Development of Algorithms for the Integration of Clock-Gating and Power-Gating Techniques

Master Thesis of:
Gaurang Upasani
Student ID: 07-908-055

Supervisor: Prof. Enrico Macii
Tutors: Prof. Alberto Macii
        Prof. Massimo Poncino
        Dr. Andrea Calimera

Academic year 2007–2009
Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Enrico Macii for his supervision and guidance throughout this work. The tasks provided by him for this thesis were motivating and gave me the opportunity to work in the cutting-edge research area of low-power design techniques. His excellently organized lecture notes provided simpler and more perspicuous explanation of fundamental concepts and also helped in correcting some confusions raised while referring to some published journal articles. Despite of being at a different location than of ALaRI and his busy schedule, his on-time communication through emails with necessary explanations and ideas were of a vital help.

I also sincerely thank all my tutors associated with this thesis work. I thank Prof. Massimo Poncino for sharing his ideas in the field of the work of this thesis. I thank Prof. Alberto Macii for valuable support and suggestions over the review of drafts of the submitted articles for publication and also for providing me with the necessary infrastructure in his laboratory of EDA research group located at Politecnico di Torino, Italy.

Special thanks goes to, Dr. Andrea Calimera, for continuously sharing the ideas and necessary discussions which revealed the solutions of the problems faced during the implementation of the prototype. His support gave me necessary push whenever there were drowning situations. I would also like to thank Dr. Leticia Bolzani for her advises and reviews of the ideas throughout the work.

I thank ALaRI for allowing me to undertake this thesis work following my expression of interest towards it and to execute some of the assignments at ALaRI laboratory. I also thank all my friends here at ALaRI and at EDA Group, Politecnico di Torino for their warm support.
To my parents and sister.
“Find a truly original idea. It is the only way I will ever distinguish myself. It is the only way I will ever matter.”

Jhon Nash Jr.
Abstract

Clock-gating and power-gating have proven to be two of the most effective techniques for reducing dynamic and leakage power, respectively, in VLSI CMOS circuits. Most commercial synthesis tools do support these techniques independently, but their combined implementation is not available, since some open issues in terms of power/timing overhead associated to the control logic required for the integration are not yet solved.

This thesis specifically targets the combined application of clock and power gating techniques. The work mainly focuses on proposing the algorithm which can check the feasibility of the application of this integrated approach by observing the savings achieved in power, area and timing overheads on the standard benchmark designs under test. In this work we also present a solution for reducing the timing overhead that may occur when the integration is performed.

In particular, we also compare the partitioning heuristics proposed here, so that to increase the efficiency of the clustering phase which is one of the key steps of our methodology considering the realtime applicability of the proposed synthesis flow. In the best of our knowledge, we also claim that this is the first attempt in which the two techniques are integrated to be compatible with the industrial design flow. And to support our ideology, all the experiments are performed on the industrial benchmarks, targeting the field of network on chip in particular, such as a network on chip switch and serial peripheral bus. The obtained results demonstrate the effectiveness of our solution; In fact, energy-delay product and timing overhead of the circuits synthesized using the proposed clustering algorithm improve by 33% and 24%, respectively.
## Contents

1 Introduction 19

1.1 Motivation 19

1.2 Moore’s Law 20

1.3 CMOS Scaling Theory 21

1.3.1 Constant Field Scaling 22

1.3.2 Constant Voltage Scaling 22

1.3.3 General Scaling 22

1.4 CMOS Power Dissipation Model 24

1.4.1 Dynamic Power 25

1.4.2 Static Power 27

1.4.3 A Designer’s Conclusion 30

1.5 Scope of this Thesis and Contributions 30

1.6 Composition of Thesis 31

1.7 Organization of Appendices 32

2 State of the Art : Low Power Methodologies 33

2.1 Clock Gating 34

2.1.1 Basic Principles of Clock Gating 34

2.1.2 Benefits of Clock Gating 36

2.1.3 Clock Gating Enhancements 37

2.1.4 Limitations 37

2.2 Power Gating and Sleep Transistor Insertion 38

2.2.1 Fundamentals of Power Gating 38

2.2.2 Implementation Challenges of Power Gating 40

2.2.3 Automated Sleep Transistor Insertion Flow 41

2.2.4 Improvements by Power Gating 42

3 Concurrent Clock and Power Gating 43

3.1 Feasibility Issues 44

3.1.1 Mutually Inclusive Nature of the Clusters 44

3.1.2 Timing and Energy Granularity during Clock and Power Gating 45

3.1.3 STI-Sleep Transistor Insertion and Physical Design Issues 46
3.2 Proposed Solution
   3.2.1 Analysis Phase
   3.2.2 Clustering Phase

4 Placement Aware Clustering
   4.1 Mathematical Modeling
      4.1.1 Defining the Cost-function
      4.1.2 Augmented Graph
   4.2 Introduction to Critical Interconnects

5 Clustering Heuristics
   5.1 Introduction of Graph Partitioning and Minimum k-cut Problem
   5.2 Partitioning Methodologies
   5.3 Greedy Heuristic
   5.4 Heuristic Based upon the Concept of Centroids
      5.4.1 Centroids Oriented Clustering
      5.4.2 K-mean Refinement

6 Experimental Setup and Results
   6.1 Experimental Setup
      6.1.1 Description of the Benchmarks
      6.1.2 Description of the Setup
   6.2 Results
      6.2.1 Results : Part 1 ”Net-criticality Matters!”
      6.2.2 Results : Part 2 ”Effects of Heuristics during Optimization.”

7 Conclusions and Future Developments
   7.1 Conclusions and Discussions
   7.2 Future Directions

A Leakage Power Components
   A.1 PN Junction Reverse-Bias Current
   A.2 Sub-Threshold Leakage
   A.3 Tunneling into and Through Gate Oxide and Hot Carrier Injection
      from Substrate to Gate Oxide
   A.4 Gate-Induced Drain Leakage - GIDL

B State of the Art : Leakage Power Reduction Mechanisms
   B.1 Design Time Techniques
      B.1.1 Dual $V_{th}$
      B.1.2 Multiple Supply Voltage
   B.2 Run Time Standby Leakage Reduction Techniques
B.2.1 Transistor Stacking ........................................ 106
B.2.2 Sleep Transistor Insertion ................................. 107
B.2.3 VTCMOS ....................................................... 107
B.3 Active Leakage Reduction .................................... 108
  B.3.1 Dynamic Voltage Scaling - DVS ......................... 108
  B.3.2 Dynamic $V_{th}$ Scaling - DVTS ...................... 109

C Proof of Minimum k-cut Theorem .............................. 111
  C.1 Gomory-Hu Tree ............................................ 111
  C.2 Minimum k-cut .............................................. 112

D User Guide to the Prototype CGPG Tool ...................... 115
  D.1 Using the CGPG tool ........................................ 115
  D.2 Compiling the CGPG tool ................................... 116
  D.3 Configuring the Tool ...................................... 116
  D.4 Tool Execution .............................................. 117

E Tools Used ......................................................... 119
  E.1 Synopsys Physical Compiler ................................. 119
    E.1.1 Commands for Gate Level Synthesis ...................... 119
    E.1.2 Commands for Physical Placement and Routing .......... 120
  E.2 Modelsim .................................................. 120
  E.3 OpenAccess API ............................................. 120

F Source Code ...................................................... 123
  F.1 Analysis Phase ............................................. 123
  F.2 Clustering Part ............................................ 125
List of Figures

1.1 Moore’s Law [40]. ................................................. 20
1.2 ITRS Roadmap. [35, 50] ........................................ 21
1.3 (a) Variation in Power Dissipation in Accordance with Moore’s Law.
(b) Power Density Compared with Scaling Factor [30] .................... 22
1.4 Scaling in INTEL Processors Following Moore’s Law [30] ............. 23
1.5 Power Consumption in Processors [14] ................................ 24
1.6 Dynamic Power Components in an Inverter [16] .......................... 26
1.7 Internal Switching Currents in an Inverter [16] ............................ 26
1.8 Leakage Power Contributors in an NMOS [28] .......................... 27
1.9 Leakage Power Components in an Inverter [62] .......................... 28
1.10 Comparison between Leakage and Active Power with respect to Tech-
nology Nodes. Predicted by ITRS [69] .................................. 28

2.1 Power Optimization Techniques for Different Stages of a Design Flow. [10] .................................................. 33
2.2 Conceptual Clock Gating. [54] .................................... 34
2.3 Register Level Clock Gating. [7] .................................... 35
2.4 Distribution of Temperature in POWER5 Architecture without Clock
Gating(left) and With Clock Gating(right) [46] ........................... 36
2.5 Variation of Leakage with Temperature [46] ............................. 36
2.6 Conceptual Power Gating. [54] .................................... 39
2.7 Conceptual Power Gating in a System on Chip [7] ......................... 39
2.8 General Flow of Sleep Transistor Insertion Technique. [54] ........... 41
2.9 Sleep Transistor Insertion Granularities (a) Block-based (b) Row-
based (c) Cell-based.[54] .................................................. 41
2.10 Improvements due to Power Gating. [54] ................................ 42

3.3 Row-based Clustering Methodology. [67] ................................ 48
3.4 Synthesis Flow ......................................................... 49
3.5 Actual Placement. [12] ................................................. 51
<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.6</td>
<td>Formation of Pure and Hybrid Clusters.</td>
<td>52</td>
</tr>
<tr>
<td>3.7</td>
<td>Cluster Generation and Bound Formation</td>
<td>55</td>
</tr>
<tr>
<td>3.8</td>
<td>Power Gating Granularities.</td>
<td>56</td>
</tr>
<tr>
<td>4.1</td>
<td>Conceptual Representation of Critical Interconnects.</td>
<td>63</td>
</tr>
<tr>
<td>5.1</td>
<td>Conceptual Graph Partitioning.</td>
<td>68</td>
</tr>
<tr>
<td>5.2</td>
<td>Minimum ( k )-cut Problem</td>
<td>69</td>
</tr>
<tr>
<td>5.3</td>
<td>Conceptual Partitioning with Greedy Heuristic.</td>
<td>73</td>
</tr>
<tr>
<td>5.4</td>
<td>Generation of Centroids.</td>
<td>74</td>
</tr>
<tr>
<td>5.5</td>
<td>Improvements in the Distortion Metric by ( k )-mean Refinement.</td>
<td>77</td>
</tr>
<tr>
<td>6.1</td>
<td>Experimental Flow of the Tool and Setup.</td>
<td>82</td>
</tr>
<tr>
<td>6.2</td>
<td>Modern Optimization Space for Sub-micron Technologies.</td>
<td>83</td>
</tr>
<tr>
<td>6.3</td>
<td>Effect of Placement Efforts on Various Cost-metric Parameters.</td>
<td>85</td>
</tr>
<tr>
<td>6.4</td>
<td>Comparison of Area and Power for Switch and SPI with Different</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>Combinations of Clock and Power Gating Techniques.</td>
<td></td>
</tr>
<tr>
<td>6.5</td>
<td>Plot of EDP to ( \alpha ) for the Switch 6x6 2.5</td>
<td>86</td>
</tr>
<tr>
<td>6.6</td>
<td>Plot of EDP to ( \alpha ) for the SPI</td>
<td>86</td>
</tr>
<tr>
<td>6.7</td>
<td>EDP and Propagation Delay for different values of ( \alpha ) for 7 Bounds of Switch</td>
<td>87</td>
</tr>
<tr>
<td>6.8</td>
<td>EDP and Propagation Delay for different values of ( \alpha ) for 6 Bounds of the SPI</td>
<td>87</td>
</tr>
<tr>
<td>6.9</td>
<td>Comparison of EDP using Greedy and Centroid Based Heuristics for</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>switch 6x6 2.5</td>
<td></td>
</tr>
<tr>
<td>6.10</td>
<td>Comparison of EDP using Greedy and Centroid Based Heuristics for</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>SPI</td>
<td></td>
</tr>
<tr>
<td>6.11</td>
<td>Comparison of EDP using Greedy and Centroid Based Heuristics,</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Considering the Best Partitions for Switch and SPI</td>
<td></td>
</tr>
<tr>
<td>A.1</td>
<td>Leakage Currents in Bulk CMOS [34]</td>
<td>97</td>
</tr>
<tr>
<td>A.2</td>
<td>Leakage Currents in a CMOS Inverter in ON and OFF State [54]</td>
<td>98</td>
</tr>
<tr>
<td>A.3</td>
<td>BTBT in Reverse Bias [60]</td>
<td>98</td>
</tr>
<tr>
<td>A.4</td>
<td>MOSFET current to Gate Voltage in Logarithmic (Left) and Normal</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td>Scales (Right) [17]</td>
<td></td>
</tr>
<tr>
<td>B.1</td>
<td>Conceptual Network with Critical and Non-critical Cells Replaced by</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Low and High ( V_{th} ) Cells</td>
<td></td>
</tr>
<tr>
<td>B.2</td>
<td>Dual ( V_{th} ) Domino Gate with Shaded Part as Low ( V_{th} ) [60]</td>
<td>104</td>
</tr>
<tr>
<td>B.3</td>
<td>Power Dissipation Ratio in a Configuration with 3 ( V_{DD} ) with ( V_{1} = 1.5V ) and ( V_{1} = 0.3V ) [17]</td>
<td>105</td>
</tr>
<tr>
<td>B.4</td>
<td>Stacking Effect in Two NMOS</td>
<td>106</td>
</tr>
</tbody>
</table>
B.5 VTCMOS 28 ................................. 107
B.6 Measured Leakage in Active and Stand-by Mode 17 .............................. 108
B.7 Total Power Compared with Clock Frequency in DVS and DVTS 60 109
List of Tables

