage power. This can be estimated by simulating the circuit without any vector transitions for a long period of time. The current drawn is the static power consumption of the chip. This static power consumption can be added to the final estimate to achieve a more accurate power estimate. We have used this method for estimating power in large circuits and achieved estimates within 1\% error of the estimate using SPICE.
3.4.2 Probabilistic Techniques

These techniques focus on modeling the transitions occurring at a gate as a probability function. It is possible statically to predict the probability of the transitions occurring at the inputs of the circuit based on the given vector set. Given these, the transition probability at each gate for every PI transition, can be calculated by analyzing the topology of the circuit. The transition density can be estimated by transistion probability × vector transitions at PIs per second. The final power estimate can be gotten from summing up the individual power estimates of each gate using the transition density. The main issues for probabilistic techniques are determining to what extent these signals are independent and also the spatial and temporal correlation of signals. For a very detailed analysis of the issues the reader is referred to the literature [127, 160].

In one of the earliest attempts at probabilistic estimation of power, Cirit [41] modeled the gate as a combination of a resistor and a capacitor and estimated the probability of a current flowing to charge or discharge the capacitor, i.e., of a transition occurring at the gate. A zero delay model assumes that all gates in the circuit contain zero delay. This model has been used [41] to estimate the transition probability of the gates by a single run through the circuit. The simplest way to propagate transition probabilities is to work with the gate-level description of the circuit. If

\[ y = \text{AND}(x_1, x_2) \]  

then

\[ P_s(y) = P_s(x_1)P_s(x_2) \]  

provided that \( x_1 \) and \( x_2 \) are independent. Similar expressions can be derived for other types of gates. The disadvantage of this method is that temporal as well as spatial independence of the input vector set is assumed.

Najm et al. [135] describe a technique called CREST whose input is in the form of a probability distribution at the PIs, which is a timing wave representing the
estimated rise and fall times of signals at a gate. This probability distribution is then propagated through the circuit to get the probability distributions at every gate output. This method is more accurate than the previous one as the probabilities at the PIs are not lumped into a single variable but are allowed to vary and the actual averaging is done at the gates. The disadvantage, however, is that it cannot predict the waveforms accurately.

Najm described another program, DENSIM [133], that propagates the transition density values from the PIs to the circuit. He used the concept of the Boolean difference to propagate the transition from level to level. As an example, consider \( y = \text{AND}(x_1, x_2) \). In this case:

\[
\frac{\partial y}{\partial x_1} = x_2 \tag{3.4}
\]

\[
\frac{\partial y}{\partial x_2} = x_1 \tag{3.5}
\]

Hence, the transition density \( D(y) \) of output \( y \) is given by:

\[
D(y) = \frac{\partial P_s(y)}{\partial t} = P_s(x_2) \frac{\partial P_s(x_1)}{\partial t} + P_s(x_1) \frac{\partial P_s(x_2)}{\partial t}
\]

\[
= P_s(x_2)D(x_1) + P_s(x_1)D(x_2)
\]

where

- \( D(x) \): transition density of signal \( x \)
- \( P_s(x) \): transition probability of signal \( x \).

The calculation of Boolean differences gets complicated as we advance toward the primary outputs (POs) of the circuit but the primary advantage is that the transition densities are computed more accurately by a single traversal of the circuit.

### 3.4.3 Statistical Techniques

The main disadvantage of the probabilistic techniques is that we need to know the transition probability (or density as in DENSIM) of the PIs with reasonable
accuracy. Since this is not very easy to predict quite often we revert to statistical techniques for power estimation. The idea behind this technique is very simple and appealing: simulate the circuit repeatedly with different input vector sets. The simulation is done using some logic simulator while monitoring the power. After every simulation, the transition density at each gate is adjusted to reduce the estimation error and eventually the power will converge to the average power. The issues are how to select the input patterns and when can we be assured that the power has converged. For this, various statistical mean estimation techniques are employed, thus making this a Monte Carlo method.

Burch et al. [25, 188] describe a program called McPOWER, which is the first known Monte Carlo approach for power estimation. McPOWER is a variable delay simulator, which measures the power for various random sets of input patterns and then computes the average. The stopping criterion is when the mean is stable for a very large set of input patterns. This assumption is valid because the independence of the inputs is assured as they are selected randomly. McPOWER is twice as slow as DENSIM but is 500% more accurate. The major advantage of this approach was its ease of handling feedback circuits, which could turn out to be a nightmare for probabilistic techniques.

Xakellis and Najm [203] have improved upon McPOWER by devising a mean estimator of density (MED), which estimates the transition densities of individual nodes statistically. The error tolerance level can be specified by the user up-front and the nodes with least tolerance can also be identified. This speeds up convergence and estimates of power are done with desired confidence levels. The reason why one might want individual transition densities is to diagnose a high-power problem and, if possible, find a solution for it. The results show 99% accurate estimates for all benchmark circuits but the time taken is in hours. The accuracy can be traded off for simulation time in this method.

The major source of error in the statistical simulation technique is the neglect of the sequential and the higher-order temporal correlation. Several suggestions have been made to reduce the error by mixing application pattern simulation with
statistical stopping criteria [136]. In addition, one can speed up the simulation by separating the problem of current analysis from the problem of activity analysis. A detailed description of these is beyond the scope of this thesis.

3.5 Summary

To effectively design circuits with low power, we need accurate estimation techniques at every design level. This chapter introduced some of the prominent power estimation techniques. The higher-level estimation is done mostly based on statistics and prior design knowledge. As the circuit proceeds through lower design levels, more details become available and the power estimates are done by detailed simulation or by probabilistic techniques. In all, the estimation accuracy depends on the design level and lower the design level, more accurate is the power estimate.
Chapter 4
Prior Work: Delay Elements in CMOS Circuits

4.1 Introduction

Delay elements are components inserted into a digital circuit that do not alter the signal value, but deliver the same waveform at the output with some extra delay. In certain applications, such as clock skew optimization, the timing of signals needs to be manipulated such that the arrival times of certain signals are delayed by given amounts. In these cases, buffers are added to delay the waveform and meet the timing requirements. In this chapter, we describe some of the popular delay elements with an insight into their advantages and drawbacks. We classify the delay elements into three categories. The first class is based on buffers, the second on transmission gates, and the third on voltage controlled transistors. Buffers have been used for glitch suppression but certain other types delay elements are not conducive for the conventional glitch suppression as we will be describe.

4.2 Buffers

A buffer is the simplest of the delay elements. It is a pair (in general, any even number) of cascaded inverters that do not alter the nature of the signal. The schematic of a buffer is shown in Figure 4.1(a). The delay added by the buffer is the time taken for the current to charge (or discharge) the load capacitance. An exact computation of the delay is non-trivial because of the non-linear relation of the current on the output voltage. Let us analyze the propagation delay of a pair of inverters shown in Figure 4.1(a). After a falling transition at the input, transistors M1 and M4 will be ON. The delay for a falling transition is thus given
Figure 4.1: Buffer delay elements. (a) Simple cascaded inverter pair buffer delay element, (b) \( m \)-cascaded inverter delay element and (c) Staged cascaded inverters.

by:

\[
d_{\text{fall}} = (R_1 \times (C_3 + C_4)) + (R_4 \times C_L)
\]

where \( R_i \) is the ON resistance of transistor \( i \) and \( C_i \) is the input gate capacitance of transistor \( i \). \( C_L \) is the load capacitance at the output of the buffer. Similarly, for a rising transition, transistors M2 and M3 are ON and the delay is given by:

\[
d_{\text{fall}} = (R_2 \times (C_3 + C_4)) + (R_3 \times C_L)
\]

We know that the ON resistance of a transistor is proportional to \( \frac{L}{W} \) and the gate capacitance is proportional to \( (W \times L) \), where \( L \) and \( W \) are the length and width of the transistor respectively. Hence, the delay of the buffer can be written as:

\[
d_{\text{rise}} = f(L_1, W_3, W_4) + g(L_4, C_L)
\]
\[ d_{\text{fault}} = f(L_2, W_3, W_4) + g(L_3, C_L) \]

where \( f \) and \( g \) are non-linear functions \cite{152}. Thus the delay of the buffer can be manipulated by changing the sizes of the transistors for a fixed load capacitance.

Buffers act as simple delay elements and they can be made to work for any load capacitance because of their signal regenerative property. They do not alter the slope of the signal. The disadvantage, however, is that they consume dynamic power. From the example above, for every input transition, the buffer makes two transitions. Every transition is a charging or a discharging operation on a capacitor. Every rising transition is caused by dissipating \( 0.5CV_{\text{DD}}^2 \) in the resistance and storing \( 0.5CV_{\text{DD}}^2 \) in the capacitor. The falling transistor dissipates the stored charge of \( 0.5CV_{\text{DD}}^2 \) through the resistance. Hence, for a pair of transitions at the input of the buffer, the energy consumed is:

\[
\text{Energy consumed} = (C_1 + C_2 + C_3 + C_4) \times V_{\text{DD}}^2 + C_L \times V_{\text{DD}}^2
\]

where \( C_i \) is the gate capacitance of transistor \( i \). When the buffer is used as a delay element, to achieve large delays we increase the capacitance \( C_i \). This also increases the dynamic power consumption in the buffer. This is one of the major disadvantages of using a buffer as a delay element. The area overhead due to the added transistors is also a disadvantage.

### 4.2.1 \( m \) Transistor Buffer

This is a variant of the normal buffer and the transistor schematic is as shown in Figure 4.1(b). The \( n \) tree and the \( p \) tree contain \( m \) transistors in series \cite{113}. This increases the resistance of the series charging path and hence the delay is increased. Assuming that the transistors are sized identically, the delay is given by:

\[
delay = (m \times R_1 \times 2m \times C_1) + (mR_2 \times C_L)
\]
where $R_1$ is the ON resistance of one transistor and $C_1$ is the gate capacitance of one transistor. This element still has the disadvantage of increasing the dynamic power of the circuit. Another added disadvantage is the area overhead. Inserting many transistors can be a significant wastage in area critical circuits.

### 4.2.2 Staged Buffers

This delay element consists of an arrangement of three inverters, in two stages as shown in Figure 4.1(c). The first stage consists of two inverters, where the output of each inverter controls the input of one transistor in the second stage [113]. The intuition is that the two large transistors in the output stage are never on at the same time, thus eliminating short circuit dissipation. The delay obtained is set by altering the sizes of the two inverters in the first stage. The transition that controls the output edge is always produced by the transistor in series with the charging resistance and it can be set by using lengthy transistors. The penalty is that the slope of the output edges increases. Also, when both of the transistors in the output stage are simultaneously OFF during transition, the output is susceptible to cross-talk. It also needs more transistors than the other delay elements.

### 4.3 Transmission Gates

A CMOS transmission gate is a bi-directional switch consisting of a parallel connection of an $n$MOS and a $p$MOS transistor, which are controlled by complementary signals as shown in Figure 4.2(a). The $n$MOS and $p$MOS transistors pass logic 0 and 1, respectively without affecting the voltage level of the signal. The effective resistance $R_{eq}$ of the CMOS transmission gate is given by the combined resistances of both transistors connected in parallel. The delay of the transmission gate for charging a load capacitance $C_L$ is given by:

$$d_{trans} = R_{eq} \times C_L$$
Figure 4.2: Transmission gate delay elements. (a) Simple CMOS pass transistor as delay element and (b) Transmission gate cascaded with a Schmitt trigger circuit for a steeper output slope.

The effective resistance $R_{eq}$ can be increased by altering the $(\frac{L}{W})$ of the transistors. The delay of the transmission is effectively controlled by simple sizing of the two transistors [113]. The advantage of the transmission gate is that it is simple and has only two transistors, hence less area. But the problem is the signal integrity. The slope of the output signal deteriorates with higher resistance. The voltage $V_{out}$ through the gate is given by:

$$V_{out} = V_{DD}(1 - e^{-\frac{\Delta t}{R_{eq}C_L}})$$

The slope of the waveform is defined as the transition time for the signal to change from 10% to 90% of its final value (90% to 10% when falling). The slope of the transmission gate output is:

$$\Delta r/f = V_{out}(10\%) - V_{out}(90\%)$$

$$= \ln(9) R_{eq}C_L$$

Thus, increasing the delay along the transmission gate also increases the slope of the waveform. This makes the gates in the fanout stay in transition for a longer time, thus making them more susceptible to noise jitters in the circuit. Noise at the gate input when a gate is in transition may cause the gate to enter a metastable or erroneous state. Thus, a transmission gate cannot be used for
obtaining large delays. The increased slope also results in higher short circuit power at the output, as the transistors in the output are simultaneously ON for a longer time. These two factors limit the delay that can be achieved by inserting a transmission gate.

### 4.3.1 Transmission Gate Cascaded with Schmitt Trigger

The *Schmitt trigger* is a circuit that generates a fast clean output from a noisy slow varying waveform. Its schematic is as shown in Figure 4.2(b). If the output of the Schmitt trigger is initially low, then the input to the second stage is driven to high continuously. If there is a rising transition at the input, then the second stage is driven faster to the output value of a falling transition because of the feedback loop. Thus, the Schmitt trigger can speed up the slope of a slow waveform. Mahapatra *et al.* propose a delay element, using a transmission gate followed by a Schmitt trigger [113]. The transmission gate is responsible for the delay and the Schmitt trigger is for slope reduction. The entire ensemble takes 8 transistors, which is a high price for slope reduction. The extra power consumed by the inverter and the Schmitt trigger is also a disadvantage.

### 4.4 Voltage Controlled Elements

In all of the designs mentioned above, the elements are designed for a static delay and cannot be changed during execution. It is advantageous in certain cases to control the delay of the elements during runtime. This can be achieved by using *voltage controlled delay elements* [113]. We will discuss three types of voltage controlled delay elements.

#### 4.4.1 \( n \)-Voltage Controlled Elements

An *\( n \)-Voltage controlled delay element* is as shown in Figure 4.3(a). The idea here is to have a pair of inverters, connected to \( n \)MOS transistors as current
Figure 4.3: Voltage controlled delay elements. (a) $n$-Voltage controlled delay element, (b) $p$-Voltage controlled delay element and (c) $np$-Voltage controlled delay element.

sources. The gates of these $n$MOS transistors are connected to a control voltage $V_n$. The delay of this element can be controlled by changing the control voltage $V_n$ between the threshold voltage $V_t$ and the supply voltage $V_{dd}$. Since either of the two inverters will be using the pull-down network, the delay can be varied by controlling just the $n$-tree. The advantage of this technique is the dynamic variability of the delay by a single control voltage. The drawback, however, is the increased slope of the output waveform. This increases the short circuit dissipation of the following stage as described above.
4.4.2 \textit{p-Voltage Controlled Elements}

A \textit{p-Voltage controlled delay element} is shown in Figure 4.3(b). The functioning is similar to the \textit{n-type controlled elements} but the pull-up network is delayed in this case. This also has the same drawback of the increased slope of the waveform.