1.1 Scaling of Different Parameters for MOSFET [48] .......................... 23
1.2 ITRS Prediction of Power Consumption for Different Technology Nodes [35] ...................................................... 24
1.3 Comparison of Different Leakage Reduction Techniques [17, 16, 60] .... 29

6.1 Explored Design Space Parameters .............................................. 84
6.2 Area, leakage and timing specifications for bounds = 7 with criticality model Benchmark Switch 6x6 2.6 ................................. 88
6.3 Area, leakage and timing specifications for bounds = 6 with criticality model Benchmark Spi_top ................................................. 89
6.4 Comparison of the Heuristics Used .............................................. 91
Chapter 1

Introduction

“The scientist is not a person who gives the right answers, he’s the one who asks the right questions.”
Claude Levi-Strauss

This chapter discusses in detail the motivation behind the trend to reduce dynamic and static power consumption in digital circuits. How the CMOS scaling pushes Moore’s law is discussed in sections 1.2 and 1.3. A general CMOS power dissipation model is discussed in section 1.4. In sections 1.6 and 1.7 the organization of thesis work and the appendices are discussed.

1.1 Motivation

With the invention of Integrated Circuit, an influential trend started; which is to accommodate as many component as possible in the same amount of silicon area. In a span of 4 decades the world has moved from Small Scale Integration to Very Large Scale Integration and all this is possible due to a streamlined semiconductor process technology which has been improving continuously over the last forty years [61]. Also the Computer Aided Design-CAD tools have made the automation of the design and verification of very complex ICs with billions of components on it possible. This enormous growth resulted in the birth of Electronic Design and Automation-EDA industries and they push the automation even further, Now there are tools available which can optimize the process starting from the designing until the final place and route, Also some tools designed to perform specific tasks such as clock tree extraction and optimization, net-list generation, overall power and timing optimization etc.

In the modern design of microelectronic system a dichotomy exist that they must be low power in concomitant with high performance. This dichotomy became popular because of the increasing popularity of the battery powered devices. [73, 14] The necessity is to increase the performance with extending the battery life. To
match with the performance criteria, CMOS technology scaling allowed the reduction of the gate delay and the increasing the operating frequency, the increase of the transistor density and, finally, the decrease of the energy per transistor \[33\]. Technology trends show that, in every circuit generation, delay and supply voltage have scaled down by 30\%, performance and transistor density have doubled every two years and, finally, transistor’s threshold voltage has reduced by almost 15\% \[28\]. In this scenario, power consumption represents one of the most important concerns in modern systems, since the battery power increases by about 15\% per year and the chip power requirements by about 35\% \[7\]. Reducing power dissipation is a design goal even for non-portable devices since excessive power dissipation results in increased packaging and cooling costs as well as potential reliability problems.

The stimulating force behind the advancement in the technology is the paper by the co-founder of Intel, Gordon Moore \[40\] in 1965, which was later modified by himself in 1975.

## 1.2 Moore’s Law

Gordon Moore predicted that *The number of transistors on chip doubles every two years*. This exponential increase in the number of transistors which can be integrated on a single die has been made true due to the advancements in the lithographic resolution and larger die sizes which is possible because advance manufacturing tools and innovative manufacturing techniques for the devices mainly transistors.

![Figure 1.1: Moore’s Law \[40\].](Image)

Figure 1.1 shows this exponential growth in processor transistor count from 1971-2008. This integration of transistors of this large scale is possible due to scaling
of device dimensions and other process parameters which is known as scaling theory. This scaling of CMOS trend is discussed in detail in section 1.3. One important note is that performance was not at all consider by Moore in his prediction, In fact performance is directly dependent on the number of transistors on the chip.

1.3 CMOS Scaling Theory

The basic idea of scaling is to reduce the dimensions of the MOS transistors and the wires connecting them in the integrated circuits [23, 44]. The goals behind doing this are to make things cheaper, to sell more functions (transistors) per chip for the same money, to build same products cheaper and to sell the same part for less money by reducing the price of a transistor simultaneously making it faster, smaller and power efficient.

Some of the important points of a case study [14] on the benefits of scaling the dimensions of a transistor by 30% can be enlisted as follows,

- Reduces gate delay by 30% (increase operating frequency by 43%).
- Doubles transistor density.
- Reduces energy per transition by 65%. Hence, 50% power savings at 43% increase in frequency.
- Die size used to increase by 14% per generation.

![Figure 1.2: ITRS Roadmap.](image)

The figure 1.2 shows the ITRS-International Technology Roadmap for Semiconductors prediction for the period of 1995-2020. As it can be seen from figure 1.2
that the leakage power (mainly sub-threshold leakage) will surpass the dynamic power of the chip as the major power component. In current submicron and deep submicron technologies leakage power is becoming dominant component of the total power consumption of the chip and it contributes to 50% of the total active power. Also for high-performance applications, effective oxide thickness of less than 1 nm with adequate reliability is needed. Introduction of higher dielectric constant (high-k) material in which tunneling current can be suppressed without sacrificing current drive, planar MOSFET requires high-channel doping to control short-channel effects, the trade-offs are mobility degradation and increased leakage power consumption[50].

1.3.1 Constant Field Scaling

As the name suggests, in this type of scaling the magnitude of internal electric field in the MOSFET is kept constant, while the dimensions are scaled down by a factor of $S$. This is an ideal methodology. Table 1.1 shows the scaling of different quantities for constant field scaling.

1.3.2 Constant Voltage Scaling

This is the most common model until recently. In this method of scaling the voltage is kept constant. As in full-scaling, scaling of voltages may not be very practical for many cases e.g., the peripheral and interface circuitry may require certain voltage levels for all input and output voltages. Also, there is a significant problem due to the restriction of decreasing the threshold voltage proportionately. Table 1.1 shows the scaling of different quantities for constant-voltage scaling.

1.3.3 General Scaling

It is the most realistic method for today's situation. In this scaling method voltages and dimensions scale with different factors such as $U$ as shown in table 1.1.
### 1.3 CMOS Scaling Theory

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Without Scaling</th>
<th>CFS</th>
<th>CVS</th>
<th>General Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel Length</td>
<td>L</td>
<td>(\frac{1}{2}L)</td>
<td>(\frac{1}{2}L)</td>
<td>(\frac{1}{2}L)</td>
</tr>
<tr>
<td>Channel Width</td>
<td>W</td>
<td>(\frac{1}{2}W)</td>
<td>(\frac{1}{2}W)</td>
<td>(\frac{1}{2}W)</td>
</tr>
<tr>
<td>Gate Oxide Thickness</td>
<td>(t_{ox})</td>
<td>(\frac{1}{2}t_{ox})</td>
<td>(\frac{1}{2}t_{ox})</td>
<td>(\frac{1}{2}t_{ox})</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>(V_{DD})</td>
<td>(\frac{1}{2}V_{DD})</td>
<td>(V_{DD})</td>
<td>(\frac{3}{4}V_{DD})</td>
</tr>
<tr>
<td>Threshold Voltage</td>
<td>(V_{TO})</td>
<td>(\frac{1}{2}V_{TO})</td>
<td>(V_{TO})</td>
<td>(\frac{1}{2}V_{TO})</td>
</tr>
<tr>
<td>Acceptor Doping</td>
<td>NA</td>
<td>sNA</td>
<td>(s^2NA)</td>
<td>(s^2NA)</td>
</tr>
<tr>
<td>Donor Doping</td>
<td>ND</td>
<td>sND</td>
<td>(s^2ND)</td>
<td>(\frac{s^2}{8}ND)</td>
</tr>
<tr>
<td>Oxide Capacitance</td>
<td>(C_{ox})</td>
<td>(sC_{ox})</td>
<td>(sC_{ox})</td>
<td>(sC_{ox})</td>
</tr>
<tr>
<td>Area</td>
<td>WL</td>
<td>(\frac{1}{2}WL)</td>
<td>(\frac{1}{2}WL)</td>
<td>(\frac{1}{2}WL)</td>
</tr>
<tr>
<td>Drain Current</td>
<td>(I_D)</td>
<td>(\frac{1}{2}I_D)</td>
<td>(sI_D)</td>
<td>(\frac{s}{8}I_D)</td>
</tr>
<tr>
<td>Power Dissipation</td>
<td>P</td>
<td>(\frac{1}{2}P)</td>
<td>sP</td>
<td>(\frac{s}{4}P)</td>
</tr>
<tr>
<td>Power Density</td>
<td>(\frac{P}{\text{Area}})</td>
<td>(\frac{P}{\text{Area}})</td>
<td>(s^3\frac{P}{\text{Area}})</td>
<td>(s^3\frac{P}{\text{Area}})</td>
</tr>
</tbody>
</table>

Table 1.1: Scaling of Different Parameters for MOSFET [48].

**Figure 1.4:** Scaling in INTEL Processors Following Moore’s Law [30]

Figures 1.3, 1.4 and 1.5 shows the implications of Moore’s law on the transistor density, power dissipation and power density in modern processors. Figure 1.5 explains the exponential growth in power dissipation for Intel processors for the the duration of 1971-2008 and it is expected to increase further.

Putting everything in a nut shell, due to technology scaling we are gaining more performance (also less area) but it tends to a higher leakage power dissipation. Reducing leakage power dissipation is having a performance penalty (also it consumes more area). Due to this trade-off, the performance and density of chips has a tendency of having a degraded slope for Moore’s law [72, 68]. The power density of
the highest performance chips has grown to the point where it is no longer possible to increase clock speed as technology shrinks. As a result, designers are designing multi-processor chips instead of chips with a single, ultra-high speed processor.

These problems are all expected to get worse as we move to the next technology nodes. The ITRS makes the predictions shown in table 1.2:

<table>
<thead>
<tr>
<th>Node</th>
<th>90nm</th>
<th>65nm</th>
<th>45nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>DynamicPower</td>
<td>1X</td>
<td>1.4X</td>
<td>2X</td>
</tr>
<tr>
<td>StaticPower</td>
<td>1X</td>
<td>2.5X</td>
<td>6.5X</td>
</tr>
<tr>
<td>TotalPower</td>
<td>1X</td>
<td>2X</td>
<td>5X</td>
</tr>
</tbody>
</table>

Table 1.2: ITRS Prediction of Power Consumption for Different Technology Nodes [35].

In the increase of chip power dissipation and power density, the operating temperatures play a vital role. Present day high performance chips work at very high temperatures thus increasing the cost of cooling system in place. This higher cooling costs has negated the cost of advantages achieved by higher integration levels due to scaling. This has inspired the designers to combat leakage power and to design power efficient chips.

### 1.4 CMOS Power Dissipation Model

In order to design circuits that consume as little power as possible, it is vital to understand the sources of power dissipation. In a CMOS circuit, power dissipation can be summarized by:

\[
P_{avg} = P_{switching} + P_{short} + P_{leakage} \tag{1.1}
\]
\[ P_{\text{avg}} = P_{\text{trans}} f_{\text{CLK}} C_L V_{DD}^2 + t_{SC} I_{\text{Peak}} V_{DD} f_{\text{CLK}} + I_{\text{Leakage}} V_{DD} \]  (1.2)

Where,
- \( P_{\text{trans}} \) switching activity factor,
- \( f_{\text{CLK}} \) clock frequency,
- \( C_L \) load capacitance,
- \( V_{DD} \) supply voltage,
- \( t_{SC} \) the time duration of the short circuit current,
- \( I_{\text{Peak}} \) the total internal switching current,
- \( I_{\text{Leakage}} \) leakage current.

In equation 1.2, the first two components are the dynamic power dissipation caused by switching activity at the various nodes within the circuits, while the third component is caused by static leakage. The following section examines these sources of power consumption in more detail.

1.4.1 Dynamic Power

The first and primary source of dynamic power consumption is switching power, i.e. the power required to charge and discharge the output capacitance on a gate. Figure 1.6 illustrates switching power. Note that switching power is not a function of transistor size, but rather a function of switching activity and load capacitance. Thus, it is data dependent.[17] In addition to switching power, internal power also contributes to dynamic power. Figure 1.7 shows internal switching currents also known as crowbar currents. Internal power consists of the short circuit currents that occur when both the NMOS and PMOS transistors are on, as well as the current required to charge the internal capacitance of the cell.

\[ P_{\text{dyn}} = P_{\text{trans}} f_{\text{CLK}} C_L V_{DD}^2 + t_{SC} I_{\text{Peak}} V_{DD} f_{\text{CLK}} \]  (1.3)

In equation 1.3, \( P_{\text{trans}} C_L \) can be merged to give the effective capacitance \( C_{\text{eff}} \) and

As long as the ramp time of the input signal is kept short, the short circuit current occurs for only a short time during each transition, and the overall dynamic power is dominated by the switching power. For this reason, designers often simplify the use the switching power formula.

There are a number of techniques at the logic design, architectural, and circuit design level of abstraction that can reduce the power for a particular function implemented in a given technology. These techniques focus on the voltage
and frequency components of the equation, as well as reducing the data-dependent switching activity\cite{5, 8}. Because of the quadratic dependence of power on voltage, decreasing the supply voltage is a highly leveraged way to reduce dynamic power. But because the speed of a gate decreases with decreases in supply voltage, this approach needs to be done carefully. SoC designers can take advantage of this approach in several ways:\cite{37}

1. For blocks that do not need to run particularly fast, such as peripherals, we can use a lower voltage supply than other, more speed-critical blocks. This approach is known as multi-voltage.

2. For processors, we can provide a variable supply voltage; during tasks that require peak performance, we can provide a high supply voltage and correspondingly high clock frequency. For tasks that require lower performance, we can provide a lower voltage and slower clock. This approach is known as voltage scaling. \cite{7}

3. Another approach for lowering dynamic power is clock gating. Reducing the operating frequency down to zero will also bring the dynamic power to zero. This is a very popular technique and some or the other form of clock gating is used on many SoC designs. \cite{4}
1.4 CMOS Power Dissipation Model

1.4.2 Static Power

This is the power consumption when the circuit is not operating. The figure ?? shows the leakage power dissipation in an inverter. Four main components of leakage currents which contributes to the leakage power in a CMOS gate are enlisted as follows[28],

- Sub-threshold Leakage ($I_{SUB}$): In the weak inversion region the current that flows from the drain to the source in a transistor is known as sub-threshold leakage.

- Gate Leakage ($I_{GATE}$): In a CMOS, due to gate oxide tunneling and hot carrier injection the current which flows directly from the gate through the oxide to the substrate which is known as gate leakage.

- Gate Induced Drain Leakage ($I_{GIDL}$): The current which flows from the drain to the substrate induced by a high field effect in the MOSFET drain caused by a high $V_{DG}$.

- Reverse Bias Junction Leakage ($I_{REV}$): The main cause of this leakage current component are the minority carrier drift and generation of E-H pairs in the depletion regions.

All these currents contribute to the static power in any transistor and for a better visualization they are shown in figure 1.8. A detailed explanation is also given in appendix A.

Figure 1.8: Leakage Power Contributors in an NMOS [28]

\[ I_{SUB} = \mu C_{ox} V_{th}^2 \frac{W}{L} e^{\frac{V_{GS} - V_T}{nV_{th}}} \]  

(1.4)

Where,

- $W$ and $L$ are the dimensions(width and length of the channel respectively) of the transistor.
28 1 Introduction

Figure 1.9: Leakage Power Components in an Inverter [62]

- $V_{th}$ is the thermal voltage $kT/q$ (25.9mV at room temperature).

- Parameter $n$ is a function of the device fabrication process and ranges from 1.0 to 2.5.

This equation 1.4 tells us that sub-threshold leakage depends exponentially on the difference between $V_{GS}$ and $V_T$. This explains that how the down scaling of $V_{DD}$ and $V_T$ to limit dynamic power, will make the leakage power worse exponentially. As explained above the gate oxide thickness ($t_{ox}$) is only a few atoms thick in 90nm gates, this is so thin that tunneling current can become substantial which adds to the gate leakage [62].

Figure 1.10: Comparison between Leakage and Active Power with respect to Technology Nodes. Predicted by ITRS [69]

In previous technology nodes, leakage current has been dominated by sub-threshold leakage. In reference to the figure 1.10 provided by the ITRS reports, starting with 90nm, gate leakage can be nearly 1/3 as much as sub-threshold leakage In 65nm it can equal sub-threshold leakage in some cases.

For future nodes 45nm and below, high-k dielectric materials will be required to keep gate leakage in check. This appears to be the only effective way of reducing gate leakage.
There are several approaches to minimizing leakage current. The detailed discussions on these techniques except power gating can be found in appendix [1].

- One technique is known as Multi-VT: using high $V_T$ cells wherever performance goals allow and low $V_T$ cells where necessary to meet timing[2].

- The other technique is to shut down the power supply to a block of logic when it is not active. This approach is known as power gating[60]. This is the main focus of the work done in this thesis and it is explained in chapter 2.

- Variable Threshold CMOS (VTCMOS)[18] is another very effective way of mitigating standby leakage power. By applying a reverse bias voltage to the substrate, it is possible to reduce the value of the term ($V_{GS} - V_T$), effectively increasing $V_T$. This approach can reduce the standby leakage by up to three orders of magnitude.

- The Stack Effect[32], or self reverse bias, can help to reduce sub-threshold leakage when more than one transistor in the stack is turned off. This is primarily because the small amount of sub-threshold leakage causes the intermediate nodes between the stacked transistors to float away from the power/ground rail. The reduced body-source potential results in a slightly negative gate-source drain voltage. Thus, it reduces the value of the term ($V_{GS} - V_T$), effectively increasing $V_T$ and reducing the sub-threshold leakage. The leakage of a two transistor stack has been observed and noticed to be an order of magnitude lesser than that of a single transistor[65]. This stacking effect makes the leakage of a logic gate highly dependant on its inputs.

The approaches briefed above can be compared on the basis of the savings in leakage and the area and timing overheads in each of them.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Leakage Reduction</th>
<th>Area Overhead</th>
<th>Timing Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Gating</td>
<td>67.58X</td>
<td>5-15%</td>
<td>5-10%</td>
</tr>
<tr>
<td>Stacking</td>
<td>1.18X</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>Dual $V_{dd}$</td>
<td>1.2X</td>
<td>&lt; 10%</td>
<td>0%</td>
</tr>
<tr>
<td>Dual $V_{th}$</td>
<td>1.91X</td>
<td>2%</td>
<td>0%</td>
</tr>
<tr>
<td>Body Biasing</td>
<td>2.19X</td>
<td>&lt; 10%</td>
<td>&lt; 0%</td>
</tr>
</tbody>
</table>

Table 1.3: Comparison of Different Leakage Reduction Techniques [17, 10, 60].

Table [1.3] provides a summery of the comparison and it can be seen that in terms of leakage savings, power gating out performs the other techniques. However, at the cost of affordable timing and area overhead. This is the main motivation factor behind the lots of research, in the direction of power gating techniques.
1.4.3 A Designer’s Conclusion

The most effective way to reduce dynamic power is to reduce the supply voltage. Over the last few decades, as the technology scaled to deep submicron dimensions, $V_{DD}$ has been lowered from 5V to 3.3V to 2.5V to 1.2V [33, 21]. The ITRS roadmap predicts that for 2008 and 2009 high performance devices will use 1.0V and low power devices will use even smaller 0.8V. Lowering the $V_{DD}$ actually reduces the $I_{DS}$, the driving current of the transistor, and as an implication to this the operating speed is reduced.

The above situation describes a conflict. To reduce dynamic power, $V_{DD}$ is brought down and to keep up with the performance, there is no alternative but to lower $V_T$ and this ends up in increased leakage current. Until now, this was a reasonable process, since static power from leakage current was so much lower than dynamic power. But as described above and shown in figure 1.10 scaling of technology below 90nm, will get to the point where static power can be as big a problem as dynamic power, and for future designs this conflict must be dealt with very carefully.

1.5 Scope of this Thesis and Contributions

The thesis work mainly limited to study the the dynamic power and static power reduction techniques specifically clock gating and power gating technique using the sleep transistor insertion method. Then the implementation of the proposed algorithm [12] for selectively merging clock gating and power gating techniques.

In [12], the authors present a layout-oriented synthesis flow which integrates the two techniques and that relies on leading-edge, commercial EDA tools. Starting from a gated-clock netlist, the circuit is partitioned in a number of clusters that are implicitly determined by the groups of cells that are clock-gated by the same clock-gating register. The main limitation of this approach is that the cost function that drives the clustering phase does not include placement information. In this way, the clustering is performed without any apriori placement information and the timing overhead that may result from the integration of the clock-gating and the power-gating techniques may grow too large.

Based upon this observation a concept of placement-aware clustering is introduced. The basic idea is to reduce the timing overhead seen in the model of [12]. To identify the potential bottlenecks of a such model, an exhaustive design space exploration of the cost-function proposed in [74] is carried out. In this way, it is possible to understand the relationship that may exist between the selected variables as a function of the energy-delay product. Based on the results of the exploration, a novel concept of critical-interconnects is introduced and exploited for reducing the
timing overhead of the integrated CG/PG designs.

Once the placement aware cost function is determined the next step is to make the prototype of the proposed synthesis flow more realistic by introducing several partitioning techniques. As proposed in [12] a greedy heuristic was to be used for the partitioning of the space values generated by the cost-function but it takes considerable amount of time and it may also possible that the partitions are not optimal. For these reasons, several partitioning techniques are implemented and tested as a part of this thesis.

1.6 Composition of Thesis

The thesis work is composed as follows,

Chapter 2 studies the state of the art discussing the low power methodologies in detail keeping the industry standards and applications into the prime focus. It also discusses the limitation, benefits and the various enhancements done in the traditional methods over the past few years. The chapter founds a base for the rest of the work done in this thesis.

Chapter 3 introduces the simultaneous application of clock gating and power gating techniques. It checks the feasibility of adapting such technique along with the challenges to be met for the real-time applications. It also discusses the proposed synthesis flow as a part of the solution proposed in this thesis.

Chapter 4 provides a detailed idea on the formulation of the problem undertaken as a part of this thesis work. It describes the mathematical model required for the implementation of the real-time applicable prototype. It also introduces a novel concept of critical interconnects for the improvement of the existing model.

Chapter 5 quickly reviews the necessary concepts required from traditional graph theory and then discusses in detail with the problem of graph partitioning. It also presents several heuristics implemented as a part of this thesis work and adapted from the open source tool.

Chapter 6 provides necessary guidelines required for the experimental set up. It also contains the obtained results by application of this synthesis flow on standard benchmarks. A detailed discussions of the obtained results close the chapter.

In chapter 7 an over all conclusion is provided along with the necessary directions and scopes for the future work in the same area.
1.7 Organization of Appendices

The appendices contain the necessary theories and explanations used in the work of this thesis in detail. The organization of these appendices is as follows,

Appendix A explains the leakage current mechanisms and power dissipation mechanisms in a CMOS in detail.

Appendix B discusses various methods adapted for leakage power reduction by the designers for deep submicron technology.

Appendix C provides the necessary proof required for the feasibility of problem of minimum k-cut theorem applied in this synthesis flow.

Appendix D describes the implementation of the synthesis flow and explains the user commands for the synthesis and the necessary commands to be set.

Appendix E briefly overviews the commands used by Synopsys synthesis and placement tools.

Appendix F presents the source code of the case study and commented.
Chapter 2

State of the Art: Low Power Methodologies

“Everything should be made as simple as possible, but not simpler.”
Albert Einstein

This chapter gives an introduction of the clock gating and power gating techniques. In section 2.1 basic clock gating principles, benefits, limitations and the enhancements in the traditional clock gating are discussed. In section 2.2 principles of the power gating, challenges in sleep transistor insertion flow and the limitations are discussed in detail.

Figure 2.1: Power Optimization Techniques for Different Stages of a Design Flow.

Figure 2.1 shows the design space of possible techniques available for power optimization from top to bottom and what are the impacts of the applications of these techniques on static or dynamic power from left to right.
2.1 Clock Gating

2.1.1 Basic Principles of Clock Gating

Clock signal is the most fundamental signal for any digital circuit. In earlier days it was considered that clock signal should be kept as clean as possible and no designer should manipulate clock signal. Very recently it was realize that clock signal consumes the most amount of the total power due to the signal it self and the unnecessary activities created due to clock in underlying logic. Up to 70% or even more of the dynamic power can be spent in the clock buffers\[9\]. This result makes intuitive sense since these buffers have the highest toggle rate in the system, there are lots of them, and they often have a high drive strength to minimize clock delay. In addition, the registers receiving the clock dissipate some dynamic power even if there is absolutely no change in input and output.