4.4.3 \textit{np-Voltage Controlled Elements}

An \textit{np-Voltage controlled delay element} is as shown in Figure 4.3(c). This is a combination of the \textit{n} and \textit{p} voltage controlled delay elements described above. The advantage is the increased control over delay. The drawback, however, is the higher slope of the waveform than the other two voltage controlled elements described above. Other disadvantages include the extra voltages required for each delay element in the circuit if different delays are desired and also the extra power consumption due to these elements.

4.5 \textbf{Summary}

In some cases during design, delay needs to be inserted for better performance or for synchronization [152]. This is achieved by the introduction of delay elements. We have described the delay elements based on buffers. These elements are very simple and reliable, but their problem is increased dynamic power. Transmission gates can be used for delay injection, but the range is limited. They also increase the slope of the waveform, which increases the short circuit power of the following stage. Voltage controlled elements can provide delay that can be controlled during operation by changing the voltage. These elements can be useful, but the problem is extra control hardware and also the increase in slope is also a problem. In general, the transmission gates are used for smaller delays and buffers for larger delays.
Chapter 5

Background on Linear Programming Techniques

In this chapter we present a detailed overview of two prior techniques that are similar to our approach. Both techniques, as well as the one in the present work, seek an exact solution of the problem. We have included the previous techniques to highlight their disadvantages in designing circuits of large size and to further emphasize the superiority of the new method, which will be described in the next chapter.

5.1 Definitions

Since the concepts of delay and hazard have been explained in previous chapters we refresh only some of the major definitions for the benefit of the reader. The following definitions are helpful for understanding what follows:

- *Inertial delay* is the time taken by the gate output signal transition to occur after the input change causing the transition has occurred.

- *Differential delay* is the difference in signal arrival times at the inputs of a gate with more than one fanin.

- *Path balancing* is the technique of equalizing the delays of paths so that the differential delay at the gate inputs is 0. This prevents the hazard transitions from being generated at the gate outputs.

- *Hazard filtering* [1] is the technique of increasing the inertial delay to be greater than the maximum differential delay at the gate inputs. This suppresses the generation of hazards at the gate outputs.
• *Gate sizing* is defined as assigning load drive capabilities to the gates of a Boolean network, such that a given delay limit is obeyed, and the total power consumption is minimal [19]. Similar to the hazard filtering technique, this method also uses the filtering effect of the gates but achieves the increase in inertial delay by resizing the gates in the circuit.

5.2 Overview of Linear Programming (LP) Approaches

In this section we present two LP approaches for power optimization, which inspired this work. Both techniques are essentially the same, even though they are both applied at different levels of design. The method of Agrawal *et al.* is applied at the gate level whereas that of Berkelaar *et al.* is applied at the transistor level.

5.2.1 Berkelaar *et al.*’s Method

To understand the LP in detail we need to look at the mathematical model of the delay of a gate expressed as [20]:

\[
   t_p = t_{\text{int}} + c \frac{C_{\text{wire}} + \sum S_i C_{\text{in},i}}{S_{\text{cell}}} \tag{5.1}
\]

where

• *t*<sub>p</sub>: gate propagation delay

• *t*<sub>int</sub>: gate internal delay

• *C*<sub>wire</sub>: wire capacitance driven by the cell

• *C*<sub>in,i</sub>: input capacitance of the gates in the fanout cone

• *S*<sub>i</sub>: speed constant of cell *i* in the fanout cone

• *S*<sub>cell</sub>: speed constant of the cell

• *c*: a constant
When a gate is sized, both the internal capacitance and the resistance of the cell increase. The gate delay due to internal capacitances remains constant during sizing. The input capacitance of the gate is affected by the resizing and hence sizing of a gate in the fanout cone of some gate \( i \) affects the delay of gate \( i \). This is modeled by including the speed factors of all of the gates in the fanout in the delay equation. The wiring capacitance, however, is unaffected during sizing. The total load capacitance, consisting of wiring and fanout, is divided by the cell speed factor. This speed factor is the factor by which the drive capability of the gate is scaled.

A glitch (also called a hazard pulse) occurs when the differential delay at the PIs is greater than the propagation delay \( t_p \) of the gate. A linear program has been formulated as follows.

5.2.1.1 Objective Function

They minimize the overall power but power is directly proportional to the gate load capacitance. Hence, they minimize the overall load capacitance from the expression:

\[
\min : \Sigma_{i\in\text{cell gate}} C_i S_i
\]  

(5.2)

5.2.1.2 Constraints

The constraints for solving the LP are:

\[
T_i - T_o < t_p
\]  

(5.3)

\[
t_p = t_{int} + c \frac{C_{wire} + \sum S_i C_{in,i}}{S_{cell}}
\]  

(5.4)

\[
1 \leq S_i \leq \text{limit}
\]  

(5.5)
where

- $T_i$: total delay at the input $i$ of gate from PIs
- $T_o$: total delay at the output $i$ of gate from PIs
- $t_p$: gate propagation delay
- $t_{int}$: gate internal delay
- $limit$: maximum speed constant allowed for a gate

We see that the constraints for the formulation are almost linear except for the expression for $t_p$. The non-linearity of this constraint poses problems for solving the model. For specific instances the formulation could be infeasible when the differential delays are too large. The number of constraints also increases exponentially with the number of gates in the input circuit as $T_i$ has as many values as the paths in the circuit. Hence, this model is impractical for the optimization of larger circuits and also since it demands more complex non-linear solvers, it increases the optimization time, too.

5.2.2 Agrawal et al.'s Method

Agrawal et al. [1, 2] described a minimum transient energy (MTE) design of digital circuits. A circuit is said to have MTE design if for any set of arbitrary simultaneous changes at PIs, steady state is reached with no more than one signal transition at each gate output. Consider the gate in Figure 5.1. Let $P1$ be the set of different paths leading from input 1 of the gate to the PIs and let $P2$ be
Figure 5.2: A 1-bit adder.

the set of paths from input 2 to the PIs. Now the MTE condition can be ensured only if the delay difference in any pair of P1-P2 paths is not greater than the inertial delay \( d \) of the gate. Since the LP constraints have gate delays as the parameters, we need to write a constraint for every pair of paths from sets \( P1 \) and \( P2 \) for every gate as:

\[
|\sum \text{delay of gates on a path in } P1 - \sum \text{delay of gates on a path in } P2| < d
\]

The paper [2] describes a combination of delay balancing and hazard filtering for ensuring the circuit to be a MTE design. MTE design can always be guaranteed, even when the circuit is constrained with a maximum overall delay, provided that delay buffers are inserted. We will illustrate the LP formulation with a 1-bit adder shown in Figure 5.2. The circuit has 9 gates, which potentially consume energy.

5.2.2.1 Variables

Every gate \( i \) has a delay variable \( x_i \). In addition, delay buffers with delays \( x_j \) are assumed at all PIs and fanouts. Hence, for the adder shown in Figure 5.2 we have:
• Gate delays: $x_i$ for $4 \leq i \leq 12$

• Buffer and PI delays: $x_j$ for $15 \leq j \leq 29$

Buffers are shown as black triangular elements in Figure 5.2. PI delays are represented by buffers 15, 16 and 17. The PO signals 13 and 14 do not require any buffers since they do not fan out. Their delays are represented by the gates producing them, namely, gates 11 and 12.

5.2.2.2 Objective Function

The main idea is to minimize the power dissipation of the circuit but this can be ensured by the conditions that follow. The insertion of buffers does ensure the elimination of glitches in the circuit but will increase the power consumption a bit, due to the extra gate in the circuit. Hence, it is their aim to eliminate glitches with the introduction of the least number of buffers.

\[
\text{Ideal objective: } \min_{15 \leq j \leq 29 \text{ and } x_j \neq 0} \frac{x_j}{x_j} \tag{5.6}
\]

Because the ideal objective function is non-linear, an actual objective function to minimize the overall buffer delay is used [2]:

\[
\text{Actual objective: } \min \sum_{15 \leq j \leq 29} x_j \tag{5.7}
\]

5.2.2.3 Constraints

There are three types of constraints described.

• Initial constraints

The lower bound is set for every parameter of the gate using this set of constraints. They write constraints for every gate as $x_i \geq 1$ and for every buffer as $x_j \geq 0$.

• Path constraints
For every path leading to a gate they write constraints for the differential delay at the inputs and make sure that it is no greater than the inertial delay of the gate. For example, for gate 5:

\[ x_{15} + x_{20} - (x_{16} + x_{21}) < x_5 \]
\[ x_{16} + x_{21} - (x_{15} + x_{20}) < x_5 \]

The difference in the sum of delays along the path from gate 5 to PI 1 and the sum of delays in the path from gate 5 to PI 2 must be less than the inertial delay of gate 5. Each pair of paths provides two constraints and thus every gate has as many constraints as there are path pairs leading from its inputs to the PIs.

- Maximum delay constraints

This is to ensure that the solver does not insert buffers in the critical path of the circuit. The speed of a circuit is determined by the delay along the critical path and increasing it would increase the overall delay of the circuit. We set the maximum delay (\( \text{maxdelay} \)) of a path as a constraint:

\[ x_{15} + x_{18} + x_4 + x_{22} + x_6 + x_7 + x_{23} + x_8 + x_{29} + x_{11} \leq \text{maxdelay} \quad (5.8) \]

Similarly, every path between PIs and POs adds an inequality of the above type to the constraint set.

The LP model is then solved using AMPL [58] and the results have also been presented by the authors for small circuits. But the constraints for every gate are proportional to the number of paths that traverse it and, hence, are exponential with respect to the size of the circuit. The number of constraints solvable by the solver is limited and hence large circuits cannot be solved using this model. This inspired us to build a model that makes solving large circuits feasible but still has the same efficiency and accuracy of this model.
5.3 Raja et al.'s Method

The method described above eliminates glitches effectively, but the number of constraints in the linear program increases exponentially with the circuit size [156, 157]. Hence, this method cannot be used for optimizing large circuits. Raja et al. proposed a linear constraint set solution for LP optimization. We describe this method in greater detail as the new LP proposed in this thesis derives the linearity from this method. We also show that the results are essentially the same as those of the previous methods.

5.3.1 The Concept of a Timing Window Per Gate

Instead of having a single parameter for a gate as was done earlier [1, 2], we introduce two new variables for every gate, one for the earliest time and the other for the most delayed time of arrival of a signal at the output of a gate. The difference of these variables is a timing window within which the various signals arrive at the gate. Consider a gate $i$ with $n$ inputs. We define a variable $T_i$ as the maximum time instant at which an event can occur at the output of the gate after the occurrence of an event at the PIs of the circuit. Similarly, $t_i$ is the minimum time instant at which an event can occur at the output of the gate. This means that events always occur in the interval $[t_i, T_i]$ at the output of the gate $i$.

**Theorem 1:** Consider a gate $i$ with $n$ inputs, receiving events arriving from fanin gates 1, 2, ... $n$ at times $m_1$, $m_2$, ... $m_n$. Assuming that $T_1 \leq T_2 \leq ... \leq T_n$ and $t_1 \leq t_2 \leq ... \leq t_n$ are the arrival time parameters for the fanin gates, the number of events at the output of gate $i$ cannot exceed:

$$\min\{n, 1 + \left\lfloor \frac{T_n - t_1}{d_i} \right\rfloor \} \quad (5.9)$$

where $d_i$ is the inertial delay of gate $i$. 
Figure 5.3: An arbitrary distribution of events at the inputs of a gate.

**Proof:** We consider two cases for the gate $i$ with $n$ inputs.

- *First upper bound.* The maximum number of events cannot be greater than the maximum number of possible events at the input, which is $n$ in this case (the number of fanins).

- *Second upper bound.* At the inputs of gate $i$, $n$ events can be arbitrarily placed in time. Without loss of generality, we order them as $1$, $2$, \ldots $n$ in Figure 5.3. From the definition of *ideal delay* \[2\] an output event can occur only if the separation between successive events is greater than $d_i$. The largest window in which the events can occur is $[t_1 + d_i, T_n + d_i]$, and the number of events is given by:

$$1 + \left\lfloor \frac{T_n - t_1}{d_i} \right\rfloor$$

(5.10)

where the $1$ is added for the first event at the start of the interval of length $T_n - t_1$.

Combining both upper bounds, we get Equation 5.9.

From Theorem 1, the number of events takes the *least possible value* (the condition for minimum dynamic power) when

$$T_n - t_1 < d_i$$
According to Equation 5.9, the number of events at the output of the gate will not exceed 1. Also, since

\[ T_i = T_n + d_i \]
\[ t_i = t_1 + d_i \]

then the condition for MTE can be written as:

\[ d_i > T_i - t_i \] \hspace{1cm} (5.11)

### 5.3.2 Linear Program

A linear program determines a set of variables such that an objective is minimized under the given constraints. We illustrate the linear programming model with the example of the adder circuit shown in Figure 5.2. Buffers are inserted at the PIs and at each fanout branch of a signal (PI or gate) with more fanouts than 1. The linear program is developed as follows.

#### 5.3.2.1 Variables

The variables can be split into two categories:

1. Gate variables
2. Buffer variables

The gate variables are a set of three variables for each gate in the circuit. They are:

- \( T_i \) for every gate \( i \): This is the maximum time at which the output of this gate can produce an event after the occurrence of an event at the PIs.

- \( t_i \) for every gate \( i \): This is the minimum time at which the output of this gate can produce an event after the occurrence of an event at the PIs.

- \( d_i \) for every gate \( i \): This is the inertial delay of the gate which has to be given as an output of the optimizer.
The buffer variables also have the same set of parameters as gate variables but are treated differently later in the program.

5.3.3 Objective Function

The injection of buffers into the circuit increases the area of the circuit and the obvious objective would be to reduce the number of them. Hence, our objective function would be to reduce the delay of the buffers. The real objective would be to achieve a balanced delay circuit with the minimum number of buffers possible and the sum of the buffer delays would be equally effective in achieving this objective.

5.3.3.1 Initial Constraints

The lower bound is set for every parameter of the gate using these set of constraints. We write the constraints $d_i \geq 1$ for every gate $i$, $d_i \geq 0$ for every buffer $i$, $T_i \geq 0$ for every gate and buffer $i$, and $t_i \geq 0$ for every gate and buffer $i$.

5.3.3.2 Gate Constraints

First let us deal with the constraints for a gate with a single fanout. This set of constraints includes the buffers, too. Consider the buffer 19 in Figure 5.2. Its fanin is buffer 16. Hence its set of constraints would be:

$$T_{16} + d_{19} = T_{19};$$

$$t_{16} + d_{19} = t_{19};$$

These constraints are self-explanatory as the maximum and minimum delay at the input of the gate would just be added to the delay of the gate (or buffer) as it proceeds to the fanout. Now consider the case where there are more than 1 fanins, as in gate 7. Then we have:

$$T_7 \geq T_5 + d_7;$$

$$T_7 \geq T_6 + d_7;$$
\[ t_7 \leq t_5 + d_i; \]
\[ t_7 \leq t_6 + d_i; \]
\[ d_i > T - t_7; \]

The first four constraints ensure that the parameter \( T; \) settles at the value that is the maximum \( (T_5, T_6) \) and \( t_7 \) would settle at the minimum \( (t_5, t_6) \). The condition for MTE is ensured by the last constraint.

### 5.3.3.3 Overall Circuit Delay Constraints

To ensure that the delay balancing does not slow down the circuit beyond the specified limit we use a given upper bound on the maximum delay at the output. This can be ensured by placing that upper bound on parameter \( T \) of all gates feeding the primary outputs of the circuit. Thus we have additional constraints as:

\[ T_{11} \leq \text{maxdelay} \quad \text{and} \quad T_{12} \leq \text{maxdelay} \]

where \( \text{maxdelay} \) is specified according to the application of the device and the amount of speed the user is willing to sacrifice for power savings. It is a user-defined parameter.

### 5.3.4 Validation of the Model

In this section, we validate the new LP model by showing its equivalence to the exact path constraint model [2]. Consider the block of circuitry shown in Figure 5.4. This gate consists of three gates and delay buffers (shown as black triangles) on the three primary inputs and two fanout lines.

First, let us generate the constraints using the old model described by Agrawal \textit{et al.} [2]. The variables here are the gate and buffer delays \( d_1, \ldots, d_8 \). Path constraints are required only for multi-input gates.

- For gate 6:

\[ d_1 + d_3 - d_2 < d_6 \quad (5.12) \]
Figure 5.4: A combinational circuit block.

\[ d_2 - d_1 - d_3 < d_6 \]  \hspace{1cm} (5.13)

The two inequalities are needed for each pair of paths to ensure that the absolute value of the differential delay does not exceed the gate delay.

• For gate 7:

\[ d_1 + d_4 - d_5 < d_7 \]  \hspace{1cm} (5.14)

\[ d_5 - d_1 - d_4 < d_7 \]  \hspace{1cm} (5.15)

• For gate 8: Since there are two possible paths to every input of gate 8 we have four pairs of paths, giving rise to eight path inequalities. These can be easily constructed and, therefore, we have avoided listing them.

All of these constraints are based on paths between the gate and PIs. Similarly, the overall circuit delay constraints are based on all paths through the circuit. The constraints defined in Section 5.3.2, on the other hand, do not depend on paths. They are derived using three gate variables, \( T_i \), \( t_i \) and \( d_i \). Using the example of gate 6 in Figure 5.4 we will show that the above path constraints are identical to and can be derived from the constraints of Section 5.3.2 and, hence, this LP will produce the same solution as one described by Agrawal et al. [2].
For gate 6 in Figure 5.4, we write:

\[ t_6 \leq t_2 + d_6 \quad (5.16) \]
\[ t_6 \leq t_3 + d_6 \quad (5.17) \]
\[ T_6 \geq T_2 + d_6 \quad (5.18) \]
\[ T_6 \geq T_3 + d_6 \quad (5.19) \]

Also, the minimum transient energy (MTE) condition of Inequality 5.11 requires

\[ T_6 - t_6 < d_6 \quad (5.20) \]

For buffers 1, 2 and 3, we have:

\[ t_1 = T_1 = d_1, \ t_2 = T_2 = d_2, \ \text{and} \ t_3 = T_3 = d_1 + d_3 \quad (5.21) \]

Substituting in Inequalities 5.16 through 5.19 from 5.21, we obtain:

\[ t_6 - d_2 < d_6 \quad (5.22) \]
\[ t_6 - d_1 - d_3 < d_6 \quad (5.23) \]
\[ d_6 < T_6 - d_2 \quad (5.24) \]
\[ d_6 < T_6 - d_1 - d_3 \quad (5.25) \]

Adding Inequalities 5.22 and 5.25, we obtain

\[ d_1 + d_3 - d_2 < T_6 - t_6 \quad (5.26) \]

Adding Inequalities 5.23 and 5.24, we obtain

\[ d_2 - d_1 - d_3 < T_6 - t_6 \quad (5.27) \]

Finally, if we substitute Inequality 5.20 in 5.26 and 5.27 we obtain the path constraints of Inequalities 5.12 and 5.13. A similar procedure can be applied recursively to gates farther away from PIs to derive the path constraints from the local constraints of the new LP constraint set of Section 5.3.2. This applies to the output gate constraints specifying the overall circuit delay. Thus the equivalence of the this LP formulation and the previous method [2] is established.
5.3.5 Why is this Model Superior?

The strength of this model is in reducing the unnecessary constraints at each gate. The inertial delay constraint for hazard filtering is not written for all paths to the inputs as done previously but it isolates the maximum and minimum possible time instants \((T \text{ and } t, \text{ respectively})\) at which a signal can arrive at the input. Since we are interested only in the maximum differential delay we have just one constraint specifying that condition after the maximum \(T\) and minimum \(t\) have been isolated.

The superiority of the model can be easily visualized by looking at the example in Figure 5.5. Let us consider the blocks such that each block is the circuit shown in Figure 5.4. As we saw in the example the number of constraints for the first block using the prior method would be a constant multiple of 4 (since it has four paths through it. Note that we ignore the initialization constraints as they will be similar for both methods.) If two blocks are cascaded then the circuit would require at least \(4 \times 4\) constraints as there are as many paths leading to the last gate. Hence, we conclude that a circuit with \(n\) such blocks cascaded would require at least \(4^n\) constraints.

Using Raja et al.'s method, the constraints for a single block would be a multiple of 15 (five constraints per gate and we have three gates here). Now if two blocks are cascaded together the number of constraints would become \(15 \times 2\) as it would just increase the number of gates as far as the model is concerned. Hence, a circuit with \(n\) cascaded blocks would have \(15n\) constraints which is linear.

Figure 5.5: A combinational array circuit using the block of Figure 5.4.
Figure 5.6: Number of constraints for the new LP model and the old (path enumeration) model.

and clearly much less than for the previous model.

To further illustrate the point we give the graph in Figure 5.6 with the number of constraints for the ISCAS '85 benchmark circuits. As seen in the figure the benchmark circuit c880 needed 6.9 million constraints with the old model but only 3,611 constraints with new model. The graph shows a linear increase with the number of gates for the new model. Though not obvious in the figure, the graph for the old model has an exponential rise and the constraint set could not be completed for some of the larger circuits (e.g., c6288 and c7552) due to the memory limits of the computer.

5.4 Summary of Linear Programming Techniques

The LP techniques optimize the circuit for glitches by constraining the differential delay at the inputs of a gate, with the gate delay. The optimization is done with
the help of delay buffers and this is a considerable overhead. As an example, the
ISCAS ’85 benchmark circuit containing 3,827 gates needed 366 buffers for glitch
elimination at the same speed. These buffers reduce the power savings achievable
by the method. In this thesis, we propose to eliminate this overhead and achieve
the maximum possible power savings.
Chapter 6
New Variable Input Delay Logic

In the previous chapters, we have seen the various LP techniques for reducing the glitches in digital CMOS circuits. The main disadvantage of these techniques is that they needed buffers to be inserted in the circuit for complete glitch elimination. The extra buffers inserted consume power themselves, thereby reducing the power savings achieved. In this chapter, we look at why buffer insertion is necessary when glitches need to be eliminated, without increase in the overall delay. We propose a variable input delay gate model that can be used for glitch elimination without inserting buffers. The variable input delay gate is a logic device with different delays along different I/O paths through the gate. However, there is an upper bound specified by the technology, which we call $u_b$, that constrains the gate design. Using this new delay model and the parameter $u_b$, we propose a new minimum dynamic power linear program formulation in this chapter. This new formulation specifies the delay assignments for all of the gates in the circuit and their respective input delay variables. We also propose two variations of the main LP, the delay specification LP and the general solution LP, which are modified for different design requirements. The implementation of the variable input delay gate and the design issues involved are explained in the next chapter.

6.1 Why Were Buffers Necessary?

Let us consider the simple combinational circuit shown in Figure 6.1. All the gates have a nominal delay of 1 unit. The critical path delay is defined as the delay of the slowest path through the circuit and it is 3 units in this case. The differential
Figure 6.1: An example circuit with a critical path delay of 3 units. The gate delays and the arrival times of signals at the gate inputs are shown.

Figure 6.2: The optimized circuit of Figure 6.1. The optimization is done by the insertion of a delay buffer, as proposed by Raja et al. [156, 157].

Path delay, as defined in the previous chapters, is the maximum difference of arrival times at the inputs of a gate. The differential path delays of every gate are shown in Figure 6.1. We need to optimize this circuit, such that the glitches are eliminated and the critical path delay is not increased. To prevent a gate from glitching, the hazard filtering condition specifies that:

\[ \text{Inertial delay of a gate} \geq \text{Differential delay at gate inputs} \]

This condition needs to be satisfied by all gates in the circuit. In our example circuit of Figure 6.1, we see that the gate 7 does not satisfy the condition. If only hazard filtering is used for optimization, the delay of the gate 7 can be increased greater than the differential delay of 1 to satisfy the condition. Since the gate is on the critical path, we cannot increase its delay.

Using the LP technique proposed by Raja et al., we can optimize the circuit
Figure 6.3: The optimized circuit of Figure 6.1 using the proposed technique. This example shows that it is possible to eliminate glitches without the help of delay buffers, if we can design the new \textit{variable input delay gate}.

by inserting a buffer at the input 4 [156, 157]. The optimized circuit is shown in Figure 6.2. The differential delays at the gate inputs after the buffer is inserted are shown in Figure 6.2. The differential delay at gate 7 satisfies the hazard filtering condition as the input 4 is delayed by the buffer. Thus, the circuit is glitch free and also works at the critical path delay of 3 units. The disadvantage, however, is the added buffer that also consumes power. The buffer was needed in this design because of the conventional gate design. The gate contains a single delay along all of the input-output paths through the gate.

In our example circuit, the path from gate 4 to gate 7 needs to slowed down without affecting the path from gate 6 to gate 7. Now assume that there is a gate that has different delays along different paths through the gate. If we use such a gate in the optimization of the circuit, we achieve the glitch-free design without the insertion of a buffer. The optimized circuit is shown in Figure 6.3. In this circuit, gate 7 satisfies the condition for every I/O path through the gate. We call this new gate design the \textit{variable input delay gate}. Thus, glitches can be eliminated without buffer insertion using this new gate design. Simplistically, this is the main idea of this work. The logic-level formulation and its issues are discussed in the following sections. The various implementations that we propose for the new gate design are described in the next chapter.
Figure 6.4: Conventional and proposed gate delay models. (a) Delay model of the conventional gate design showing a single inertial delay, (b) Variable input delay gate and (c) Variable input gate delay model showing the input delay variables that are independent of the inertial delay.

6.2 Variable Input Delay Logic

Gate delay is defined as the time taken by a gate output to reach 50% of its value after the input transition causing the change has reached 50% of its value. The conventional gate design contains a single delay value, no matter which input causes the output transition. Hence, there is only one delay for each I/O path through the gate. This delay is known as the output delay or the inertial delay of the gate. The gate delay model of a conventional gate design is shown in Figure 6.4(a). As shown, the delay $d$ of the gate remains the same irrespective of which input causes the transition to occur. The output delay of the gate can be varied by changing the sizes of the transistors in the gate.

As explained in the previous section, it is advantageous to have a gate with different delays along different I/O paths for glitch reduction. We define such a gate as the variable input delay gate as shown in Figure 6.4(b). The delay model of such a gate is shown in Figure 6.4(c). All of the paths through the gate share a common delay $d$ as shown. But there is an extra delay component known as the input delay $d_i$ for every input $i$ through the gate.

Input delay of a gate is defined as the extra delay on a single I/O path through the gate, which can be controlled independently of the other input delays through the gate.

Hence, the overall delay of a path through the gate becomes

$$I/O \text{ path delay} = \text{Output delay} + \text{Input delay}$$
\[ d_{1\rightarrow 3} = d_3 + d_{1,3} \]
\[ d_{2\rightarrow 3} = d_3 + d_{2,3} \]

where \( d_3 \) is the output delay of the gate, \( i = 1 \) or \( 2 \), and \( d_{i,3} \) is the variable input delay of the gate through the I/O path \( i \rightarrow 3 \). Design using these new variable input delay gates is defined as variable input delay logic.

**Variable input delay logic** is defined as the logic level design of circuits using components with variable input and output delays along different I/O paths through a gate.

The components are the variable input delay gates that we described earlier. We propose this new logic family of devices, which are functionally equivalent to the conventional logic gates, but are different in their timing properties. These models when designed effectively can result in glitch-free designs. We discuss their logic level properties and uses in the following sections. The various transistor level implementations of the gate and the limitations are discussed in the next chapter.

### 6.2.1 Technology Parameter \( u_b \)

From the definition of the variable input delay, we see that the delay value of a path has to be varied independently of the delays of the other paths in the gate. The input delay can be varied by either manipulating the capacitance offered by the respective transistor pair, or by inserting a resistance. The input delay, thus, is a function of certain technology-dependent parameters such as the ON resistance of a transistor, per area input capacitance of a MOSFET, etc. These technology parameters, when varied, have second-order effects after a certain range, which makes them dependent on each other. Hence, the independence of the input delays cannot be guaranteed beyond a certain range. Once the delays are dependent, the program formulation is no longer linear. Hence, the method would suffer from
non-linearities like the other methods. To maintain the independence, we need to constrain the variable delay components within an acceptable range. We call this range the *gate differential delay upper bound* $u_b$.

**Gate input differential delay upper bound** $u_b$ is a measure of the maximum difference in delay for any two I/O paths through the gate that can be designed in a particular technology.

At the logic level, the delay assignments are given as an output of the LP. If unconstrained, the program may give gate input delays that differ by large amounts as its solution. However, these delays should be realizable in practice to make the technique useful. This is done in anticipation of the problems that would be faced by the designer at the transistor and layout levels. We quantify this problem in a single value known as $u_b$, which can be defined for every technology. Once $u_b$ is quantified for a technology, we can design glitch-free circuits as described in the following section. The $u_b$ allowed depends upon the specific gate design method chosen, and the fabrication technology the design is targeted for. We describe various implementations in the next chapter.
6.3 A New Minimum Dynamic Power LP

This section introduces the new linear programming formulation that we propose. We use the example of a simple circuit to explain the new LP formulation. Consider the combinational circuit shown in Figure 6.5. Traditionally, for the purpose of gate sizing the circuit is generally viewed with each gate having a single inertial delay and all I/O paths through the gate are assumed to have the same delay. We redefine the gate delay. A gate can be viewed as having one basic inertial delay and a set of input delays for the I/O paths running through the gate. This is illustrated in Figure 6.6.