Then after extensive efforts were made\[9, 5, 8, 4\] to reduce the dynamic power by gating the clock signal and disabling some of the portions of the logic under specific idle conditions, known apriori. As described in \[5\] most of the functional units are not used in any microprocessor for almost 50% of the time. If the unused period is significant enough then it is advisable that such block should be turned off to save power.

Clock gating is particularly useful for registers that need to maintain the same logic values over many clock cycles. The main challenges of clock gating are finding the best places to use it and creating the logic to shut off and turn on the clock at the proper times. One of the possible configuration is drawn in the figure 2.2.

![Figure 2.2: Conceptual Clock Gating.](image)
2.1 Clock Gating

shown in the figure 2.3.

Figure 2.3: Register Level Clock Gating. [7]

In the figure 2.3 it is observed that in original RTL, the register is updated independent of the enable pin (EN). Without altering the functionality of the original circuit similar results can be obtained by gating the clock gating in the same RTL. The amount of the achieved saving is directly proportional to the size of the registers. The larger the size of the registers the larger the savings.

In the early days of RTL design, engineers would code clock gating circuits explicitly in the RTL. [7] This approach is error prone in the sense that it is very easy to create a clock gating circuit that glitches during gating, producing functional errors. Today, most libraries include specific clock gating cells that are recognized by the synthesis tool. The combination of explicit clock gating cells and automatic insertion makes clock gating a simple and reliable way of reducing power. No change to the RTL is required to implement this style of clock gating.

During clock-gating, while evaluating clock network power, four contributions are considered:
The input capacitances of the module and of the AND gate, the capacitance switched by the interconnection in the clock tree and by the interconnection that feeds the control signal to the gating logic.

As suggested in [34], Power dissipation can be modeled considering the above four components as, \( c_0 \) be the unit wire capacitance, \( l_i, l_g \) the interconnection length of the clock tree and of the control gating logic signal, respectively, \( C_i \) and \( C_g \) the
input capacitance for the module and the gating logic then,

\[ 2(c_0l_i + C_i)p(i) + (c_0l_g + C_g)p_{tr} \]  \hspace{1cm} (2.1)

Where,

- \( p(i) \) is the probability for the module to be active.

- \( p_{tr} \) is the probability to have a transition at the net. Which is defined as the ratio between the number of transitions in the activation function evaluated over consecutive clock cycles.

### 2.1.2 Benefits of Clock Gating

In the todays high frequency microprocessors, clock gating is adapted as the 70% of the total switching power is consumed in the clock circuitry. The major part of this clock power is consumed at the leaf nodes of the clock tree. Gating the clock at the last few levels mainly leaf nodes which are the driving latches of clock tree is an effective way considering the penalty costs.

![Figure 2.4: Distribution of Temperature in POWER5 Architecture without Clock Gating(left) and With Clock Gating(right) [46]](image)

![Figure 2.5: Variation of Leakage with Temperature [46]](image)
In addition to reducing the dynamic power, clock gating can also reduce the static power. Leakage through CMOS devices exponentially dependent on the temperature. The figure 2.5 indicates the relation of leakage and the temperature. Hence, the temperature reduction achieved by clock gating provides significant reduction in leakage power. Figure 2.4 shows different temperature plots through an infra-red sensor in different regions in POWER5 chip. The temperature drop is of over 10 degrees in the hottest regions. And it has been analyzed that for a uniform 10 degrees of reduction in temperature would give approximately 10% of reduction in leakage.

2.1.3 Clock Gating Enhancements

Clock gating techniques can be applicable to any level of granularity and different methods are effective at different levels of applications. And there is always a trade-off between the amount of power saving achieved compared to the area overhead and design complexity. And based upon this different enhancements are developed to improve the power savings over the traditional clock gating.

Activity driven clock gating\[1\] takes the activity of the registers into account and the registers with higher activities should not be clock gated. In this method the flip flops are sorted in the increasing order of switching activities and the short-listed flip flops are clock gated.

In Observability Don’t Care (ODC) driven clock gating\[36\] the registers which are not observable during the given clock cycle should not be gated. In this method a boolean expression of the ODC variable has to be pre-computed to properly assign the clock gating conditions in the next clock cycle.

The register level clock gating technique takes the advantage of the division of data into multiple stages of pipeline. It will reduce the power by 44% over traditional clock gating\[7\]. A much advanced version of this method uses master slave flip flops and it is based on the concept of elastic pipeline. It shows 5% of improvement over the traditional clock gating.

2.1.4 Limitations

The major limitations of the clock gating techniques are enlisted as below,

1. The main limitation is the timing of the clock signal and the ability to group latches with identical gating conditions. Sometimes the grouped latches are too small to be clock gated considering the penalties with compared to the
amount of power saving achieved. Also with the increasing wire delays the, 
placement and routing of the latches close to the cone of the logic may conflict 
with the placement necessary to group a set of latches needed for clock gating.

2. Sometimes it is difficult to reach the timing closure if the clock gating signal 
have larger fan out and it is driving many clocks drivers if the latch group is 
very large.

3. Another problem for deep-submicron technology is the inductive noise due on 
supply voltage rails. To nullify this effects sometimes designers use on chip 
decoupling capacitors which can increase the leakage power significantly.

4. In traditional clock gating as shown in figure 2.2 it does not take into account 
the switching activities of the registers it involves.

5. It also does not consider the possibility of one part of the functional unit is in 
use while the other is not in use.

6. Clock gating reduces test-coverage of the circuit because clock-gated registers 
are not clocked unless the enable signal is high.

Automatic insertion of power-management logic provides sizable power sav-
ings. But it is unrealistic considering the area overhead. Clock tree planning also 
helps to improve the application of clock gating[36].

2.2 Power Gating and Sleep Transistor Insertion

2.2.1 Fundamentals of Power Gating

Leakage power dissipation grows with every generation of CMOS process tech-
nology. This leakage power is not only a serious challenge to battery powered or 
portable products. To reduce the overall leakage power of the chip, it is highly desir-
able to add mechanisms to turn off blocks that are not being used. This technique 
is known as power gating[2, 3, 77].

One of the beneficial technique is to provide an external power switch to the 
logic. And turn off this switch when the logic is not in function. This will end up 
in almost zero leakage power but this method is costly and unpractical in terms of 
timing and reactivation costs. Hence, the idea of an internal switch was introduce 
and this switch is known as sleep transistor[60, 22].

The figure 2.6 shows a conceptual implementation of power gating. The cluster 
refers to the group of standard cell, and the power switches sleep are connected to
VDD and VSS.

The figure 2.7 shows a conceptual SoC with internal power gating mechanism. As it is clear from the figure that the power gated functional block receives power from the power gating controller and power gating switching network. This network connects the power gated block to ground or supply rail. Generally, this power gating switching network occupies sleep transistors with higher $V_{th}$ and thicker $t_{ox}$. Power gating controller enables these sleep transistors.

The important thing to notice in figure 2.7 is that with power gating it is possible that outputs of the power gated block may stabilized very slowly. And as an implication of this, these outputs spend a significant amount of time at threshold voltage, causing large crowbar currents in non-power gated blocks. To prevent issue, isolation cells the $isol$ block in the figure 2.7 are placed between the outputs of the power gated block and the inputs of the non-power gated block. These isolation cells are designed so that they do not experience crowbar current when one of the inputs is at threshold, as long as the control input is off. The power gating controller
provides this isolation control signal.

### 2.2.2 Implementation Challenges of Power Gating

While implementing the power gating techniques the designer has to face certain challenges\cite{20, 17, 62, 54}, as follows,

1. Specification of the power switching fabric. In other words the specific size of the sleep transistors.

2. Identification of the shut down events. This is a crucial task as sometimes it is not desirable to turn off certain part of the logic depending on the total time of sleep mode. If the sleep mode is small enough then the savings achieved will be nullify by the reactivation energy and timing cost for that functional block. This decision is taken by the control module.

3. Proper selection of the isolation registers and the retention registers.

4. Specifying the correct constraints.

5. Performing state dependent verification for each active state.

6. Synchronization of the clocks and reset.

7. Interfacing issues between sleep regions and logic blocks without gated ground. And between sleep regions controlled by different activation signals.

There are some challenges during the process of sleep transistor selection and insertion\cite{58, 22},

1. The foremost architectural challenge is weather to switch VDD using header, \textit{High $V_T$ pMOS} switch or to switch VSS using footer, \textit{High $V_T$ nMOS} switch or to switch both. The solution of this is totally application dependent. An nMOS sleep transistor usually produces higher switch efficiency and hence smaller total transistor size than its pMOS counterpart.

2. Sleep transistors must be compliant with the standard cell library.

3. nMOS has smaller width (for the same performance) pMOS has smaller gate leakage, pMOS can provide $V_{DD}$ isolation. While IP integration and system level power is the prime concern a header switch is the correct choice and by applying reverse body bias the switch efficiency can be increased and area cost can be reduced.

4. High-$V_T$ has less leakage, Low-$V_T$ has smaller width (for the same performance) and larger size preferred for performance smaller size preferred for leakage.
2.2 Power Gating and Sleep Transistor Insertion

5. Selection of gates to which STI should be applied requires layout information.

2.2.3 Automated Sleep Transistor Insertion Flow

![Diagram of Automated Sleep Transistor Insertion Flow]

An automated STI flow is shown in the figure 2.8. It starts from the standard cell placed design. The next step is clustering in which proper cells are grouped together and then depending on the peak current estimation of the cluster the dimension of the sleep transistor is decided. Based upon this the layout is modified and validated.

![Diagram of Sleep Transistor Insertion Granularities]

Figure 2.9: Sleep Transistor Insertion Granularities (a) Block-based (b) Row-based (c) Cell-based.
Depending on the cluster granularity different type of the sleep transistor insertion techniques are occupied.

- Block-based (distributed). Sleep transistor cells placed at the boundaries of each row. All cells in the block are power-gated and controlled by the same sleep signal. Shown in figure 2.9(a). [62]

- Row-based (coarse-grained clustering). As displayed in figure 2.9(b) in row based STI, sleep transistor cells placed at the boundaries of selected rows. Clusters consist of sets of rows to be power-gated. All cells in rows of the clusters are power-gated. [67]

- Cell-based (fine-grained clustering). Sleep transistor cells placed at the boundaries of selected rows. Clusters may contain incomplete rows. Each sleep transistor controls all the cells in a cluster. This method is shown in figure 2.9(c). [75]

2.2.4 Improvements by Power Gating

This automated STI flow reduces the leakage upto 80%.

![Bar chart showing improvements due to Power Gating](image)

Figure 2.10: Improvements due to Power Gating. [54]

The figure 2.10 shows the comparison between area overhead and the leakage savings. The results are clear that the additional area cost due to insertion of the sleep transistors might be neglected compared to the amount of leakage savings achieved by implementing power gating in the given design.
Chapter 3

Concurrent Clock and Power Gating

“A new scientific truth does not triumph by convincing opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it.”

Max Planck

This chapter discusses the concept of the simultaneous application of clock-gating and power-gating techniques. Section 3.1 dictates the feasibility issues related to design, challenges while integration of clock-gating and power-gating and some physical design issues related to placement and routing considering the real-time applications. The section 3.2 describes the proposed synthesis methodology and the design flow of the prototype tool developed under the work of this thesis.

In a broader sense, clock-gating and power-gating are the techniques to selectively turn-off a part of the digital circuit. From the discussion in chapter 2 clock-gating and power-gating may look as unrelated techniques. A careful observation reveals that both the techniques are actually two different ways of exploiting the same property of any design, called idleness. However this idleness is characterized very differently in both the techniques. In clock-gating on the occurrence of the idle conditions the clock signal is stopped while in power-gating on the occurrence of such situation the part of the logic block is isolated from the ground lines. Clock-gating extracts the idle conditions on a cycle-by-cycle bases and stops the clock signal for that period while power-gating is activated by an external signal. Integrating clock-gating and power-gating techniques means to adapt the conditions used for clock-gating, to power-gate the circuit so that both dynamic and static power is saved during the idle situations.
Figure 3.1: Conceptual Integration of Clock and Power Gating. [11]

Figure 3.1 shows the conceptual integration of clock-gating and power-gating methodologies. The clock-gating conditions are logically OR-ed with the external activation signal for power-gating and it is provided to the sleep transistor.

3.1 Feasibility Issues

Practically, as discussed in [12] the integration of the two techniques has to meet a lot of challenges related to the behavior of the circuit. All the problems discussed in the coming section requires the exploration of the power-saving to performance tradeoffs by the designer to check the possibility of the integration of the two techniques.