Each gate can be assumed to have a basic delay variable with input delay elements at the inputs of the gate. We assert that these input delay elements are only for analysis purposes and are not actual extra components in the circuit. The inertial delay and input delays of a CMOS gate are not independent. In the LP we treat them as independent variables, which are bounded by a feasibility constraint so that the delays can be realized in practice. The LP consists of variables that define various parameters in a circuit, constraints on the variables and an objective function that needs to be minimized. This LP produces the minimum dynamic power consuming circuit for the given \( u_b \). We describe the LP.
for the circuit shown in Figure 6.6.

6.3.1 Variables

The variables for the circuit are:

- Basic inertial delay of the gates: \(d_4, d_5, d_6\).

- Gate Input Delays: \(d_{i,j}\), which is the extra delay on the path from the fanin gate \(j\) to gate \(i\). For instance \(d_{4,1}\) is the extra delay of the path through gate 4 while arriving from PI 1. This models the differences in delays of various I/O paths through the gate. Its minimum value is 0.

- \(T_i\) is the latest time of signal change at the output of gate \(i\).

- \(t_i\) is the earliest time of signal change at the output of gate \(i\).

6.3.2 Constraints on Delays

The following constraints set the lower and upper bounds on the variables:

- Lower bounds on gate inertial delays are set to 1. The actual value of this time unit will depend on the specific technology used.

- Lower bounds on gate input delays are set to 0.

- We also set an upper bound \(u_b\) on the gate input delays (see Subsection 6.3.6).

6.3.3 Glitch Suppression Constraints

These constraints ensure that the timing window for signal transitions at every gate output does not exceed the inertial delay [156, 157]. Consider gate 6 in Figure 6.6. The constraints for it are given as:

\[
t_6 \leq t_4 + d_{6,4} + d_6;
\]
\[ t_6 \leq t_5 + d_{6.5} + d_6; \]
\[ t_6 \leq t_8 + d_{6.8} + d_6; \]
\[ T_6 \geq T_4 + d_{6.4} + d_6; \]
\[ T_6 \geq T_5 + d_{6.5} + d_6; \]
\[ T_6 \geq T_8 + d_{6.8} + d_6; \]
\[ d_6 > T_6 - t_6 \]

6.3.4 Maxdelay Constraints

These are the constraints that define the speed of the circuit. The speed is determined by the slowest path in the circuit. The delay of the slowest path is bounded by the latest time of arrival of a signal transition at a PO. Hence, for every PO of the circuit in Figure 6.6 we have:

\[ T_7 \leq \text{maxdelay} \]

6.3.5 Objective Function

The following objective function makes the circuit as fast as possible:

\[ \text{Minimize maxdelay} \]

6.3.6 Feasibility Constraints

The main issue here is the extra upper bound added to the delay of I/O elements. The idea behind this formulation is to design a gate that can have different delays along different I/O paths through it. If unconstrained, the program may give gate input delays that differ by large amounts as its solution. However, every technology has a limit to the amount of flexibility that the designer is allowed. This limit of flexibility shows the feasibility of designing the gate input delays for the technology used at the transistor and layout levels. Hence we call this the
feasibility condition. Now the feasibility constraints for gate 6 would become

\[
\begin{align*}
    d_{6,4} & \leq u_b; \\
    d_{6,5} & \leq u_b; \\
    d_{6,8} & \leq u_b;
\end{align*}
\]

This allows the gate input delay to be varied up to a value of \( u_b \) by the program. This value is a design parameter and is specific to the design technology in which the circuit is being designed. As explained in Section 8.3, \( u_b \) can be determined by the delay analysis of actual gate layouts. Given that feasibility value we can use the minimum dynamic power linear program to design the lowest power consuming yet fastest realizable circuit. The results using this formulation are shown in Chapter 8. The problem with this formulation is that, the maxdelay is not always up to specification. The fastest minimum dynamic power circuit may be too slow for a given purpose. Hence, we propose two variations of the minimum dynamic power LP which are discussed in the next section.

6.4 Delay Specification LP

This is a variant of the minimum dynamic power LP described above. In speed-critical designs, the critical path delay should not exceed a given value maxdelay. If the delay requirement is stringent, then the circuit designed using the minimum dynamic power LP may not meet the specification due to less \( u_b \) of the technology. In such cases, we provide the designer with this new version of the LP in which, the design is optimized using the \( u_b \) of the technology and to increase the speed, buffers are added at select nets. We call this the new delay specification LP.

The LP is similar in many parts to the minimum dynamic power LP, but it is modified in certain aspects. We describe only the modifications to avoid repetition.

- The variables include all the variables in the minimum dynamic power LP.

  In addition to that, buffer variables are inserted at fanout stems and PIs
similar to the buffer insertion technique of Raja et al. [156, 157].

- The \textit{maxdelay} variable becomes a parameter, whose value is the delay specification of the circuit.

- The objective function becomes

\[
\text{Minimize} \sum_{j \in \text{buffers}} d_j
\]

As pointed out in Chapter 5, this is a linear objective function used in place of the exact objective function, which would have been non-linear [2].

The LP obtains the minimum dynamic power condition and the delay specification. This LP inserts lesser buffers than the prior techniques and hence, achieves more power savings than them. The power consumed is not the \textit{minimum dynamic power} of the circuit as there are a finite number of buffers added that increase the power consumption. Results of this delay specification LP are described in Chapter 8.

\section{6.5 General Solution LP}

This is a combination of the \textit{minimum dynamic power} LP and the \textit{delay specification} LP. The basic structure of the LP is very similar to the delay specification LP, but the objective function is altered. This LP is used for designs that need to exploit the entire power-delay tradeoff curve. The objective function is given by:

\[
\text{Minimize} \quad a \times \sum_{j \in \text{buffers}} d_j + b \times \text{maxdelay}
\]

where \(a\) and \(b\) are the weights showing the relative criticality of the design for power and delay. For higher speed applications, \(b\) is increased and for power critical applications \(a\) is increased. Thus, the designer has more flexibility of designing to specification and can make the right choice according to the application. The solution curves for all the LPs described above are shown in the next section.
6.6 Solution Curves

Let us consider the solution space explored by the three proposed LPs and the previous LP technique proposed by Raja et al. Consider the graph shown in Figure 6.7. The linear constraint set technique finds the lowest power consuming circuit for a given maxdelay [156, 157]. The solution curve at the point shown as \( u_b = 0 \) is the circuit with no inserted buffers. To increase the speed of the circuit, buffers are inserted by the technique proposed by Raja et al. The power consumed by the extra buffers is added to the total power consumed by the circuit. According to the technique, more buffers need to be added to increase the speed of the circuit [158]. Thus, the total power increases as maxdelay is decreased and the solution curve is as shown by the Raja et al. solution curve. In the proposed differential delay upper bound technique using the minimum dynamic power LP, if we use conventional gate design, i.e., \( u_b = 0 \), we get the same buffer-less design as shown. But if we increase \( u_b \) the designs get progressively faster, and since no buffers are added, the power is still the minimum dynamic power. As the upper bound \( u_b \) is increased to higher levels, the designs get faster as shown by the different points labeled minimum dynamic power LP in Figure 6.7. Hence, the proposed minimum dynamic power LP produces designs that consume the lowest power. However, every technology has a single \( u_b \) and it is difficult to manipulate it. Hence, once the \( u_b \) is fixed the designs cannot be made faster by this LP.

By using the proposed delay specification LP, we can speed up the circuit for a given delay requirement and a given \( u_b \), by adding extra buffers. Since, the added buffers consume power themselves, the total power of the circuit increases as shown by the solution points marked delay specification LP. But since lesser buffers are added by this LP than the prior technique, the total power is still less than the power of the circuit designed using prior LP techniques. Thus, the proposed delay specification LP produces the fastest possible circuits with lesser buffers inserted than the prior technique. Both the proposed techniques are the end point solutions of what is known as the power delay tradeoff curve. This curve
Figure 6.7: Plot of the power and solution curves of the previous method of Raja et al. [156, 157] and the proposed methods of this thesis. The minimum dynamic power LP solution points show the solutions which are minimum power consuming for a given technology specification $u_b$. The designs given by these points are the fastest minimum power circuits for that $u_b$. The maximum $u_b$ allowed for a technology is given by $\text{Max } u_b$. This is the bound of the feasibility region for that technology. For the delay specified circuits, the minimum power design solution can be speeded up by adding extra buffers. The power is slightly increased as shown by the points labeled proposed delay specification LP method. The proposed general LP solution is shown by the dotted line which is the entire power-delay tradeoff curve. Hence, the methods proposed in this thesis effectively exploit the power-delay solution space.

is a line of solutions that have different delay and power consumption and any of the solutions can be chosen as per the requirement of the design. The proposed general solution LP exploits the entire power-delay tradeoff curve and is shown by the dotted lines in Figure 6.7. Once the $u_b$ of a technology is decided, the designs can be made faster or lower power consuming, depending on the requirement. Thus, in the proposed technique we presented three new LPs that can exploit the solution space much more effectively than the prior LP techniques.
6.7 Original Contributions

We described the new variable input delay logic, which is functionally equivalent to the conventional CMOS circuits, but has different timing properties. The logic contains gates that have different, independently controllable input delays along different I/O paths through the gate. This is the first LP method to use such a formulation. The non-linear dependencies in the gate delays are constrained by a single parameter $u_b$. This keeps the formulation linear and the dependencies are dealt with at the transistor level. Thus, this technique is the only one that can optimize large circuits, keeping the formulation linear. Also, we have presented three new variants of the LP, which exploit the solution space more effectively than the prior techniques. The designer is given more choice in terms of power-delay tradeoff.

6.8 Summary

A given CMOS circuit is optimized for minimum dynamic power when its operation contains no glitches. In the design presented, all glitches are suppressed by adjusting the gate delays. However, to keep the total number of signal transitions at the lowest level, no gates or delay buffers are inserted. This would have forced an increase in the overall circuit delay in the previous designs. In the present method, the overall circuit delay is minimized through a novel design of CMOS gates in which inputs of a gate can have different delays. The amount of differential input delay is restricted by the certain aspects of the CMOS technology. Thus, for a given technology the optimization procedure produces a glitch-free circuit with the minimum possible delay. For designs constrained by a delay specification, a delay specification LP is proposed, where buffers are added to the minimum dynamic power design to meet the delay requirement. The power of this design is also lesser than the prior designs as the number of buffers added are lesser than the prior techniques. A general solution combining both the proposed LPs is also described.
Chapter 7

Transistor Level Design of Variable Input Delay Logic

In the logic-level LP formulation described in the previous chapter, we assumed the design of a variable input delay gate. This gate design has different delays along different I/O paths through the gate. This timing behavior cannot be realized by the conventional gate design. In this chapter, we will look at the various components of delay in a gate. The alterable components are identified in a gate. We then propose four new ways of implementing the new variable input delay gate design. We analyze the advantages and disadvantages of each of the proposed designs.

7.1 Conventional CMOS Logic Gates

There are many ways of combining the transistors to perform the logic functions such as NOT, NAND and NOR. We will describe the static CMOS design style, which is the most prominent in current day technologies.

First, let us examine the inverter (NOT gate) shown in Figure 7.1. It contains a pFET and an nFET connected as shown with input A and output B. The terminal $V_{dd}$ is the power supply and is always at a higher potential (logic 1) and the terminal $G_{nd}$ is always at logic 0. If the terminal A is maintained at logic 1, the pFET is open and the nFET is closed, pulling the output B to 0. If the input at A is a logic 0 then the pFET is closed and the output is pulled up to $V_{dd}$ or logic 1. Thus, this logic gate performs the negation logic function and hence is called the NOT gate or inverter.
Figure 7.1: (a) Schematic of a CMOS inverter. (b) Switch configuration when input applied is 1. (c) Switch configuration when input applied is 0.

Figure 7.2: Schematic of a CMOS NAND gate and the truth table of input combinations.

Similarly, a two-input NAND gate is designed as shown in Figure 7.2. An analysis similar to that of the inverter verifies the truth table of its logic function. For an $n$-input NAND gate, there will be $n$ nFETs in series and $n$ pFETs in parallel. Similarly, an $n$-input NOR gate will have $n$ nFETs in parallel and $n$ pFETs in series.

### 7.1.1 Components of Gate Delay

A MOSFET is not an ideal switch. When *open* it provides a large but finite resistance between its source and drain terminals. When *closed* it provides a small non-zero resistance. For a CMOS gate the output signal change follows the input change with a certain delay. First, the closing and opening of MOSFETs in the gate depends upon the slope of input signals. Then, the output signal
Figure 7.3: Timing diagram illustrating the delay of the NAND gate of Figure 7.2. Input 2 is the cause of the transition at output 3. The interval d is the delay of the gate.

change requires charging or discharging of the output capacitance through a low resistance path provided by the “on” MOSFETs.

Let us analyze the gate delay of the NAND gate schematic shown in Figure 7.2. Assume that a rising signal transition occurs at the input 2 of the NAND gate. The switching characteristics of such a scenario are shown in Figure 7.3.

The rising transition on the input 2 opens the pFET and closes the nFET connected to input 2. Hence the output 3 falls as shown in Figure 7.3. Due to the finite switching time of the transistors, there is a delay between the input transition and the output transition. We call this the gate delay [152, 197].

**Gate Delay** is the time taken for the output signal at the output of the gate to reach 50% of Vdd after the signal at the input of the gate has reached 50% of Vdd.

Gate delay is a function of the amount of resistance and capacitance in the current path. A MOSFET when closed offers a finite resistance $R_{on}$ that is a function of the width and length of the device. Since gate delay is given by $R_{on} \times C_L$ (where $C_L$ is the load capacitance) the gate delay can also be varied by changing the width and length of the transistor [53, 152]. For example, a NAND gate output rises due to current flow in its pFETs. Hence, the delay of the NAND gate for a rising transition can be altered by changing the sizes of pFETs. To increase the
delay, we increase the resistance of the transistor by increasing the length of the transistor. Similarly, the output delay for a falling transition can be varied by changing the length of the nFETs. The delay can also be reduced by increasing the width of the transistor. The delay can be effectively changed by manipulating the width and length of the transistors in the gate. Note that it is possible only to manipulate the overall delay of the gate but not the individual delays along the different paths. For instance, the delay of the gate when input 1 transitions cannot be independently controlled without altering the delay when input 2 transitions. The delays are inter-related.

Consider the path shown in Figure 7.4. The delay of a gate is a function of the on resistance $R_{on}$ and the load capacitance $C_L$. The load capacitance is given by:

$$C_L = C_p + C_r + C_{in} \quad (7.1)$$

where $C_p$ is the parasitic capacitance due to the on transistor, $C_r$ is the routing capacitance of the path and $C_{in}$ is the input capacitance of the transistors in the fanout. The gate capacitance is the main component of the input capacitances. The routing capacitance and the parasitic capacitance are non-controllable and hence, we ignore them in the current discussion. The delay of the path during a signal transition is given by:

$$Delay = R_{on} \times C_L \quad (7.2)$$

The delay can be manipulated by changing the load capacitance or the on resistance by sizing the transistor accordingly. This alters the delay of the gate along
all paths equally. This is called *conventional gate sizing*. Clearly, this cannot be used for designing a variable input delay gate. In this section, we propose three ways of implementing the new variable input delay gate and their design issues.

### 7.2 Gate Design by Input Capacitance Manipulation

The overall delay of the gate is given by Equation 7.2. In the new gate design we need to manipulate the input delay of the gate without affecting the output delay too much. Substituting Equation 7.1 into Equation 7.2, we get:

\[
\text{Delay} = R_{on} \times C_L \tag{7.3}
\]

\[
= R_{on} \times (C_p + C_r + C_{in}) \tag{7.4}
\]

\[
= R_{on} \times (C_p + C_r) + R_{on} \times C_{in} \tag{7.5}
\]

\[
= \text{Output Delay} + \text{Input Delay} \tag{7.6}
\]

From the above analysis we separate the input delay of the gate from the output delay. As seen above, the output delay depends on the parasitic and routing capacitances, which are unalterable. The input delay is a function of the on resistance and the input capacitance of the transistor pair in the fanout. Now the delay of the gate can be changed by increasing the input capacitance offered by the input gate transistor pair. The capacitance offered by a particular input can be increased by altering the sizes of the transistor pair to which the input is connected without altering the sizes of the other transistor pair.

#### 7.2.1 Calculation of \(u_b\)

The delay of the transistor pair can be calculated by using Equation 7.3. The input capacitance of a transistor pair is given by:

\[
C_{in} = W \times L \times C_{ox} \tag{7.7}
\]

where \(W\) is the width of the transistor, \(L\) is the length of the transistor and \(C_{ox}\) is the oxide capacitance per unit area, which is a function of the technology in
which the transistors are designed. The maximum difference in delay that can be achieved is, thus, a function of the amount of capacitance that can be manipulated using the $W$ and $L$ of the transistors. There is a limit to the dimensions of the transistors in any technology that can be used for digital design. The limits are governed by second-order effects, such as \textit{channel length modulation}, \textit{threshold voltage variation}, etc. Although all the gate designs described in this thesis can be used in the \textit{full custom} design flow, we have chosen the standard cell place- and-route design flow for faster design cycles. In the standard cell design flow, the height of the standard cell restricts the maximum length of the transistor. We have chosen the limit of the transistor length for 0.25\,$\mu$m technology as 3\,$\mu$m, which is determined by the standard cell height. The minimum gate length in the same technology is 0.3\,$\mu$m. Hence, the maximum difference in input capacitance is $2.7 \times C_{\text{ox}}$. The maximum differential delay $d_{\text{diff}}$ obtainable in the technology can thus be:

$$\text{Maximum Differential Delay } d_{\text{diff}} = R_{on} \times 2.7 \times C_{\text{ox}}$$

Assuming negligible $C_p$ and $C_r$, the minimum gate delay $d_{\text{min}}$ in the technology is given by:

$$\text{Minimum Gate Delay } d_{\text{min}} = R_{on} \times 0.3 \times C_{\text{ox}}$$

Thus, the gate differential delay upper bound $u_b$ is given by:

$$u_b = \frac{d_{\text{diff}}}{d_{\text{min}}} = \frac{R_{on} \times 2.7 \times C_{\text{ox}}}{R_{on} \times 0.3 \times C_{\text{ox}}} = 9$$

Thus, the $u_b$ of the technology can be calculated by using the bounds on the dimensions of the transistors in the particular technology. There are several design issues in this gate design which will be discussed in the following section.

### 7.2.2 Design Issues

The gate design proposed in the previous section has several drawbacks.
In the gate design it is assumed that the output delay and the input delay are independent. But this is not true for both falling and rising transitions. The input differential delay is realized when the two transistors in the different I/O paths are in parallel. For example, the p-tree of the NAND gate consists of two pMOS transistors in parallel. The input differential delay can be realized when a rising transition occurs at the inputs. If both of the pMOS transistors are sized differently, depending on which input changes, the delay of the gate is different. This is the purpose of the new variable input delay gate design. But the same is not true for the n-tree. The nMOS transistors are in series and both of them have to switch for the gate to transition. Assume that both the nMOS transistors are sized differently and the transition occurs on one of the inputs. Altering the size of one of the transistors affects the series resistance of the output charging path and, thus, the output delay. Hence, the output delay and the input delay are in fact dependent in the case of the rising transition at the input of the NAND gate. Thus, the sizing can be a non-linear problem, which can be difficult to converge.

The parasitic capacitance $C_p$ is assumed to be constant and independent of the transistor sizes. But in reality, $C_p$ is a function of the *sidewall capacitance* of the transistors, which in turn is a function of the transistor sizes. Thus, altering the sizes of one transistor can affect $C_p$ and thus the inertial output delay of the gate.

When the transistors are connected in series to one other, some of them are ON and some are OFF. This leads to different potentials at the source terminals of certain transistors. The static delay calculations are done by assuming that the source is always at 0 potential. But due to the different potentials at the source, the threshold voltages of the transistors are changed drastically and this effect is known as the *body effect*. If the transistors are sized differently in the same series path, then due to the body effect, the
inertial output delay is not constant. It varies with every input pattern to the gate. This is a problem as the LP gives a single delay for every gate output, and the extra delays are not accounted for. Moreover, the minimum dynamic power has to be guaranteed for every input pattern and not only for certain specific cases.

Hence, we see that this design has many disadvantages. Some of them are alleviated in the designs explained below.

7.3 Gate Design with nMOS Pass Transistors

In the previous proposed design by capacitance manipulation, the main problem was the interdependence of the output delay and the input delay. In this second design, we propose to leave the input capacitance unaltered, and increase the resistance of the path. There are certain basic advantages of increasing the resistance of the path, which we will present. Then we will look at the actual gate design and some issues involved with it.

7.3.1 Concept of Increasing Resistance

Consider the charging path shown in Figure 7.4. Energy is drawn from the supply to charge the load capacitance through the resistance. The energy consumed by a signal transition is given by:

\[
\text{Energy} = 0.5C_L V_{dd}^2
\]

where \(C_L\) is the load capacitance and \(V_{dd}\) is the supply voltage. Note that the energy expression does not include resistance \(R_{on}\) in it. The resistance governs the time constant at which the capacitance is charged but the overall energy per transition remains the same. Hence, increasing the resistance of the path does not alter the energy consumed per transition. It does alter the slope at which the charging occurs and, thus, the time of switching. We use this basic observation
in the design of our new variable input delay gate. Increasing the input slope has signal integrity issues which will be discussed in the next section.

### 7.3.2 Effect of Input Slope

The gate delay expression we have used for the above calculations assumes a step input. The input slope is assumed to be infinite. But in reality the input slope of waveforms is finite and has to be accounted for. The delay of a gate can be described as:

\[
Gate \ Delay \ = \ t_{step} + t_{slew}
\]

where \( t_{step} \) is the delay of the gate when the input is a step waveform and \( t_{slew} \) is the delay of the gate due to the input slope or slew. In the new variable input delay gate described, the delay added by the extra resistance is by increasing the slope of the input waveform. The resistance degrades the slew of the arriving signal at the gate input. This increases the overall delay of the gate. But increasing the input slew decreases the robustness of the circuit. The tolerance of a logic gate to a given noise level in the circuit is defined as the robustness of the circuit [152]. The logic gate should function correctly in the presence of noise at the output and also maintain the static timing behavior for which it is designed. But a large input slope means that the circuit is in transition for a longer period of time and is more susceptible to noise.

The input slope is restored or improved by using regenerative gates. The CMOS logic gates are regenerative as they improve the slope of the waveform while passing the signal transition from the input to the output. In our new variable input delay gate design by inserting resistance, we use this regenerative property of the CMOS gates in the output for restoring the slope. However, the slope restoration also has limits and hence, the CMOS gates cannot improve very slowly varying inputs. Thus, there is a practical limit to degrading the input slope. This means that there is a practical limit to the amount of resistance that
Figure 7.5: The proposed single added nMOSFET variable input delay NAND gate. (a) Transistor Level showing the nMOS transistor added and (b) charging path for transitions along the different paths through the gate.

can be added in the series path. This is one of the major factors that influence the practical value of $u_b$ of a given technology.

### 7.3.3 Proposed Gate Design

The transistor, when ON, behaves like a resistor with an ON resistance $R_s$. We use this concept to increase the resistance of the charging path of the gate. We insert a single nMOS transistor that is always ON in the series charging path. A modified NAND gate is shown in Figure 7.5. The delays of the gate along both I/O paths are given by:

\[
d_{2\to3} = R_{on} \times C_L \\
d_{1\to3} = (R_{on} + R_s) \times C_L \\
= R_{on} \times C_L + R_s \times C_L \\
= Output\ Delay + Input\ Delay
\]

Thus, the input and output delays are separated completely from each other. The output delay can be controlled by sizing the gate transistors and the input delay can be controlled by altering $R_s$ offered by the series transistor. We have independent control over the input and output delays. The delay along the path $2 \to 3$ is not affected by altering the delay along the path $1 \to 3$ using the nMOS
transistor. This concept can be extended to a \( n \)-input gate. The differential delay of path \( x \) with respect to the other \( n - 1 \) paths, can be controlled by inserting \( n - 1 \) transistors in series with the inputs. These paths can be independently controlled by sizing the \( n - 1 \) transistors. Thus, we have a variable input delay gate design that is extendible to all gate types and multi-input gates as well.

### 7.3.4 Calculation of \( u_b \)

As seen from Equation 7.11, the input delay can be controlled independently by altering the size of the nMOS transistor. The nMOS transistor passes logic 0 effectively but degrades the signal when passing logic 1. Let us assume that there is a degradation of voltage \( \lambda \) when a logic 1 is passed through the transistor [152, 197]. When the transistor is acting as a resistor, there is an \( IR \) voltage drop also across the capacitor. The drop can be significant for two reasons:

- If the drop is too large, then the transistors in the fanout will not switch OFF completely. This increases the short circuit dissipation of the gate in the fanout.
- The leakage power of the transistors is a function of the gate to source voltage \( (V_{gs}) \). Hence, larger drop would increase the leakage current of the gate in the fanout.
Let us analyze the functioning of the circuit in Figure 7.6(a). The circuit shows a single transistor pair at the output of the nMOS for simplicity, but the transistor pair is actually a part of the multi-input gate in the fanout. Now for the correct functioning of the gate in the fanout when a logic 1 is incident at the input, the nMOS transistor has to remain in the linear region and the pMOS transistor has to be in cutoff. The critical condition in this configuration is the pMOS transistor remaining in cutoff. If this condition is not met, the pMOS transistor remains in either the linear or saturation region (both mean ON) and, hence, there is a direct path from the supply to the ground. This increases the short circuit dissipation of the circuit. To meet the condition, we need to make sure that the gate voltage of the pMOS transistor is below the threshold voltage for switching.

\[ V_g > V_{dd} - V_{tp} \quad (7.12) \]

where \( V_{tp} \) is the threshold voltage of the pMOS transistor. There are two factors that control the input voltage \( V_g \) is this case:

- The drop across the series transistor \( I_{ds}R_s \), where \( I_{ds} \) is the drain to source stand-by current through the series transistor and

- The signal degradation \( \lambda \) [152].

Hence, the condition for the pMOS transistor to remain in cutoff is given by:

\[ V_{dd} - \lambda - I_{ds}R_s \geq V_{dd} - V_{tp} \quad (7.13) \]

\[ R_s \leq \frac{V_{tp} - \lambda}{I_{ds}} \quad (7.14) \]

Consider the input configuration in Figure 7.6(b). The transistor passes a logic 0 in this case. The nMOS transistor passes a logic 1 without any degradation and hence the only degradation here is the signal drop \( I_{ds}R_s \). For correct functioning, the pMOS transistor has to be in the linear region and the nMOS transistor has to be in cutoff. The critical condition here is the nMOS transistor in cutoff. If the nMOS transistor is not completely switched OFF, it would short the supply and
the ground leading to short circuit power. By using a similar analysis as above, the condition is given by:

\[ I_{ds}R_s < \frac{V_{tn}}{I_{ds}} \]  
\[ R_s < \frac{V_{tn}}{I_{ds}} \]

Equations 7.14 and 7.16 give the upper bound on \( R_s \). This limits the amount of resistance that can be added to the charging path. Thus, the amount of input delay that can be added is also limited by this condition.

\[ d_{diff} = R_{max} \times C_L \]  
\[ d_{min} = R_{on} \times C_L \]  
\[ u_b = \frac{d_{diff}}{d_{min}} \]  
\[ u_b = \frac{R_{max}}{R_{on}} \]

where \( R_{max} \) is the maximum resistance that can be added and \( C_L \) is the load capacitance of the gate. Thus, \( u_b \) for a particular technology can be calculated. We have chosen a \( u_b = 10 \) for the 0.25\( \mu \)m technology. Note that this is not a hard limit and can be increased for the same technology. The \textit{theoretical limit} of \( u_b \) calculated in this section is based on the correct operating regions of the transistors. But the practical limit is governed by signal integrity issues. The resistance added degrades the input slope of the gate and this could lead to noise margin issues making the \( u_b \) smaller than the theoretical limit calculated here. As described in Chapter 4, the Schmitt trigger circuit can be used for improving the slope of the waveform. The Schmitt trigger itself consumes power and hence, is not an option in this low power technique. Hence, to keep the input slope degradation within limits, we need to limit the delay obtainable by the transmission gates to \( u_b \).

### 7.3.5 Design Issues

The new variable input delay gate design using the single nMOS transistor is an improvement on the capacitance manipulation design because of the independent
control of the input and output delays. But there are certain design issues that need to be considered.

- The theoretically calculated $u_b$ can be reduced by limits on the dimensions of the series transistors. This can be a practical limit governed by the technology parameters and signal integrity issues due to increased slew.

- The short circuit dissipation is a function of the ratio of the input and output waveform slopes. By inserting the resistance we are increasing the slope of the input waveform. This increases the short circuit dissipation by a slight margin.