3.1.1 Mutually Inclusive Nature of the Clusters

While applying the clock-gating conditions on a circuit, the circuit is divided into clusters based on the idle conditions. Dedicated clock-gating registers are allocated to each of the clusters keeping in mind the real-time switching activities at each and every net of the cells in the respective clusters. However the intersection of these clusters in terms of the overall functionality is not null. This means the clusters are not mutually exclusive and some logic cells are shared among the clusters. This implies that the intersected part must be treated carefully while applying the power-gating conditions because electrically isolating such intersected cells may result in a traumatic circuit behavior.

The solution for this problem is to generate a new cluster as shown in figure 3.2 (b) that includes the logic cells, belonging to the intersection of two or more clusters.

If all the cells of cluster1 are power-gated and controlled by Fa1, when cluster1 is in the idle state (i.e. Fa1 = 0), all its cells are isolated from ground and they can’t provide a correct output voltage; this automatically implies that cluster2,
3.1 Feasibility Issues

which is in the active state (i.e. $Fa_2 = 1$), cannot work properly (metastability in cluster 2 due to power-gating of cluster 1). A dedicated control signal ($Fa_1$ and $Fa_2$) guarantees the functionality of both clusters: cells belonging to the intersection are disconnected from ground if and only if both clusters are in the idle state. The size of the overlapping region is a good measure to determine the leakage power saving compared to the amount of overhead in terms of area and propagation delay.

In the context discussed above the term cluster means the grouping of logic cells. It is worth noticing that here the initial clusters are determined by the clock-gating implicitly. In other words a group of logic cells that is clock-gated by the same clock-gating register will form a cluster. There is also a possibility that a number of clock gating registers are triggering the same enable net in this case all the cells with the same enable net should also be included in the same cluster.

3.1.2 Timing and Energy Granularity during Clock and Power Gat- ing

This is a crucial design issue while integrating clock-gating and power-gating techniques. Time granularity for clock-gating can be defined in terms of number of clock-cycles in other words minimum number of clock-cycles for which a cluster can be clock-gated that is one clock cycle. And for power gating it is the so called wake-up time, or wake-up latency. The wake-up time is the time required by the power gated cluster to be fully functional once again, and thus the time that one must wait to consider the power-gated cluster reactivated, in order to guarantee the

---

1 Clock-gating clusters are the initial clusters generated by merging the group of standard cells, whose idleness is characterized by the same clock gating register or a group of cell whose idleness is characterized by the intersection of two or more clock-gating registers. The figure 3.6 shows these initial clusters.

2 Power-gated clusters are the clusters formed by selectively merging the initial, clock-gated clusters. These two terminologies should not be confused with each other. In the filed of placement and routing the term clustering unanimously means the application of power gating and hence, the generation of bounds(power-gated clusters).
full functionality of the system.

Since the granularity of clock-gating is 1 clock-period, it could be quite possible, that a cluster can also be gated for only 1 clock period and to apply the power gating in this situation the reactivation-time must be lesser than 50% of the clock-period. If this does not happen, then the block may not be fully reactivated at the edge of the next active clock cycle; then, the cluster may not be able to perform its work by the time deadline. This may cause latching error, metastability issues and, even worse, system failure.

Moreover, in clock-gating the reactivation delay is zero. While power-gating has the reactivation energy costs and it can be applied only when the duration of the sleep mode is sufficiently longer than the reactivation time of the sleep transistors. The estimation of the reactivation energy every time the cluster is turned-on, a simplified model is adopted. This model describes the power-gated clusters as a capacitance during the transient phase of reactivation. This is also known as the virtual ground capacitance denoted by $C_{vgnd}$. Since at the end of the stand-by period the $C_{vgnd}$ is charged up to $V_{dd}$, the turning-on of the cluster implies the discharging of such a capacitance. This can be modeled as $E_{original} = 0.5 \cdot C_{vgnd} \cdot V_{dd}^2$. Where, $C_{vgnd} = \sum C_i$. $C_i$ is the virtual ground capacitance of each standard cell under that bound. Frequent switching between sleep mode and activation mode because of the short sleep duration will offset the resulting leakage saving.

Once the amount of total leakage savings during the idle period, $E_{Leak}$, and the total reactivation energy overhead, $E_{original}$, is known it is very easy to compare the two in order to estimate the efficiency, as well as the convenience of the combined application of CG and PG. If the total energy savings obtained during the idle period is larger than the one spent to reactivate the cluster, then CG+PG can be considered as feasible; otherwise, losses overcome gains, therefore joint application of CG and PG is not feasible.

For power-gating, a good practice is to determine the amount of power savings by sleep mode compared to the reactivation costs. This implies that for optimum savings designers can omit to power-gate some clusters, as not all clock gating conditions are suitable for power gating.

### 3.1.3 STI-Sleep Transistor Insertion and Physical Design Issues

In power-gating a sleep-transistor is inserted in between the cluster of logic and the corresponding virtual ground. All the clusters who qualified to be power-gated should be assigned an independent sleep-transistor, whose size and virtual
ground voltage are determined by the amount of current drawn by the respective cluster. This makes it necessary that all the cells belonging to the same cluster, are to be placed in a close physical proximity during layout. Violating this, will result in scattering of the cells of a cluster all over the layout area and it will make the connections to the sleep-transistors impractical. Ideally it is also possible to dedicate a sleep-transistor per cluster but this will force the placement-tool to allocate independent regions for each cluster and this will result in higher performance penalty because of the interconnect delay among them.

The fact is that standard methods of placement are not able to manage the integrated clock and power gating because they place cells belonging to different clusters in the same row. This implies a larger number of $V_{\text{gnd}}$ metal wires for each row (i.e large layout disruption and area penalty). One solution is to force the tool to place cells of to the same cluster in dedicated regions called bounds (dedicated number of rows). Now the problem is that the number of clusters could be very large (e.g. 18) and a solution like that [12] is still too costly from a physical place and route aspect (large layout disruption results in huge timing overhead). [13] proposes to merge the clusters in an intelligent way. This helps to reduce the layout modification, thus achieving better timing results at the cost of little penalty in energy saving.

For this work, row-based sleep-transistor insertion methodology is adapted [67]. In this methodology the regions are the rows, which will be further referred as "bounds" in the further chapters. In Each bound will be allocated its own sleep-transistor and a virtual ground rail.

This methodology proves very efficient in case of multiple-$V_{\text{gnd}}$ based clustering proposed in [3]. This is due to the fact that in multiple-$V_{\text{gnd}}$ based clustering, since one has to route more than one virtual ground line, the routing overhead and congestion is higher than in single virtual ground based methods and hence has the potential to overwhelm the advantages of this methodology.

Figure 3.3 shows a simple conceptual standard cell layout based upon row-based power gating methodology. Here the layout shows distribution of cells of 3 clusters in 8 rows of design. As the figure shows a dedicated row allocated to the sleep-transistors which is also known as sleep cavity. The figure does not include the global power supply lines connecting the local horizontal supply lines (VDD, GND, VGND1, VGND2 and VGND3) for better readability.

Row-based sleep transistor insertion has several advantages.
Figure 3.3: Row-based Clustering Methodology. [67]
3.2 Proposed Solution

1. Row-based power gating methodology permits us to allocate multiple ground lines with minimum changes in the already placed design.

2. It enables the application of power-gating to non-structured logic, i.e. when it is not possible to completely isolate at the physical level a large, monolithic logic block to be power gated.

3. It greatly enhances the opportunities for trading off area and speed overheads for leakage power reduction, as it makes it possible to fine tune both row selection and sleep transistor sizing.

The main implication of the issues discussed above is that to reduce the possible area and timing overheads physical design constraints limit the number of bounds that can be used, therefore multiple clusters are merged and assigned to a given bound sharing a single sleep transistor and virtual ground rail.

3.2 Proposed Solution

![Synthesis Flow Diagram]

Figure 3.4: Synthesis Flow.

Figure [3.4] shows the synthesis flow proposed [12] as a solution for the problems discussed in the integration of clock-gating and power-gating techniques. The flow consists of 5 steps starting from the synthesis of an RTL net-list to the sleep transition insertion.
• Step: 1 *(Synthesis)*:

This step performs the synthesis of the RTL design. *Synopsys Physical Compiler* is used to perform the synthesis of the HDL net-list according to a set of constraints and synthesis directives such as enabling the clock-gating feature etc. The process converts the HDL net-list into the gate level design using standard cell libraries of 65 nm technology provided by STMicroelectronics. The outputs of this steps are

- Clock-gated, gate-level description of the design.
- Power, timing and area reports.
- The place and route information (.DEF), gate level net-list (.DDC) and internal switching activities (.SAIF).
- Information regarding the parasitic parameters (capacitance etc.) (.SPEF)

Once the new gate-level description is generated, it is used to generate the real-time switching activities at each and every net of the design. This is done by the simulation of test-benches for the benchmarks in Modelsim and the information is stored in a .VCD file.

• Step: 2 *(OA Database)*:

During this step an OpenAccess database of the gate-level design is generated. Further information regarding this step is provided in appendix [E].

• Step: 3 *(Clustering)*:

This is the core of the whole design flow. In this step the feasibility of integration of two methodology is being evaluated using the real-time activities at every net and the OpenAccess database of step 2. For the better understanding of the functionality, this step is divided into two phases namely *Analysis Phase* and *Clustering Phase*. The subsequent sections 3.2.1 and 3.2.2 will discuss these two phases in detail.

• Step: 4 *(Placement)* During this step the physical placement of the gate-level design generated in step 1 is carried out following the placement constraints resulting from the generation of the power-gated clusters.

Figure [3.5](3.5) shows the possible physical layout. It is important to observe that the power-gated clusters "bounds" can contain different number of cells and hence they will be placed in different areas. This implies that, if the area required to place the power-gated cluster is larger than the area dedicated for a row of the layout then the cluster will not be power-gated considering the overhead in terms of dynamic power and the length of interconnects resulting
3.2 Proposed Solution

because of the placements of the cells belonging to the same cluster in different rows.
It is important to find the trade-off between the proposed leakage energy saving and the overheads due to placement.

- Step: 5 (Sleep Transistor Insertion) The last step is to perform the STI. In this step the size of the required sleep-transistor is defined based upon the respective power-gated cluster dimensions. Sleep transistors are inserted into a dedicated row named sleep device cavity. The number of virtual grounds is associated to the number of power-gated clusters present in a given row.

3.2.1 Analysis Phase

During analysis phase the information of the synthesized RTL(with clock-gating feature enabled) with the constraints, the place and route information (.DEF), the parasites (.SPEF) and the net-lists, along with the realtime activities on each net of the design (.VCD), the OpenAccess database of the design and the standard cell libraries of 65nm technology are provided as the inputs.

The analysis phase generates the clusters who qualify for power-gating and categories them as "clock-gated clusters" and "clock-gated intersection clusters".

- Clock-gated clusters consist of the cells belonging to only single cluster, in other words they are the cells whose idleness is characterized by only one clock-gating register, let’s call these clusters ”pure-clusters”.

- Clock-gated intersection clusters consist of the cells which belong to two or more clusters, in other words they are the cells whose idleness is determine by
more than one clock-gating register, let’s call such clusters “hybrid-clusters”.

Referring to figure 3.6, cluster 1 and cluster 2 are the pure clusters and cluster 1/2 is a hybrid cluster whose activation function is the logical AND of cluster 1 and cluster 2. This explains that cluster 1/2 can be turned off only when both cluster 1 and cluster 2 are turned off at the same instance of time.

The Algorithm for Analysis Phase

The functionality of this phase can be described as follows. It starts with the identification of the clock net and propagates through the entire design tracing the clock signal. Every time upon the encounter with the cell it performs several procedures enlisted below.

1. If the cell is not a register, the tool launches a recursive function until a register is found.

2. If the cell is a register the recursion function is terminated. Then, if the register is a regular (non-gated) register, the tool simply writes its name in the output file.

3. If the cell is a CG register, the cell is printed on the output file and marked. The marking consists of a CG register property which is attached to the register, thus allowing a quick identification of the CG registers. The second task of
the Step 2 identifies the activation functions associated to the CG registers. The following operations are executed from the CG register:

- It reads all the cells and their properties in order to identify the CG registers. Then, a backward trace analysis is performed starting from the enable pin belonging to respective the register.

- It identifies the net associated to the enable pin and visits the tree. During this step, the tool performs the following controls on the cell: Once it has been examined, the cell is marked with the name of the net that is the input of the CG register, through the enable pin. In this way, cells remain associated to the activation function they belong to. Encountering a register will terminate any further exploitation of the branch of the clock tree.

4. The last phase of Step 2 identifies the cones of logic that fan out of each CG register. During this phase the tool counts the total number of cells and performs the calculation of the power costs and savings. The analysis starts from the CG registers, their outputs are selected and a recursive visit of the tree of logic connected to them is launched.

5. The clock-gated and clock-gated intersection clusters are generated based upon the information of the clock-tree and fanout of each cell.

The above algorithm can be outlined in a pseudocode as indicated in algorithm 1.

This phase of the clustering step estimates the area required, leakage power, parasitic capacitance and total idle period for every cluster. It also generates the information regarding to the clock tree, the activation function associated to every clock-gated and clock-gated intersection clusters and the fanout of all cells. It also generates the pre-placement information including the co-ordinates of the placed clusters with the row numbers.

### 3.2.2 Clustering Phase

During clustering phase the information of the activation function and the information of pure and hybrid clusters generated by the analysis phase are given as the inputs and based on these information a cost-function is being evaluated to generate a matrix. And this matrix is partitioned forming the bounds, which are to be placed. A detailed discussion about the selection of cost-function with its design space exploration is done in the chapter 4.

Figure 3.7(a) shows the conceptual distribution of the cells in clusters after the analysis phase. As it can be seen that cluster 1, 2, 3, 4 and cluster 5 are the pure clusters while cluster 6 is a hybrid cluster. Cluster 6 can be turned off only when
Algorithm 1 Pseudocode for Analysis Phase

1: **INPUT:** Synthesized RTL description of the design.
2: **INPUT:** Real-time switching activity (.VCD).
3: **INPUT:** OpenAccess database of the design and technology library files.
4: **INPUT:** Information about the leakage power and the virtual ground capacitance of the standard cell library.
5: Call GenerateNetList()
6: Call ReadDesign()
7: Call ReadLeak()
8: Call ReadCVGND()
9: Call ParseVCD()
10: GenerateClockTree() {
11: if (NetName = clk) then
12: while (NumberOfCells != NULL) 
13: if (TypeofCell = Register) then
14: if (TypeofRegister! = clockgated) then 
15: Task : Mark register
16: else
17: Task 1 : Mark register
18: Task 2 : Identify register properties.
19: while (TreeNode! = Register) do
20: Call FindEnablePin()
21: Task 1 : Add cells associated with the same enable pin.
22: Task 2 : Generate a clock-tree.
23: end while
24: end if
25: Identify the activation function
26: else
27: Call FindRegister()
28: end if
29: endwhile
30: end if}
31: Call GenerateClusters()}
both the cluster 4 and cluster 5 are turned off. Total 6 clusters are to be partitioned into 5 bounds. The defined cost-function should be intelligent enough to decide that what cluster will map to which bound.

Figure 3.7(b) shows how the clusters are mapped to the bounds after the clustering phase. As it will be discussed in subsequent chapters that there is a possibility of not power gating some clusters, such non power-gated clusters will be allocated to a special bound called "null bound" or "sink". In this example cluster 3 is non power-gated. The problem of mapping of the clusters to bounds is further explained and discussed in the chapter 4.

It is important to mention here that according to this methodology power-gating is applied to only clock-gated clusters.

The clustering phase is provided with the information regarding the clock-gated cluster parameters (leakage, propagation delay, area etc.) and the activation function for each clock-gated cluster as the input.

The figure 3.8 shows the conceptual formation of the cluster by grouping the standard logic gates. And according to the dimension of this cluster appropriate sleep transistor is being chosen.

The Algorithm for Clustering Phase

The stepwise functionality of the clustering phase can be enlisted as below,

1. The information regarding the activation function is converted into the strings of 1’s and 0’s discretizing the active or idle situation of the respective net.
2. Pre-computation of the binary vector associated to activation function and the attraction among the clusters. Here the measure of idleness represents the percentage indicating the occurrence of the value ’1’ in the binary vector form of the activation function.

3. Computation of the cost-function. And generation of a undirected, connected and weighted graph.

4. Running the greedy partitioning heuristics for efficient power-gating.

5. Generation of the final bounds ready for the physical placement and routing.

The pseudocode for the clustering phase can be drawn as shown in algorithm 3.
Algorithm 2 Pseudocode for Clustering Phase

1: **INPUT:** \( N \) = Number of Clusters, \( P \) = Number of Edges, \( K \) = Number of Desired Bounds.
2: **INPUT:** Synthesized RTL description of the design.
3: **INPUT:** Information of the activation function.
4: **INPUT:** Information of the clock-gated and clock-gated intersection clusters.
5: **INPUT:** OpenAccess database of the design and technology library files.
6: Call ReadActivationFunction()
7: Call ReadClusterInformation()
8: \( \text{Idleness} \leftarrow \text{GenerateIdlePeriods()} \)
9: if \((\text{TypeCluster} = \text{Pure})\) then
   10: while \((\text{iter } \leq \text{SimulationTime})\) do
      11: if \((\text{BitCompare(ActivationFunctionVector)}) = 1)\) then
         12: \( \text{CountIdleness}++ \);
         13: \( \text{ActivationFunctionVector} \leftarrow 1 \)
      14: end if
   15: end while
else
   16: Call Find(Symbol1)
   17: Call Find(Symbol2)
   18: while \((\text{iter } \leq \text{SimulationTime})\) do
      19: \( \text{ActivationFunctionVector} \leftarrow ((\text{ActivationSymbol1})|(\text{ActivationSymbol2})) \)
   20: end while
   21: while \((\text{NumberFanouts} \neq 0)\) do
      22: while \((\text{iter } \leq \text{SimulationTime})\) do
         23: \( \text{AuxVector} \leftarrow ((\text{ActivatioFunctionVector})|(\text{ActivationSymbol})) \)
      24: end while
   25: end while
   26: while \((\text{iter } \leq \text{SimulationTime})\) do
      27: if \((\text{BitCompare(ActivationFunctionVector)}) = 1)\) then
         28: \( \text{CountIdleness}++ \)
         29: \( \text{ActivationFunctionVector} \leftarrow 1 \)
      30: end if
   31: end while
end if
32: Call ReadCriticalNet()
33: \( \text{Coupling} \leftarrow \text{SignalShared()} \)
34: \( \text{CriticalityFactor} \leftarrow \text{FindCriticalNet()} \)
35: Call GraphGeneration()
36: \( \text{iterations} \leftarrow \text{Clustering(Graph, Edge, N, P, K)} \)
37: Call BoundGeneration();

†described in chapter [4]
‡described in chapter [5]
Chapter 4

Placement Aware Clustering

“These thoughts did not come in any verbal formulation. I rarely think in words at all. A thought comes, and I may try to express it in words afterward.”

Albert Einstein

This chapter mathematically models the problem using the concepts of traditional graph theory. Section 4.1 discusses the formulation of a one to one function measuring attraction among the clusters based upon the cluster properties such as switching activities and propagation delay etc. In section 4.2 a new concept of critical interconnects is introduced to overcome the timing overhead seen in the proposed solution.

Now, based on the discussion in chapter 3 the problem can be described as the mapping of each of the $n$ clusters belonging to the set $C = \{ c_1, ... , c_n \}$ to one of the suitable bound corresponding to the set $B = \{ b_1, ... , b_k \}$ and $\forall k, k < n$.

This problem can be solved by two methods

1. Placement Aware Clustering

2. Clustering Aware Placement

Commercially available tools for placement and routing acquire an interconnect delay dominating cost function for physical placement. Hence they produce optimal results as far as propagation delay is concerned. They do not consider other design metrics like static power and cell proximity to support fine-grained power-gating. To add this kind of “power-awareness” playing around the clustering phase is the preferred choice. By applying placement aware clustering, the placement can be characterized as “locally-timing-oriented” and “globally-power-oriented”, which is fully compliant with commercial design flow, and it does not require any kind of modifications in the algorithms of standard placement tools.
This guarantees minimum timing overhead with maximum integrability.

In contrast, the literature survey concludes that a clustering aware placement solution may require:

- Tedious and complex modifications of placement algorithm resulting in longer development time.
- Less compatibility with commercial design tools.
- Larger timing overhead.

The proposed synthesis flow in this thesis adapts the placement aware clustering methodology [12, 11].

Ideally the number of bounds can be equivalent to the total number of clusters but as discussed earlier this will end up in unacceptable timing and area overheads. Hence it is quite necessary to find a smarter approach to merge the clusters until the desired number of bounds are achieved. For the merging of the clusters, it is important to define a function which measures the attraction between the two clusters. This function should include the properties of the clusters which significantly affects the overall behavior of the circuit.

In [13] the cost-function is described as the function of two characteristics for any pair of clusters. The two characteristics are described as follow,

- **Idleness**
  
  This feature characterizes the amount of overlapping among the clusters. If the two clusters are significantly overlapped then they should be merged together at the loss of some power savings. And if the clusters are leaky enough and the overlapping between them is smaller then, they should not be merged together.

- **Degree of Coupling**

  Although all the pure clusters are completely disjoint among them selves, they share signal dependencies with corresponding hybrid clusters, and vise versa. Referring to figure [3.6] it can be observed that there are shared signals between cluster 1 and cluster 1/2, cluster 2 and cluster 1/2 while cluster 1 and cluster 2 do not share any signals. The clusters with maximum amount of coupling should be kept together. This will help to shorten the interconnect lengths and hence improve the propagation delay.
4.1 Mathematical Modeling

4.1.1 Defining the Cost-function

The problem can be described easily by traditional graph theory [24]. The set of \( n \) clusters can be modeled as undirected graph \( G(V,E,w) \), where \( V = \{ v_1, \ldots, v_n \} \) are the vertices representing \( n \) clusters, \( E \) represents the edges, forming one to one relation among the clusters, \( w \) represents the weight of the edge. \( w_{i,j} : E \mapsto \mathbb{R}^+ \).

For any cluster \( i \) and \( j \) the edge weight \( w_{i,j} \) can be defined as,

\[
    w_{i,j} = \alpha e_{i,j} + \beta \left[ \frac{I_{i,j} + I_{j,i}}{I_{tot_{i,j}}} \right]
\]

(4.1)

Where,

- \( e_{i,j} \) is the overlapping of the activation functions \( e_i \) and \( e_j \) of clusters \( i \) and \( j \) respectively. \( e_{i,j} \in [0,1] \). The \( \gamma \in [1,2] \) is a smoothing factor which reduces the importance of the lower values of \( e_{i,j} \).

- \( I_{i,j} \) is the total number of signals going from cluster \( i \) to cluster \( j \), \( I_{j,i} \) is the total number of signals going from cluster \( j \) to cluster \( i \) and \( I_{tot_{i,j}} \) is the total signals coming out from clusters \( i \) and \( j \).

- \( \alpha \) and \( \beta = 1 - \alpha \) are the values between 0 and 1.

4.1.2 Augmented Graph

It is important to note here that for the graph \( G \) all the clusters must be power-gated. To include non power-gated clusters a unique node, "sink" is added in the graph and the attraction of the rest of the nodes to the sink is defined. The generation of the sink bound is of the special interest in the modeling because it will provide the flexibility to the designer to omit some of the clusters being power-gated mainly because one of the following reasons,

1. They are too small.
2. They are not leaky enough to qualify for the power-gating.
3. They are active most of the time.

The graph is modified to \( G'(V',E',w') \). where \( V' = V \cup \{ v_0 \} \). \( v_0 \) represents the sink. \( E' \) defines the edges. The edge weight \( w'_{i,j} : E' \mapsto \mathbb{R}^+ \).

\[
    w'_{i,j} = \begin{cases} 
        w_{i,j}, \forall i \neq 0. \\
        (\delta(1-s_i)^p + \eta(1-e_i) + \lambda(1-l_i)), \forall i = 0.
    \end{cases}
\]

(4.2)

Where,
• $s_i$ is the size of the cluster $i$. It actually represents a fraction of the physical layout row. A significantly smaller value will force to put cluster $i$ to the sink. A sufficiently larger exponent $\rho$ (i.e $\rho > 5$ ) will make sure of this merging.

• $e_i$ is the idleness of cluster $i$.

• $l_i$ is the normalized value of leakage for cluster $i$. Which can be obtained by dividing the leakage of any cluster to the total leakage of all the clusters, i.e $l_i = \left[ \frac{L_i}{\sum_{j=1,\ldots,n,j\neq i} L_j} \right ]$

It is worth noticing that here since the edges model attraction to the sink. Hence the weight consist of the complemented quantities defined as waste of space($1 - s_i$), activity($1 - e_i$) and no leakage($1 - l_i$). And $\delta + \eta + \lambda \equiv 1$.

4.2 Introduction to Critical Interconnects

Based on the observation of timing overhead seen in the results obtained with only common idle conditions in the equation 4.1, one more characteristic of the clusters called criticality factor has been introduced in equation 4.1 to make it more intelligent. In the subsequent discussion it is also made clear that in equation 4.1 the degree of coupling is necessary but not sufficient characteristic to overcome the timing overhead.