- The leakage power is a function of the gate to source voltage ($V_{gs}$). Due to the signal degradation, the drop across the $n$MOS transistor is increased while passing logic 1. Hence, the leakage power through the transistors in the fanout is increased while passing logic 1. This drawback is alleviated in the design discussed in the next section.

- This design increases the transistor count in the circuit as it requires the addition of extra transistors. However, the addition of these transistors does not add to switching power but only increases the area.

### 7.4 Gate Design with CMOS Pass Transistors

The problem with the single $n$MOS transistor added is because of the degraded logic 1 value. This can be a problem as leakage power is becoming significant in present-day technologies and cannot be neglected. Hence, we propose another variant of the single $n$MOS transistor design by adding a CMOS pass transistor instead. The modified circuit for a NAND gate is shown in Figure 7.7. The concept is similar to the single $n$MOS transistor addition because it also increases the resistance of the series charging path. The CMOS pass transistor consists of an $n$MOS and a $p$MOS transistor connected as shown. Both transistors are kept always ON by connecting the gates to $Gnd$ and $V_{dd}$, respectively. The advantage
Figure 7.7: The proposed CMOS added pass transistor variable input delay NAND gate. (a) Transistor Level showing the nMOS and pMOS transistors added and (b) Charging path for transitions along the different paths through the gate.

of this design is that the signal is not degraded while passing either logic 1 or logic 0. This does not increase the leakage power of the transistors in the fanout due to non-zero $V_{gs}$.

### 7.4.1 Calculation of $u_b$

The $u_b$ calculation is similar to the single nMOS added design but without the factor for signal degradation. Using a similar analysis we can deduce:

1. $V_{dd} - I_{ds}R_s > V_{dd} - V_{tp}$  
   
2. $R_s < \frac{V_{tp}}{I_{ds}}$  
   
3. $I_{ds}R_s < V_{tn}$  
   
4. $R_s < \frac{V_{tn}}{I_{ds}}$  
   
5. $d_{diff} = R_{max} \times C_L$  
   
6. $d_{min} = R_{on} \times C_L$  
   
7. $u_b = \frac{d_{diff}}{d_{min}} = \frac{R_{max}}{R_{on}}$  

Note that the resistance $R_s$ is the effective parallel resistance of both the transistors together. Hence, the $u_b$ can be calculated for any technology given these
parameters. We have achieved $u_b = 10$ for the 0.25μm technology.

7.4.2 Design Issues

The design issues involved in this gate design are:

- The resistance calculated is the effective series/parallel resistance of both the nMOS and the pMOS transistors. The effective resistance is reduced because both transistors are in parallel to the current path. Hence, the transistors have to be longer to achieve the same resistance as a single nMOS transistor. Hence, effective resistance per unit length reduces.

- Since effective resistance per unit length reduces by using CMOS gates, the transistors have to be bigger and, hence, the area increase is higher.

- In a CMOS pass transistor, there is a parasitic charging path from the gate of the nMOS FET to the gate of the pMOS FET through the sidewall. In the proposed gate design, since the gates of the transistors are connected to Gnd and Vdd, this parasitic path serves as a short circuit. Although the path conducts negligible current in present day technologies, it might become significant in future technologies and needs to be accounted for.

7.5 Technology Mapping

In the previous section we have described the various implementations for the variable input delay gate. We have seen that the LP gives delays that need to be implemented in the circuit for every gate. The transistor sizes for every gate have to be manipulated to achieve the required delay. The process of designing gates for a specific delay by altering the dimensions of the transistors is called technology mapping or transistor sizing. In this section we describe the procedure that we used to size the gates in the circuit.
Figure 7.8: Flowchart showing the standard cell place-and-route. The different cells for every gate instance have to be present in the Cell Library.

### 7.5.1 Place-and-Route Design Flow

For shorter design cycles we propose to design the circuit using the *place-and-route* design flow. In this design flow, the circuit components are broken up into logic gates. Each logic gate has a different *standard cell* that can be used in the flow. A typical standard cell design flow is shown in Figure 7.8.

The library contains different standard cells for different delay requirements. The netlist of the circuit contains the various gates, gate types and their interconnection details. After the LP optimization, the delays of each gate in the circuit are decided. The delay assignments from the LP are independent of the gate type, since it is a logic-level optimization. For example, a NAND gate with 2 inputs might have a delay assignment of 2 units for one instance and 4 units for another instance. Thus, a single cell for a 2-input NAND gate in the library is not sufficient to realize the delays. Moreover, the delay is dependent on the load capacitance, which varies from instance to instance. Hence, we need to size the cells according to their load capacitance in that instance and also according to the delay requirement. The parameters that we can change are the dimensions of the transistors in a particular instance. Hence, we modify this design flow to dynamically change the cells in the library according to the delay requirements and the load capacitance of the instances.
7.5.2 Delay Components

We have seen that the delay of the gate is a function of the load capacitance \( C_L \).
The load capacitance at the output of a net is a function of the gate capacitances of all of the gates in the fanout and their routing capacitances:

\[
C_L = C_r + \sum_{i=1}^{n} C_{in,i}
\]  \hspace{1cm} (7.29)

where \( C_r \) is the routing capacitance at the output, \( n \) is the number of fanouts at the gate output and \( C_{in,i} \) is the input capacitance of the fanout \( i \). So for every instance of a gate to achieve the required delay assignment, we need to guarantee that the delay is achieved at this load capacitance. The input capacitance \( C_{in,i} \) is a function of the transistor dimensions as well. Hence, the dimensions of the gates in the fanout have to be decided before the gate dimensions can be changed for a delay assignment. We use a reverse breadth first search methodology for designing the entire circuit and this is described in Chapter 10. The design of a single gate with a given load capacitance is described in this section.

7.5.3 Search Dimensions

The objective is to design a gate with a load capacitance \( C_L \) in a particular instance, in order to have a required delay \( d_{\text{req}} \). The procedure is searching for the appropriate dimensions for all of the transistors in the circuit that satisfy the two criteria. The dimensions for the search space of an \( n \)-input gate are:

- \( n \) pMOS transistor widths
- \( n \) pMOS transistor lengths
- \( n \) nMOS transistor widths
- \( n \) nMOS transistor lengths
- Load capacitance.
In circuit designs, gates have up to 5 inputs. In such cases, the dimension of the search space becomes 21, which is huge and intractable. Moreover, the dimensions of all the pMOS transistors and the nMOS transistors have to be the same for equal inertial delay along all paths. Hence, the search dimension reduces to:

- pMOS transistor width
- pMOS transistor length
- nMOS transistor width
- nMOS transistor length
- Load capacitance.

Searching the 5 dimensional space for every gate can also be time consuming. Also the load capacitance is a continuous quantity and we cannot have a single lookup table approach. Hence, we propose a two step approach. The first is a quick lookup from a lookup table that has the dimensions for certain load capacitances. This can be used as a first guess for the sizes to be used in the second step. In the second step, we use recursive simulation for fine tuning the dimensions for the particular load capacitance.

### 7.5.4 Lookup Table Generation

The lookup table is a coarse approximation of the various dimensions for a given delay requirement and the given load capacitance. We first decide upon certain discrete load capacitances that span the range of capacitances that might occur within the circuit. Then we decide upon the various delays that the LP is likely to generate in the delay assignments. This can be an even distribution of the range of delays in the circuit. The safest range would be from 1 to maxdelay of the circuit. The lookup table is useful to generate only if the optimization is going to be used for different circuits. This is done once and can be used for any
circuit in the same technology. But if a single design is needed, then it may not be advantageous to generate the lookup table.

Now the gate types are divided into 6 main categories:

- NAND
- NOR
- INV or inverter
- BUF or buffer
- OR
- AND

The derived gates such as XOR and XNOR can be used as a combination of the other gates. For generating a cell to be used in the design flow, we need to have different cells for different numbers of inputs as well. For example, the NAND gate type may have four different cells in itself, such as a 2-input NAND, a 3-input NAND and so on. Thus, in our current discussion we will refer to each of them as a single gate type. Hence, in the following discussion a 2-input NAND gate is perceived as a different gate type from a 3-input NAND gate cell.

For every gate type we need to generate the transistor dimensions for a particular delay assignment and a particular load capacitance. The procedure is shown in Figure 7.9. We use the Spectre analog simulator for the measurement of delay. The load capacitance is attached as a single capacitor at the output of the gate. The procedure is driven by two quantities: *rise delay* and *fall delay*. Each gate type under consideration is given a pair of input transitions, one rising and one falling. The resulting rise and fall delays, \( d_{\text{rise}} \) and \( d_{\text{fall}} \), respectively, of the gate are measured. The decision taken is:

1. If \( d_{\text{rise}} < d_{\text{req}} \), then increase the pMOS length of the transistor by 1.
2. If \( d_{\text{rise}} > d_{\text{req}} \), then increase the pMOS width of the transistor by 1.
3. If \( d_{\text{fall}} < d_{\text{req}} \), then increase the nMOS length of the transistor by 1.

4. If \( d_{\text{fall}} > d_{\text{req}} \), then increase the nMOS width of the transistor by 1.

Since the dimensions for every gate type are started from the lowest dimension, the method actually reaches the first possible implementation of the required delay. The problem with the method is the quantization of the transistor sizes. The sizes can only be varied in certain discrete increments. Hence, it might be impossible to get the exact delay at the gate output. We set an error percentage within which the delay error is acceptable. The condition for the convergence of the delay is:

\[
\frac{|d_{\text{req}} - d_{\text{rise}}| + |d_{\text{req}} - d_{\text{fall}}|}{d_{\text{req}}} < \epsilon
\]

where \( \epsilon \) is the error percentage, which can be assumed normally to be 2-3\%.

This error is acceptable as the delay is only a coarse adjustment of the delay assignment. Once the procedure converges for a load capacitance, the procedure is repeated for another load capacitance. Once all capacitances in the range considered are done, another delay assignment is considered. Thus, a complete lookup table is generated for every gate type and delay assignment.
7.5.5 Fine Tuning the Transistor Size Assignment

The lookup table is generated for all of the different gate types and some load capacitances. This can be used in common to all of the different circuits. But when a particular circuit is being optimized, the load capacitances might fall between two load capacitances for which the lookup table is generated. In such cases we need to fine tune the dimension assignment for that particular load capacitance. The starting point for this procedure is the closest dimension assignment in the lookup table. This procedure is driven by the sensitivities of each of the dimensions. The sensitivity is defined by:

\[
\text{Sensitivity} = \frac{d_{\text{current}}}{|d_{\text{req}} - d_{\text{rise}}| + |d_{\text{req}} - d_{\text{fall}}|}
\]

where \(d_{\text{current}}\) is the present measured delay of the gate, and \(d_{\text{rise}}\) and \(d_{\text{fall}}\) are the rise and fall delays after the perturbation in the dimension. Each dimension has a sensitivity when perturbed by one unit. The perturbation can be an increase or a reduction in the dimension. There can be 8 perturbations, one for each of the dimensions. The perturbation with the highest sensitivity is incorporated and the gate is simulated again. This procedure is called the steepest descent method as the objective function is minimized by driving the dimensions based on sensitivities. The procedure can be summarized as:

1. Find the sensitivity of each dimension for perturbations
2. Choose the perturbation with highest sensitivity
3. Incorporate the perturbation into the dimensions of the gate
4. Check the error function \(\epsilon\)
5. If \(\text{error} > \epsilon\) go to Step 1 else END.

This procedure is used for the final dimension assignment for the transistors. The complexity is greatly reduced by using the lookup table as the search is limited to the neighborhood of the solution. Hence, local minima will not be a problem.
The procedure can also be tuned for including the area of the cell in the objective function.

### 7.6 Original Contributions

We have presented the *variable input delay logic*, which consists of gates with different delays along different I/O paths through a single gate. We have also presented three possible implementations of the variable input delay gate. This is the first technique in which such gates were designed. We have thoroughly analyzed the various designs proposed and presented their shortcomings. This is the first time such gates have been used for low power applications. The procedure for transistor sizing has been split into two stages, such that the gates can be designed differently for each gate instance. The complexity of the transistor sizing is reduced commendably due to the two-step approach, as the neighborhood of the solution is reached through a single lookup.

### 7.7 Summary

In this chapter, we presented the various components of gate delay. We then explained why conventional CMOS gates cannot be used as variable input delay gates. We presented three new implementations of the variable input delay gate. We presented an analysis of each of the gates and listed their shortcomings. Once the basic gate design topology is selected, the dimensions of the transistors need to be fixed. We explained why transistor dimensions of each instance need to be different while optimizing an entire circuit. Then we proposed a two-step approach for fixing the transistor sizes of every instance in the circuit. The entire chapter focused on the implementation of the delay assignments received from the LP for optimization. We present the results of the LP optimization in the next chapter. The entire design methodology and the tools used will be presented in Chapter 9.
Chapter 8

Results

In the previous chapters, we have presented the new variable input delay logic and the transistor-level implementation for glitch power reduction. In this chapter, we present the power consumption results and the speed of operation achieved by using this technique. We present the results on an example circuit first and analyze the benefits of the method. Then we present the power savings results at the logic-level for all of the ISCAS’ 85 benchmark circuits. We compare the results to the buffer insertion technique as well [156, 157]. The entire design flow and the results from the layout extraction are presented in the next chapter.

8.1 Experimental Procedure

Consider the flowchart shown in Figure 8.1. This is the experimental procedure for generating the power consumption results at the logic level. We write a C++ program to generate the constraints from the logic level netlist. The constraints are then fed to AMPL, a linear programming tool [58]. The results from the LP are in the form of delay assignments for every gate in the circuit. The maxdelay achieved is also given as the output of the LP. The delay assignments are then used in an event-driven simulator for power analysis. This is the power estimation technique based on counting transitions in the circuit and weighting them with the fanout of each gate [77]. The \( u_b \) is taken as an input to the procedure from the technology. Since this is a logic-level analysis it is not technology specific. Hence, we present results for different \( u_b \) values as well. The purposes of the logic level analysis are:
Figure 8.1: Flow chart depicting the experimental procedure for generating the results at the logic level.

- To check the speed of the circuit
- To see how much power can be saved by the technique
- To see if the power saved is significant enough to go through with the rest of the optimization procedure.

First we present some results on the speed of the circuit obtained and then we move on to the power saving results.

### 8.2 Feasible Gate Differential Delay Upper Bound

The *gate differential delay upper bound* ($u_b$) is a measure of the flexibility we have in terms of designing a gate with different I/O path delays through the gate. Theoretically, we proved that as the flexibility of the design increases, the circuits get progressively faster. In order to prove it experimentally, we ran the LP formulation on the ISCAS’ 85 benchmark circuits for different feasibility bounds. The results are shown in Figure 8.2. Each curve in the figure corresponds to
Figure 8.2: Normalized maxdelay versus $u_b$. The maxdelay is normalized to the fastest possible circuit design, i.e., without altering delays along the critical path.

a different circuit. We have used the minimum dynamic power LP described in the previous chapters for this evaluation. The minimum dynamic power LP gives the fastest possible realization of the circuit at a given $u_b$. We can see that as the feasibility upper bound is increased we have lower maxdelay and hence a faster circuit. All the circuits designed for this evaluation consume the minimum dynamic power as no buffers are added.