• Criticality factor

This characteristic takes the possibility of the shared critical paths between two clusters into consideration. For any given net the criticality factor $cf_{i,j}$ between two clusters $i$ and $j$ can be defined as,

$$cf_{i,j} = \left[ \frac{WCCpath_{net}}{WCCpath_{circuit}} \right ]$$

(4.3)

Where,

• WCCpath$_{net}$ is the worst-case critical path through the net.

• WCCpath$_{circuit}$ is the worst-case critical path of the whole circuit.

And $cf_{i,j} \in [0, 1]$. For the better visualization of the model, a conceptual case is explained in figure 4.1.

As it can be seen in the figure 4.1 among the 3 clusters, cluster 1 and cluster 2 shares 3 signals and cluster 1 and cluster 3 shares only one connection which lies on the critical path. Now in general cases with the equation 4.1 more weight will be given to the edge between cluster 1 and cluster 2 regardless of the critical path. But with the criticality factor now more weight will be given to the edge between...
cluster 1 and cluster 3. This also clears the argument of insertion of the \textit{criticality factor} along with \textit{degree of coupling}.

Hence, including equation 4.3 to equation 4.1 means giving priority to the critical interconnects between two clusters.

\[
w_{i,j} = \alpha e_{i,j}^{-i} + \beta \left[ \frac{I_{i,j} + I_{i,\text{tot}}}{I_{\text{tot},j}} \right] + c_{f_{i,j}}
\]  

(4.4)

Correct identification of the critical path of the whole circuit and improvements based upon that critical path has always been a challenging task for the designers. One of the major challenges faced in this work and especially while introducing the concept of the critical interconnects was the situation when the same critical net is shared among many number of clusters. In this situation if all the critical nets will be allocated the same static weight then it will not satisfy the purpose to give more priority to the critical most net. To overcome this problem a dynamic weight should be allocated to such nets with criticality factor greater than the average criticality factor of the respective cluster. And in this case the weight allocated is ,

\[
c_{f_{i,j}} = 1 + c_{f_{i,j}}'
\]  

(4.5)

Where,

- \( c_{f_{i,j}}' \) is the criticality factor between cluster \( i \) and \( j \).

A simple heuristic is implemented to calculate the described cost-function of equation 4.4 which is explained as below and a pseudocode is developed as shown in algorithm 3.

1. Collect the information regarding the activation function for every clock-gated clusters.

Figure 4.1: Conceptual Representation of Critical Interconnects.
2. Collect the various properties such as total leakage, area propagation delay etc. of all clock-gated clusters.

3. Identify the net corresponding to the activation function.

4. Identify the criticality of all the output nets for every cluster. Compute the average criticality factor for every cluster. For any cluster $i$ with $n$ number of output nets then the average($cf_i$) = $\left\{ \frac{\sum cf_i}{n} \right\}$.

   Now all the nets with $cf_i > \text{average}(cf_i)$ should be taken into consideration to compute the $cf_{i,j}$ of the equation 4.4.

5. Compute the shared signals by comparing the type of the nets between two clusters. This computation is based upon the fact that all the shared signals between two clusters will have a common net name.

6. $\text{Graph}_{i,j} \Leftarrow \alpha \ast \text{Idleness}_{i,j}^\gamma + \beta \ast \text{Coupling}_{i,j} + cf_{i,j}$
Algorithm 3 Pseudocode for Computing Cost-Function of Equation 4.4

1: \( \text{AvgCriticality} \leftarrow 0 \)
2: \( \textbf{while} (\text{NumberOfClusters}1 \neq 0) \) \( \textbf{do} \)
3: \( \quad \text{Counter}1 \leftarrow 0 \)
4: \( \textbf{while} (\text{NumberOfClusters}2 \neq 0) \) \( \textbf{do} \)
5: \( \quad \text{Counter}2 \leftarrow 0 \)
6: \( \quad \textbf{if} ((\text{NumFanouts} = 1) \|(\text{NameCluster}1 = \text{NameCluster}2)) \) \( \textbf{then} \)
7: \( \quad \quad \text{SharedSignals} \leftarrow \text{NULL} \)
8: \( \quad \quad \text{CriticalityFactor} \leftarrow \text{NULL} \)
9: \( \) \( \textbf{else} \)
10: \( \quad \textbf{while} (\text{NumberOfNets}1 \neq 0) \) \( \textbf{do} \)
11: \( \quad \quad \textbf{if} (\text{TypeofNet} = "Q" \| "QN" \| "Z") \) \( \textbf{then} \)
12: \( \quad \quad \quad \textbf{while} (\text{NumberOfNets}2 \neq 0) \) \( \textbf{do} \)
13: \( \quad \quad \quad \quad \textbf{if} (\text{NameNet}1 = \text{NameNet}2) \) \( \textbf{then} \)
14: \( \quad \quad \quad \quad \quad \text{Counter}1 \text{++} \)
15: \( \quad \quad \quad \quad \text{Counter}2 \text{++} \)
16: \( \quad \quad \textbf{end if} \)
17: \( \quad \) \( \textbf{end while} \)
18: \( \quad \text{AvgCriticality} \leftarrow \frac{\sum \text{CriticalityNet}}{\text{Counter}1} \)
19: \( \textbf{end if} \)
20: \( \textbf{while} (\text{NumberOfCells} \neq 0) \) \( \textbf{do} \)
21: \( \quad \textbf{if} (\text{NetCriticality} \geq \text{AvgCriticality}) \) \( \textbf{then} \)
22: \( \quad \quad \textbf{while} (\text{NumberOfNets}3 \neq 0) \) \( \textbf{do} \)
23: \( \quad \quad \quad \text{CriticalityFactor} \leftarrow (1 + \text{CriticalityNet}) \)
24: \( \quad \) \( \textbf{end while} \)
25: \( \) \( \textbf{end if} \)
26: \( \) \( \textbf{end while} \)
27: \( \) \( \textbf{end while} \)
28: \( \) \( \textbf{end if} \)
29: \( \text{InOut} \leftarrow \text{Counter}2 \)
30: \( \) \( \textbf{end while} \)
31: \( \text{TotalOut} \leftarrow \text{Counter}1 \)
32: \( \) \( \textbf{end while} \)
33: \( \text{Coupling} \leftarrow \frac{\text{InOut}}{\text{TotalOut}} \)
Chapter 5

Clustering Heuristics

“Attacking is the only secret. Dare and the world yields, or if it beats you sometimes, dare it again and you will succeed.”

William Makepeace Thackeray

In this chapter the section 5.1 introduces the required concepts of graph theory and also addresses the problems during the partition of the graphs. Section 5.2 gives detail knowledge about the heuristics developed for the partitioning of the graphs, they are based on two different approaches namely the greedy approach and the other one is based on an iterative approach involving the concept of centroids.

5.1 Introduction of Graph Partitioning and Minimum $k$-cut Problem

Consider a connected, undirected graph $G = (V, E)$ with a weight function $w : E \rightarrow \mathbb{R}^+$ and $V = \{v_1, \ldots, v_n\}$ is a set of vertices. It is also noticeable that $G$ does not contain any self edges. A cut is defined by a partition of $V$ into disjoint sets $V'_1$ and $V'_2$ and consists of all edges $E' \in E$ which have one vertex in $V'_1$ and the other in $V'_2$. The weight of this cut is defined as $w(E') = \sum_{e \in E'} w(e)$ [24, 76].

In general a good solution to the graph partitioning problem merges the vertices together such that,

1. The sums of vertex weights are approximately equal for each resulting partition. This means that the problem is load balanced.

2. As few edges cross partition boundaries as possible. This minimizes communication, since each crossing edge $e_{i,j}$ means that $V_j$ must be sent to the partition containing $V_i$.

The figure 5.1 illustrates such a partitioning of a solution space distributed in the form of an 8x8 matrix into 4 partitions colored blue, red, green and magenta.
The case of power aware placement and routing, undertaken as a part of this work is more specific to the general partitioning idea. Converging from the general case, the idea of vertex weight on the basis of discussions in section 4.1 of chapter 4 can be defined as,

\[ V_{w_i} = f(A_i, L_i, P_{Delay_i}) \]  \hspace{1cm} (5.1)

Where,

- \( A_i \) is the actual physical area acquired by the cluster \( i \).
- \( L_i \) total leakage power of the cluster which is the summation of the leakage power of all the standard cells located into the cluster \( i \).
- \( P_{Delay_i} \) is the total propagation delay for any net of the cluster \( i \).

However, it is worth noticing that this vertex weight is used during the analysis phase to identify the potentially non-power gated clusters and later on it will be also used by the final place and route tools to calculate the size of the sleep transistors. And now the most common problem involving the cuts is to find the minimum-cost cut separating any two nodes \( t_i \) and \( t_j \) [41]. Such nodes will be referred as the terminal nodes.

There are two \textit{NP-hard} methods to deal with such problems[38],

1. \textit{Multiway cut} : If there is a set of terminal vertexes \( T = \{t_1, \ldots, t_n\} \subseteq V \). Now find the set of edges \( E' \subseteq E \) such that \( w(E') \) is minimum and removal of such edges will separate the pair of terminal nodes.
2. Minimum $k$-cut: If the desired number of partitions $k$ are provided, then find the set of minimum weight edges $E' \subseteq E$, and removal of such edges will produce $k$ connected partitions.

At first both the methodologies look alike. However, the multiway cut methodology is polynomial time solvable when $k = 2$, $\forall k>2$ it is $NP$-hard [38]. Other way round the minimum $k$-cut problem is polynomial time solvable for any fix value of $k$. A problem is polynomial time solvable only if it has the algorithmically relevant combinatorial structure that can be used as platform to efficiently home in on a solution. The process of designing a polynomial time algorithm is a two-pronged attack: unraveling this structure in the given problem and finding algorithmic techniques that can exploit this structure. Further discussion and the proof of the minimum $k$-cut theorem is given in the appendix C.

For a graph with $n$ number of vertices, the time taken by minimum $k$-cut problems is of the order of $O(n^{k^2})$. Although it is polynomial, a practical solution is still impractical for smaller values of $k$. This concludes that the partitioning problem to be solved heuristically. Several good heuristics do exists to solve minimum $k$-cut problems [43, 66].

The figure 5.2 shows three different configurations keeping the total number of vertices and the desired number of partitions constant. In this example the 8 vertices are shown as grey circles and 4 partitions colored blue, red, yellow and magenta respectively. Now there exist several ways to partition the vertices into required partitions. But under the application of minimum $k$-cut the configuration 2 will be the worst possibility and should be avoided. The main reason behind
this is the total number of edges crossing the partitions shown as dotted lines in the figure should be minimum. In the configuration 1 and 3 the total number of edges to be cut will be 6 while configuration 2 has all 10 edges crossing the partitions.

As a part of this thesis one heuristics, which reduces the number of crossing edges is developed and implemented.

5.2 Partitioning Methodologies

5.3 Greedy Heuristic

A greedy algorithm can be described as any algorithm that follows the problem solving heuristic of optimally choosing the local optimal for each stage with a final aim to reach a global optimum solution. A simple example of the application of the greedy strategy to the traveling salesman problem yields an algorithm which does the following: "At each stage visit the unvisited city nearest to the current city".

In general, greedy algorithms have:

1. A candidate set, from which a solution is created.
2. A selection function, which chooses the best candidate to be added to the solution.
3. A feasibility function, that is used to determine if a candidate can be used to contribute to a solution.
4. An objective function, which assigns a value to a solution, or a partial solution, and
5. A solution function, which will indicate when a complete solution has been discovered.

Now comparing to the problem undertake as a part of this work, the graph \( G = (V, E) \) is a candidate set, selection function is the cost-function discussed in chapter. Simple algorithms were implemented to check the feasibility of the selected solution and to indicate the discovery of the complete solution and termination of further processes.

A pseudocode of the implemented algorithm is shown in algorithm

First of all the edge weights are mapped to respective edges. Once the edges are allocated with the weights sorting of edges is carried out in increasing order of
5.3 Greedy Heuristic

Algorithm 4 Pseudocode for the Greedy Heuristic

1: **INPUT:** Graph \( G = (V, E) \) with edge weights \( w : E \mapsto \mathbb{R}^+ \)
2: Call GenerateEdges()
3: Call SortEdges()
4: \( i \leftarrow 0 \)
5: **while** \( (N_{cc} \leq k) \) **do**
   6: Remove edge \( e_i \) from \( G \)
   7: \( N_{cc} = \text{Compute}_{cc}(G') \)
   8: \( i \leftarrow i + 1 \)
6: **end while**

weight. And then iteratively the edges are removed in that order until the graph consists of exactly \( k \) connected components abbreviated as CC.

The functionality of \( \text{Compute}_{cc}(G', n) \) is shown in algorithm 5.