8.3 Example Circuit

Let us analyse the merits of the proposed technique using a simple example circuit. In this section we will describe the implementation of the technique on a simple combinational circuit and present the power and energy savings results. We also consider the buffer insertion technique by Raja et al. for comparison [156, 157].
Figure 8.3: An example circuit.

**Unoptimized Circuit**

Consider the simple example circuit of Figure 8.3. Assume that the delays of every gate are the minimum allowed by the technology. We observe that the differential delay, at gates 5 and 6, exceeds the inertial delay and we expect these gates to glitch. The circuit was simulated for rising signals at all three inputs and the waveforms are shown in Figure 8.4. The simulation was done using the *Spectre* circuit level simulator from *Cadence* [28]. As expected, gates 5 and 6 transition 2 and 3 times, respectively. These are the glitches we wish to eliminate in the following design.

**Buffer Insertion Technique**

The buffer optimization using conventional gate design requires the use of one buffer for the circuit to operate at the same speed. The optimized circuit with the buffer is shown in Figure 8.5. The simulation was done using *Spectre*. The buffer optimized circuit was simulated for the same vector-pair as the unoptimized circuit and the waveforms are shown in Figure 8.6. As expected, the buffer optimization has eliminated the glitches.

**Low-Power Design with the Proposed Gate**

Instead of the buffer, the variable input-delay gate is designed and the optimized circuit is shown in Figure 8.7. We have used the single nMOS transistor implementation in this but any of the proposed designs could have been used.
The simulation was done using Spectre. This circuit was simulated for the same vector-pair as the previous circuits and the output waveforms are shown in Figure 8.8. The optimized circuit with the proposed gate design also eliminates the glitches effectively.

### 8.3.1 Energy Consumption in the Example Circuit

The energy consumption of a circuit is a function of the current drawn by it from the supply. During our simulations for the three circuits described above, we
measured the input current for the given input vector and computed the energy. The energy consumption results are shown in Figure 8.9. The simulations were done with the Spectre analog simulator from Cadence. As seen in the Figure, the unoptimized circuit consumes 800fJ, the buffer optimized circuit consumes 550fJ and the new gate optimized circuit consumes 300fJ. Thus the energy savings of the new design are 62.5% with respect to the unoptimized circuit. The new gate design achieves 36.8% more savings than the buffer optimized design with respect to the unoptimized circuit.

8.3.2 Leakage Current Analysis for the Example Circuit

The introduction of a single nMOS transistor degrades the signal at the gates of the transistors. This increases the leakage current of the circuit and may even drive the transistors out of cut-off. The current flowing in steady state is called the quiescent current and is due to the leakage through the different transistors. The quiescent current is a function of the input vectors at the PIs of the circuit. To analyze the relative effect, we have simulated the above circuit with two different input vectors and let the circuit settle for a long time. The results are shown in Figure 8.10. Three circuits were considered in the simulation: an unoptimized circuit, an optimized circuit with single nMOS transistors used to implement the variable input delay gates and the third with CMOS transmission gates used to implement the variable input delay gates. The leakage currents for the vector
Figure 8.6: Output waveforms for one input transition at the PIs of the buffer optimized circuit. Glitches are eliminated.

000 show no change for the three circuits as in this state the nMOS transistors are passing logic 0, which is not degraded. For vector 111, however, there is an increase of 0.45% in leakage current due to the nMOS transistor addition. The circuit with the CMOS transmission gates had an increase of only 0.2%. This increase is not due to the degradation in value but due to the leakage path added from $V_{dd}$ to $Gnd$ through the sidewall capacitance. This is a very minor increase for the present-day 0.25μm fabrication technology but further analysis needs to be done for more recent technologies.
Figure 8.7: Optimized example circuit with the proposed gate.

8.4 Power Savings for Benchmark Circuits Using the *Minimum Dynamic Power* LP

The *minimum dynamic power* LP optimizes the speed of the circuit keeping the power at the minimum value. No buffers are inserted by this procedure. We have chosen a $u_b$ of 10 for the 0.25μm technology. As described above, this is not a hard limit but a chosen one for illustration. The LP was written for the ISCAS’ 85 benchmark circuits and solved using AMPL [58]. The resulting maxdelay of the circuit is noted. The resulting delay assignments are used in the delay simulator for the power estimation analysis as described by Hsiao et al. [77]. The LP gives the optimal set of delays for the gates and also the delay elements. The circuits were then simulated using a variable delay simulator with the delays taken as output from AMPL. The results on the ISCAS’ 85 benchmark circuits are shown in Table 8.1. The vectors used are the uncompacted ATPG test vectors for each of the circuits. Each unoptimized gate is assumed to have a delay of one unit. This is the smallest possible delay realizable at the physical level in that technology. The maxdelay and $u_b$ are shown in the same delay units. The normalized delay is the critical path delay normalized to the maxdelay of the fastest possible implementation of the circuit.
8.5 Power Savings for Benchmark Circuits Using the Delay Specification LP

The minimum dynamic power LP produces the minimum energy consuming circuit but the speed achieved may not be the desired one. Hence, we proposed the delay specification LP where the procedure inserts minimum number of buffers to the minimum dynamic power design to speed the circuit to specification. Since the number of buffers inserted in this technique is lesser than the number of buffers inserted by the linear LP technique of Raja et al., we achieve more power.
Figure 8.9: Energy consumption comparisons of the three different circuits in fJ. The new gate optimized circuit consumes 62.5% less energy than the unoptimized circuit and 45% less energy than the buffer optimized circuit.

savings. We verify the power savings of all the ISCAS’ 85 benchmark circuits and compare them to the Raja et al.’s method in this section. The constraints are written in AMPL and the optimized delays for all the gates and the inserted buffers are obtained. The results are shown in Table 8.2. The vectors used are the uncompacted ATPG test vectors for each of the circuits. The normalized delay is the critical path delay normalized to the maxdelay of the fastest possible implementation of the circuit. Power is normalized to the unoptimized circuit consumption. The number of buffers inserted by the previous method is also shown.
Figure 8.10: Leakage current analysis for three different circuit configurations in fJ. The circuits considered were: (1) Unoptimized circuit, (2) Optimized circuit with nMOS pass transistors added and (3) Optimized circuit with CMOS pass transistors added. The leakage current is increased for input vector 111 by 0.2% with CMOS pass transistors and by 0.43% for nMOS pass transistors. The leakage current is unaffected for the input vector of 000.

The power savings are much better for the proposed method than the existing methods. For example, for the circuit c432 the power savings are increased by 24% for the same maxdelay. The insertion of 95 buffers for the same case has also been eliminated by the new method. Thus the proposed method performs better in terms of power savings at the same speed without the insertion of delay buffers.
Table 8.1: Dynamic power dissipation in ISCAS’85 benchmark circuits for the proposed (no buffer) design method. We used the *minimum dynamic power* LP and chose \( u_b = 10 \). The method gives the fastest minimum dynamic power circuit for the chosen \( u_b \) value. *Maxdelay* is normalized to the fastest possible circuit before optimization, i.e., the length of the *critical path* of the circuit. Power measurements are done by using a weighted fanout variable delay simulator [77]. The table shows the optimized power consumption normalized to the power consumption of the unoptimized version of the circuit. Power savings are (1-value shown in the table).

<table>
<thead>
<tr>
<th>Circuit</th>
<th>No. of Vectors</th>
<th>maxdelay</th>
<th>Norm. Delay</th>
<th>Unoptimized Avg.</th>
<th>Peak</th>
<th>Optimized Avg.</th>
<th>Peak</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>56</td>
<td>71</td>
<td>4.17</td>
<td>1.0</td>
<td>1.0</td>
<td>0.65</td>
<td>0.55</td>
</tr>
<tr>
<td>c499</td>
<td>54</td>
<td>34</td>
<td>2.26</td>
<td>1.0</td>
<td>1.0</td>
<td>0.70</td>
<td>0.65</td>
</tr>
<tr>
<td>c880</td>
<td>78</td>
<td>45</td>
<td>1.50</td>
<td>1.0</td>
<td>1.0</td>
<td>0.48</td>
<td>0.45</td>
</tr>
<tr>
<td>c1355</td>
<td>87</td>
<td>67</td>
<td>2.05</td>
<td>1.0</td>
<td>1.0</td>
<td>0.47</td>
<td>0.36</td>
</tr>
<tr>
<td>c1908</td>
<td>144</td>
<td>173</td>
<td>4.32</td>
<td>1.0</td>
<td>1.0</td>
<td>0.54</td>
<td>0.44</td>
</tr>
<tr>
<td>c2670</td>
<td>82</td>
<td>35</td>
<td>1.09</td>
<td>1.0</td>
<td>1.0</td>
<td>0.68</td>
<td>0.56</td>
</tr>
<tr>
<td>c3540</td>
<td>200</td>
<td>347</td>
<td>7.38</td>
<td>1.0</td>
<td>1.0</td>
<td>0.53</td>
<td>0.43</td>
</tr>
<tr>
<td>c5315</td>
<td>157</td>
<td>542</td>
<td>11.06</td>
<td>1.0</td>
<td>1.0</td>
<td>0.53</td>
<td>0.44</td>
</tr>
<tr>
<td>c6288</td>
<td>141</td>
<td>124</td>
<td>1.87</td>
<td>1.0</td>
<td>1.0</td>
<td>0.22</td>
<td>0.18</td>
</tr>
<tr>
<td>c7552</td>
<td>158</td>
<td>50</td>
<td>1.16</td>
<td>1.0</td>
<td>1.0</td>
<td>0.28</td>
<td>0.26</td>
</tr>
</tbody>
</table>

8.6 Summary

In this chapter, we have seen the logic level results for the technique. The technique produces faster designs as the flexibility allowed by the technology is increased. We have shown the logic and transistor level results on an example circuit. The results show a decrease of up to 58% in the energy consumed by the circuit. The leakage current increases for certain input configurations but this effect can be countered by using CMOS pass transistors instead of a single nMOS transistor. We have also presented the power saving results on some of the ISCAS’85 benchmark circuits. The proposed technique achieves 18% more savings than the previously published techniques. However, at the logic level we cannot predict the affect of the technique on area, routing, parasitics, etc. Hence, we present the entire methodology of the physical design of a circuit c7552, in the next chapter.
Table 8.2: Power measurement results of the proposed delay specification LP on all ISCAS’ 85 benchmark circuits. The circuits are designed for two delay specifications given by maxdelay, and for $u_b = 10$. The number of buffers inserted by this method are compared with the published results of Raja et al. [156, 157]. Maxdelay is normalized to the fastest possible circuit before optimization, i.e., the length of the critical path of the circuit. Power measurements are done by using a weighted power out variable delay simulator [77]. The table shows the optimized power consumption normalized to the power consumption of the unoptimized version of the circuit. Power savings are (1-value shown in the table).

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Norm. maxdelay</th>
<th>No. of Vectors</th>
<th>Unopt.</th>
<th>Raja et al. [157]</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Avg</td>
<td>Peak</td>
<td>Bufs</td>
</tr>
<tr>
<td>c432</td>
<td>1.0</td>
<td>56</td>
<td>1.0</td>
<td>0.72</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>56</td>
<td>1.0</td>
<td>0.62</td>
<td>0.60</td>
</tr>
<tr>
<td>c499</td>
<td>1.0</td>
<td>54</td>
<td>1.0</td>
<td>0.91</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>54</td>
<td>1.0</td>
<td>0.70</td>
<td>0.66</td>
</tr>
<tr>
<td>c880</td>
<td>1.0</td>
<td>78</td>
<td>1.0</td>
<td>0.68</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>78</td>
<td>1.0</td>
<td>0.68</td>
<td>0.52</td>
</tr>
<tr>
<td>c1355</td>
<td>1.0</td>
<td>87</td>
<td>1.0</td>
<td>0.58</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>87</td>
<td>1.0</td>
<td>0.57</td>
<td>0.48</td>
</tr>
<tr>
<td>c1908</td>
<td>1.0</td>
<td>144</td>
<td>1.0</td>
<td>0.69</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>144</td>
<td>1.0</td>
<td>0.59</td>
<td>0.44</td>
</tr>
<tr>
<td>c2670</td>
<td>1.0</td>
<td>82</td>
<td>1.0</td>
<td>0.79</td>
<td>0.65</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>82</td>
<td>1.0</td>
<td>0.71</td>
<td>0.58</td>
</tr>
<tr>
<td>c3540</td>
<td>1.0</td>
<td>200</td>
<td>1.0</td>
<td>0.64</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>200</td>
<td>1.0</td>
<td>0.58</td>
<td>0.46</td>
</tr>
<tr>
<td>c5315</td>
<td>1.0</td>
<td>157</td>
<td>1.0</td>
<td>0.63</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>157</td>
<td>1.0</td>
<td>0.60</td>
<td>0.45</td>
</tr>
<tr>
<td>c6288</td>
<td>1.0</td>
<td>141</td>
<td>1.0</td>
<td>0.40</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>141</td>
<td>1.0</td>
<td>0.36</td>
<td>0.34</td>
</tr>
<tr>
<td>c7552</td>
<td>1.0</td>
<td>156</td>
<td>1.0</td>
<td>0.38</td>
<td>0.34</td>
</tr>
<tr>
<td></td>
<td>2.0</td>
<td>156</td>
<td>1.0</td>
<td>0.36</td>
<td>0.32</td>
</tr>
</tbody>
</table>
Chapter 9
Physical Design and Implementation of the c7552 Benchmark Circuit

In previous chapters, we have presented a new variable input delay CMOS logic and the proposed gate design implementations. We have also seen that the technique saves up to 68% of the power in circuits at the logic level. But logic-level savings do not translate into layout-level savings. There are many effects such as parasitics, routing capacitances, etc. that need to be taken care of. Also for optimizing large circuits, it is imperative to have an automated design methodology. In this chapter, we present an automated design methodology for optimizing circuits using the proposed technique. We test the design methodology on the largest benchmark circuit c7552 and present the measured energy consumption results.

9.1 Overall Design Flow

The overall design flow is shown in Figure 9.1. The flow chart shows the various stages in the design process, including the various tools that are used. In this section we describe the functioning of each of the tools, some of which are commercially available and some of which have been written for this specific purpose. We try to explain the entire methodology by explaining each step in detail. This design process can be used for optimizing any generic circuit. The results of each stage during the design of the c7552 circuit are presented at the end of the chapter.
Figure 9.1: Flow chart showing the entire design flow for automated design of large circuits.

9.1.1 Logic-level Constraint Generation

The logic-level constraints are generated using a C++ program. The program takes in a circuit level netlist and generates the constraints in AMPL format. AMPL is a linear programming tool that minimizes an objective function based on given constraints. The program uses a CPLEX solver to generate the solution. The output of the program is a set of optimized delays for every gate in the circuit and also the extra input delays that need to be realized in the circuit. The results are used in the technology mapping phase which is described below.
9.1.2 Technology Mapping

Once the delays of a gate are decided, they need to be implemented by changing the transistor dimensions in the gate. In Section 7.5, we have described the approach for designing a particular gate for a given delay assignment. We call this procedure sizegen in the current discussion. This procedure needs to be scaled for the gates in the entire circuit. We use a reverse breadth-first search for designing the gates. We use a first-in-first-out (FIFO) queue as the basic data structure in the process. The procedure can be summarized as:

1. Place all of the fanins of all POs in the FIFO
2. Get a gate \( i \) from the FIFO
3. Size the transistors using sizegen
4. Place fanins of gate \( i \) in the FIFO
5. If the FIFO is not empty, go to Step 2 else END.

Thus, the sizes of all the gates in circuit are determined.

9.1.3 Standard Cell Generation Using Progenesis

The technology mapping tool generates the sizes of all of the transistors in the circuit. Now, we wish to incorporate this into the standard cell place-and-route design flow. The design can also be done using full custom design where each gate is hand crafted. But for faster design cycles, we used the standard cell place-and-route. For this procedure we need to generate the different standard cells for each of the different sizes that are generated in the technology mapping stage. We used an automatic program called Progenesis for generating the standard cells [148]. The Progenesis software takes in a SPICE netlist of the gate, with the sizes of each transistor and a rule file [132]. The rule file contains the height of the cells, the routing grid being used and the technology-dependent spacing rules.
for the different layers. The program generates the standard cells for the desired functionality, and the desired sizes of transistors that are given by the technology mapping. The cells are given in the general data stream (GDS) format [29]. This is a standardized format used for interchanging data between CAD tools.

9.1.4 Abstract Generator

The place-and-route tools take the information in the form of a library exchange format (LEF). To generate this format from the GDS output of the Progenesis tool, we use a tool called Abstract Generator [29]. This tool takes in the entire library of cells in GDS format and generates the LEF [29]. It is a part of the Cadence design suite.

9.1.5 Verilog Generator

This is a C++ program that converts the circuit netlist into verilog format. This is a standardized format for circuit description. We generate the structural description of the circuit. The standard cells in the circuit are changed by technology mapping. Hence, the appropriate cells have to used for every gate instance. We generate the Verilog netlist after the technology mapping based on the standard cell linked to each gate instance.

9.1.6 Silicon Ensemble

This is the place-and-route tool which is a part of the Envisia tool suite from Cadence. This tool takes in the verilog netlist of the circuit, the LEF of the entire standard cell library. We specify the floor plan, power line widths and other design parameters in this tool. It generates the complete layout of the circuit. The output is in the design exchange format (DEF) which can be imported back to the Cadence Layout Editor [29].
9.1.7 Extraction

The design is imported into the Cadence Layout editor. The design is then extracted using the Virtuoso layout extractor program. This tool is also a part of the Cadence design suite. The extractor converts the physical layout into a transistor schematic, with the parasitic capacitances for routing. The routing capacitances are then measured and the routing delays at all of the signal nets are calculated. The routing delays can alter the delay assignments of the gates and the circuit can glitch. To avoid this we need to account for the routing delays in the final optimized design. The procedure for incorporating the routing delays into the LP is described in the next section.

9.2 Incorporating Routing Delay into the LP Formulation

We have so far ignored the delay of the gate due to routing capacitance in the layout. In this section we present a technique to calculate the routing delays from the layout of the circuit and then incorporate them into the original LP. The routing delay of each signal net is measured as explained above. This routing delay is then normalized to the minimum delay of an inverter. The constraints in the LP can be reformulated into the following form. Consider the circuit in Figure 9.2. Let the interconnect delays be as shown on the nets. Now the modified

![Diagram of a combinational circuit with routing delays](image-url)
constraints are for the gate 7 would be:

\[
T_7 \geq T_5 + d_7 + 0.5 \\
T_7 \geq T_6 + d_7 + 2.5 \\
t_7 \leq t_5 + d_7 + 0.5 \\
t_7 \leq t_6 + d_7 + 2.5 \\
d_7 > T_7 - t_7
\]

The reason for this formulation is very simple. The earliest time of arrival of signal at gate 7 \((t_7)\) is possible only after the change of gate 5 \((t_5)\) and also after the delay of the interconnect \((0.5)\) and then the delay of the gate 7 \((d_7)\). Note that the interconnect delay from gate 6 to gate 7 is the sum of the delay of the stem and the branch at the multiple fanout.

The routing delays are added to the LP and the optimization cycle is repeated to get modified gate delays. These modified delays can then be used for the optimization of the circuit. If the place-and-route is done again, the routing delays are changed again. The process is repeated until the routing does not change significantly from pre-optimized to post-optimized.

### 9.3 Physical Design

We have implemented the circuit c7552 using the above procedure. The circuit contains 3,827 gates, with 207 PIs. We designed the unoptimized circuit using the generic standard cell design library with one instance for every gate type. The circuit is shown in Figure 9.3. The procedure described above was used for the optimization. We designed the optimized circuit using the single nMOS variable input delay gates. The optimized circuit is as shown in Figure 9.4. The circuit had an area increase of 14%. The power consumption results are explained in the next section. The optimized circuit using the buffer insertion method is also shown in Figure 9.5. This design needed 366 buffers for the same speed of
Figure 9.3: The layout of the unoptimized circuit c7552 designed using the generic standard cell design library.
operation. This design had an area increase of 24% over the unoptimized circuit.

9.4 Power Consumption Results

For measuring the power consumption of small circuits, the Spectre analog simulator can be used. But for large circuits, the simulator program crashes due to memory limits of the computer when run for vector sequences greater than 10. Hence, we have implemented a power estimator based on transition counting and actual load capacitances from the extracted layout. The program is similar to a
logic-level transition counter, but the transitions at every gate are weighted by the extracted load capacitance at the output of the gate from the layout. The circuit is then left idle for a long time to measure the stand-by power consumption. This power measurement was found to agree with the power measurement from the Spectre analog simulator to within an error of 1%. The results of instantaneous and average power measurements are shown in Figures 9.6 and 9.7. The circuits were simulated on the same set of 156 uncompacted ATPG test vectors. Leakage currents were measured using the Spectre analog simulator and added to the final results, to get an accurate estimate of the energy consumption. The results show a peak power savings of up to 68% and average power savings of up to 58%. For comparison we have simulated the buffer optimized design shown in Figure 9.5 using the same technique described above and the same vector set. The proposed technique using variable input delay gates consumed 18% less power compared to the buffer optimized circuit. The technique is thus scalable to larger circuits as well.

9.5 Summary

In this chapter, we described the entire design flow for the automated design of large circuits using this technique. We have described the various tools that were used in the optimization. This technique is incorporated into the place-and-route design flow in this case, but it can be used for full custom design as well. The entire design methodology was tested for designing the circuit c7552 with 3,827 gates. The optimized circuit had an area increase of 14% but achieved an average power savings of 58% and peak power savings of 68%. This proves that the method can be used for optimizing large circuits and the power savings are significant.
Figure 9.5: The layout of the optimized circuit c7552 designed using the buffer insertion technique proposed by Raja et al. [156, 157]. The area increased 24% compared to the unoptimized circuit in Figure 9.3. All three circuits function at the same speed of operation.
Figure 9.6: Instantaneous energy consumption results for benchmark circuit c7552 for 156 vectors. Results of the optimized circuit using the proposed method show a peak power savings of 68% over the unoptimized circuit.

Figure 9.7: Average energy consumption results for benchmark circuit c7552 for 156 vectors. Results of the optimized circuit using the proposed method show an average energy savings of 58% over the unoptimized circuit.
Chapter 10

Conclusion and Future Work

In this chapter, we summarize the entire work and provide some pointers for future work.

10.1 Conclusion

Glitches are a power wastage in CMOS circuits and need to be eliminated for low power applications. Prior glitch reduction techniques suffered from non-linearities, convergence problems, inserted extra buffers or could not handle large circuits. In this thesis, we have presented a new design methodology that can be used for designing glitch-free circuits without inserting buffers and non-linearities. The technique is based on a new variable input delay logic, which is a design style where logic gates have different delays along different I/O paths through a single gate. Such gates can be designed but have limitations to the amount of differential delay that can be realized at the inputs of the gate. This limit is a function of the technology in which the gates are designed and we call this the gate differential delay upper bound \( u_b \). It is the measure of the amount of flexibility a designer is allowed in designing the new gates. Every technology can be quantized by this single parameter. We formulate a new linear program using these gates and the \( u_b \) for the technology. The gate input delays are considered to be independent of each other when varied within the range \( u_b \). Hence, the formulation remains linear and can be solved using a linear programming tool. This is the first formulation that optimizes large circuits by keeping the formulation linear. The output of the LP is the set of optimized delays for every gate in the circuit. These
delays need to be realized at the transistor level. We proposed three new ways of designing these gates at the transistor level. Variable input delay gates can be designed by input capacitance manipulation, single nMOS transistor addition or a CMOS pass transistor addition. There are advantages and disadvantages of each of these designs that are described. We present a two-step technology mapping algorithm that sizes the transistors of every gate instance in the circuit. Every instance in the circuit has different load capacitance and since the delay is a function of the load capacitance, we need to design each gate independently to meet the delay requirement. A first approximation is derived from a lookup table and the fine tuning is done by a steepest descent method. We then described the entire methodology for automatic design of large circuits. The logic level power consumption results on the ISCAS’ 85 benchmark circuits showed an average power savings of up to 60%. We demonstrated the technique for larger circuits by designing the benchmark circuit c7552. The technique achieved average power savings of 58% and peak power savings of 68%. Thus, the technique is scalable for large circuits as well.

10.2 Future Work

The design methodology opens new research problems, some of which we list in this section.

10.2.1 Glitch Reduction Using a Statistical Delay Model

The delay of the gate is modeled as a single number in these calculations. But in reality, the gate delay is a distribution because of the various process variations, such as temperature, fabrication errors, etc. These variations force the delay to be a distribution, which is generally Gaussian. The optimization under the current method is not guaranteed due to these variations. The formulation has to be modified to account for these statistical distributions. Consider the timing window shown in Figure 10.1. The static delay for which the circuit is designed is
Figure 10.1: Hazard filtering condition for a statistical delay model. The old timing window is stretched to account for the delay distribution due to process variations. The delay varies as a Gaussian curve with a deviation $\sigma$ as shown. shown as $d_i$. Now due to the process variations the delay is actually a Gaussian distribution as shown in Figure 10.1. The technique assigns a single value for the gate delays and due to these variations, the gates in the circuit may not satisfy the hazard filtering condition and produce glitches. To account for this, the design procedure has to have the characteristics of the Gaussian distribution of the process parameters a-priori. The limits of this distribution can then be used during constraint generation. The hazard filtering condition can then be modified as:

$$d_i > (T_i + \sigma_i) - (t_i - \sigma_i)$$

where $\sigma_i$ is one half of the entire width of the Gaussian distribution of delay at gate $i$. This is an over compensated solution as it increases the critical path delay of the circuit. More research can be done on efficiently using a statistical delay model for the optimization procedure.
10.2.2 Leakage Current Reduction

The leakage current is a potential problem in future technologies. The variable input delay gate using a single nMOS transistor increases the leakage power in certain input configurations. This leakage power needs to be reduced by combining this technique with a leakage power reduction technique such as inserting sleep transistors. The sleep transistors are extra transistors in the direct current path that have a universal signal as their gate input. The gate input is disabled when the combinational block is unused, thereby reducing the leakage power. Other automated procedures such as LECTOR also can be used [69].

10.2.3 Design of a Low Power Standard Cell Library

The standard cell library is designed specific to every circuit. If a single library can be designed with enough configurations of cells such that it can be used for any circuit optimization, it would reduce the design time even further. We currently have 155 standard cells that are used for designing the benchmark circuits. These cells may be insufficient for designing other circuits. If an algorithm can be devised, which can link the cells in the library to the delay assignments, it can eliminate the step of creating new cells for every design.

10.2.4 Efficient Search Algorithms for Gate Sizing

The gate sizing algorithm described in the thesis contains a two stage process. The sizing, although faster than prior work, can be improved further to get a fast and accurate estimate of the transistor dimensions given the delay requirement and the load capacitance. If efficient models can be developed, the recursive simulation for fine tuning can be eliminated and the design time is lesser.
Figure 10.2: A new delay element combining the buffer and the pass transistor. This element does not suffer from the input slope degradation as much as a single pass transistor as the second inverter stage acts as a slope shaper circuit.

10.2.5 Variable Input Delay Logic Gate Modeling

Efficient models need to be designed for the variable input delay gate described in the thesis. We have used a basic procedure to calculate the $u_b$ of the technology based on the operating regions of the transistors. But the calculation of $u_b$ due to the signal integrity issues is also an open problem.

10.2.6 New Delay Elements

The main restriction on $u_b$ is the degradation of the slope of the input waveform and the limited regenerative capacity of the CMOS gates. This can be eliminated by designing a slope shaping circuit at the output of the pass transistor to improve the slope. Combinations of the delay elements discussed can also be experimented with. We give an example delay element that can be used in Figure 10.2. It shows a modified buffer with a pass transistor between the pair of inverters. This can be used more effectively as the second buffer can act as the shaping circuit and the regenerative property of the CMOS gate in the output can also be used.
References


Curriculum Vita

Tezaswi Raja

Aug, 1996- May, 2000  Bachelor of Engineering (B.E.)
Electrical and Electronics Engineering
Andhra University, Visakhapatnam, India

Aug, 2000- May, 2002  Master of Science (M.S.)
Electrical and Computer Engineering
Rutgers, The State University of New Jersey, USA.

May, 2002- May 2004  Doctor of Philosophy (PhD)
Electrical and Computer Engineering
Rutgers, The State University of New Jersey, USA.

Aug, 2000- Jan, 2001  Part-time Lecturer in the Mathematics Department of Rutgers University.

Jan, 2001- Aug, 2003  Graduate Assistant in Center for Advanced Information Processing (CAIP), Piscataway, NJ for conducting research in the field of VLSI Design.

Aug, 2003- Jan, 2004  Research Assistant at Wireless Networking Laboratory (WINLAB), Piscataway, NJ
for tool maintenance.