Algorithm 5 Pseudocode for the \( \text{Compute}_{cc}() \)

1: **INPUT:** Graph \( G' = (V', E') \) with edge weights \( w : E' \mapsto \mathbb{R}^+ \)
2: **INPUT:** Desired number of partitions \( n \).
3: Allocate sufficient memory to VisitedNode
4: Allocate sufficient memory to TraverseNodeList
5: /* initialize vertices to unvisited */
6: **while** \( (\text{Vertices} \leq n) \) **do**
7: \( \text{VisitedNode}[\text{Vertices}] \leftarrow 0 \)
8: **end while**
9: **while** \( (\text{Vertices} \leq n) \) **do**
10: /* if a vertex is unvisited */
11: **if** \( !\text{VisitedNode}[\text{Vertices}] \) **then**
12: \( \text{UCS}(\text{Vertices}, G', n, \text{VisitedNode}, \text{TraverseNodeList}) \)
13: **end if**
14: \( \text{Component} + + \)
15: **end while**

The pseudocode of UCS-Uniform Cost Search is dictated in algorithm 6.

The UCS-Uniform Cost Search is the modified version of the BFS-Breadth First Search [63]. UCS is mainly used for searching paths in weighted graphs. Like BFS in UCS the search begins at the root vertex and it continues with all the neighboring vertices which has the least cost from the root. All vertices are visited in this manner until a final solution is reached.

The main points related to the algorithm described above can be enlisted as follows,
Algorithm 6 Pseudocode for the \textit{UCS()}

\begin{align*}
1: & \textbf{INPUT:} \text{ Graph } G' = (V', E') \text{ with edge weights } w : E \rightarrow \mathbb{R}^+ \\
2: & \textbf{INPUT:} \text{ Desired number of partitions } n. \\
3: & \text{ Vertices, VisitedNodes, Component.} \\
4: & \text{ Initialize a Queue} \\
5: & \text{ VisitedNode}[\text{Vertices}] \leftarrow 1 \text{ /* mark this vertex as visited */} \\
6: & *\text{component} \leftarrow \text{Vertices} \text{ /* and store it to the component */} \\
7: & i \leftarrow 0 \\
8: & \textbf{DO} \{ \\
9: & \text{ /* if adjacent vertex is unvisited */} \\
10: & \text{ if } ((G'[\text{Vertices}[i]] = 0) \& (\text{VisitedNode}[i])) \text{ then} \\
11: & \text{ Enqueue the component} \\
12: & \textbf{end if} \\
13: & \text{ Vertices } \leftarrow \text{Dequeue}(i) \text{ } \text{ /* get next vertex to search */} \\
14: & \textbf{WHILE} \text{ (All vertices are visited)} \\
15: & \text{ ResetQueue(Queue) } \text{ /* free memory */} \\
\end{align*}

- It involves expansion of the vertices by adding all unexpanded neighboring vertices that are connected through weighted edges to a priority queue.

- In the queue, each vertex is associated with its total path cost from the root vertex, where the least-cost paths are given highest priority.

- The vertex at the head of the queue is expanded, adding the next set of connected vertices with the total path cost from the root to the respective vertex in the graph.

Here, UCS guarantees that (if all edge weights are non-negative) the shortest path to a particular node has been found once the node is extracted from the priority queue. Uniform-cost search is a special case of the A* search algorithm (pronounced A star) provided that the heuristic is a constant function. Breadth-first search (BFS) is a special case of uniform-cost search when all edge costs are positive and identical. Another difference is in terms of execution and that is BFS first visits the vertex with the shortest path length (counting number of vertices, irrespective of the weight edges) from the root vertex, UCS first visits the vertex with the shortest path costs (sum of edge weights of the edges connecting respective vertices) from the root vertex.

The figure 5.3 shows the final partitions resulting from this greedy heuristic. In figure the left part shows how each vertex of the design under-test is related to the other vertices. The number on the top of the arrows shows the total number of connecting edges. In the right hand part of the image, the final distribution of the ”\textit{population of solutions}” measured by the calculation of the cost-function into
5.4 Heuristic Based upon the Concept of Centroids

The main motivation of this method is the possibility that the solution space might be grouped into natural clusters. In this case by implementing this heuristic a lot of improvement in the execution time is possible. The other important thing to mention here is that unlike the greedy heuristic this approach models the same weights formulated in chapter 4 as vertex weights instead of the edge weights. And based upon this idea the following heuristic will treat the vertices as an independent identity and iteratively try to merge the vertices.
5.4.1 Centroids Oriented Clustering

In the development of this heuristic an assumption is made that the clusters can be modeled as spheres. And this spheres can be characterized by a single point called centroid. In this discussion the term centroid means the center of mass for any cluster.

![Figure 5.4: Generation of Centroids.](image)

The figure 5.4 shows the distribution of the solution of the cost function of equation 4.4. The red dots are the normal values and the black dots are the points representing the centroids of the clusters.

For initial partitioning, a bottom-up clustering approach is adapted. It begins by assuming every cluster as a possible potential centroid. And then the clusters will be merged together according to the weight given to them until the desired number of centroids are achieved. A simplest method is to combine to closest centroids forming a single centroid until a desired number is reached.

Now, to measure this closeness of the clusters a distance metric or distance function $f(i, j)$ for any two values $i$ and $j$ of the cost-function, is used which satisfies the following conditions,

1. $f(i, i) = 0, \forall i \in G$
2. $f(i, j) = f(j, i), \forall (i, j) \in G$
3. For any third value of $k$, $f(i, k) \leq f(i, j) + f(j, k)$

A number of metrics are available for the calculation of this function, a simple Euclidean function is used which is also know as $L^2$ metric and in general. It is a...
5.4 Heuristic Based upon the Concept of Centroids

part of the family of $L^p$ metric.

$$f_p(i, j) = \left( \sum_{n=1}^{N} (i_n - j_n)^p \right)^{\frac{1}{p}} \tag{5.2}$$

Simplifying equation [5.2] for the 2-D application of partitioning planner graph and putting $p = 2$ yields,

$$f_2(i, j) = \sqrt{(i_1 - j_1)^2 + (i_2 - j_2)^2 + \ldots + (i_n - j_n)^2} \tag{5.3}$$

Equation 5.3 is the standard notion of distance in Euclidean Space.

Now while clustering, a care should be taken that each value point of the cost-function should be kept as close as possible to the nearest centroid. This implies that for the set of the values of graph $G = \{g_1, \ldots, g_n\}$ and a set of centroids $C = \{c_1, \ldots, c_m\}$ where set $C$ is the prefix of set $G$ and $m \leq n$, for any point $g_n \in G$ the nearest centroid $c_{i(n)}$ then $f(g_n, c_{i(n)}) = \min_m f(g_n, c_m)$.

This demands a measuring term which can indicate the amount of distortion of the given point in the set $G$ from the possible closest centroid of the set $C$. This term can be defined as follows,

$$\text{Dist}(C) = \left[ \sum_{n=1}^{N} f(g_n, c_{i(n)}) \right] \tag{5.4}$$

The pseudocode of the above discussed heuristic is given in algorithm 7.

Some issues can be raised based upon the optimality of this minimization of Dist($C$) of equation 5.4. The method described above provides with the locally optimum values of the centroids it is a good initial point but not the optimum. To improve the optimality an iterative refinement method is implemented which is know as $k$-mean refinement [15].

5.4.2 K-mean Refinement

As discussed above in the clustering method adapted here, there is no guarantee that the generated set of centroids will minimize the distortion. Once the total number of desired centroids are decided, suppose $k$ and the initial set of centroids are obtained by the above method, now this refinement will iteratively move the centroids such that the distortion is the minimum [59].

Suppose the set of the values of graph $G = \{g_1, \ldots, g_n\}$ and a set of initial $k$ centroids is denoted by $C_0 = \{c_1^0, \ldots, c_k^0\}$. Now at first iteration of the $k$-mean refinement will produce a new set of centroids $C_1 = \{c_1^1, \ldots, c_k^1\}$ such that
Algorithm 7 Pseudocode for the Heuristic based on Centroids.

1: \textbf{INPUT:} Graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}^+$
2: /* Computation of Distance Matrix and Local Minima */
3: \textbf{while} ($i \geq \text{NumberOfClusters}$) \textbf{do}
4: \hspace{1em} \text{Initialize min}
5: \hspace{1em} \textbf{while} ($j \leq i$) \textbf{do}
6: \hspace{2em} $k \leftarrow j + 1$
7: \hspace{2em} \textbf{while} ($k \leq i$) \textbf{do}
8: \hspace{3em} $\text{dist}[j][k] \leftarrow 0$
9: \hspace{3em} \textbf{while} ($e \leq \text{NumberOfColumns}$) \textbf{do}
10: \hspace{4em} $\text{dist}[j][k]+ = (\text{Graph}[j][e] - \text{Graph}[k][e])^2$
11: \hspace{3em} \textbf{end while}
12: \hspace{3em} \textbf{if} ($\text{dist}[j][k] \leq \text{min}$) \textbf{then}
13: \hspace{4em} $\text{ind}[0] \leftarrow j$ and $\text{ind}[1] \leftarrow k$
14: \hspace{3em} $\text{min} \leftarrow \text{dist}[j][k]$
15: \hspace{3em} \textbf{end if}
16: \hspace{2em} \textbf{end while}
17: \hspace{1em} \textbf{end while}
18: /* Merging */
19: \textbf{while} ($e \leq \text{NumberOfColumns}$) \textbf{do}
20: \hspace{1em} $\text{Graph}[\text{Ind}[0]][e]+ = \left[\frac{(\text{Graph}[\text{Ind}[0]][e] - \text{Graph}[\text{Ind}[1]][e])^2}{2}\right]$
21: \hspace{1em} \textbf{end while}
22: \textbf{end while}
23: \textbf{OUTPUT:} Set of centroids $C = \{c_1, ..., c_m\}$ with value points from the set $G$
5.4 Heuristic Based upon the Concept of Centroids

\[ \text{Dist}(C^1) \leq \text{Dist}(C^0). \]

As from the above discussion for any point \( g_n \in G \) the nearest centroid \( c_{i(n)} \) then \( f(g_n, c_{i(n)}) = \min_k f(g_n, c_k) \), a new set of values \( G'_k \) should be defined as shown in equation 5.5 whose data points are closer to the centroid \( c_k^0 \).

\[ G'_k = \{ g_n : i(n) = k \} \]  

(5.5)

Now, it is very easy to define the new \( k_{th} \) centroid after one iteration as,

\[ C^1_k = \frac{1}{\|G'_k\|} \sum_{g_n \in G'_k} g_n \]  

(5.6)

Once the \( C^1_k \) is ready, further refinement will be carried out until it converges, and every subsequent iteration will reduce the distortion. It is worth noticing that the quality of this refinement entirely depends upon the quality of initial set of centroids \( C^0 \).

![Figure 5.5: Improvements in the Distortion Metric by k-mean Refinement.](image)

The figure 5.5 shows the refinement by \( k\)-mean algorithm and the improvements in the distortion with respect to the number of iterations considering that the distortion in \( C^0 \) is 1. It can be seen in the figure that the clustering heuristic used here gives the best initial choice as from \( C^0 \) to \( C^1 \) the distortion reduces by 64%. Once the refinement starts at the end of \( 10^{th} \) iteration a total improvement of 69% is observed. For better visualization the \( C^0 \) is not shown in figure. After certain iterations the distortion becomes constant or gives very little improvements.

Similar to other algorithm, \( k\)-mean clustering has many weaknesses:

1. When the numbers of data are not so many, initial grouping will determine the cluster significantly.
2. We never know the real cluster, using the same data, if it is inputted in a different way may produce different cluster if the number of data is a few.

3. We never know which attribute contributes more to the grouping process since we assume that each attribute has the same weight.

One way to overcome these weaknesses is to use \textit{k-mean} refinement only if there is sufficient data available. A pseudocode of the \textit{k-mean} algorithm can be drawn as displayed in algorithm 8.
Algorithm 8 Pseudocode for the \textit{k-mean} Refinement.

1: INPUT: Graph $G = (V, E)$ with edge weights $w : E \rightarrow \mathbb{R}^+$
2: INPUT: initial set of centroids $C^0$. Number of Iterations
3: /* Computation of Distance Matrix and Local Minima */
4: while (NumIteration != 0) do
5:   Initialize distortion[NumIteration] = 0;
6:   while ($j < \text{NumRows}$) do
7:     Initialize $MIN$;
8:     while ($k < \text{NumCentroids}$) do
9:       Initialize $\text{dist}[j][k] = 0$;
10:      while ($l < \text{NumCols}$) do
11:        $\text{dist}[j][k] += (\text{Graph}[j][e] - \text{cent}[k][e])^2$;
12:        if ($\text{dist}[j][k] < rMin$) then
13:          $\text{BestCentroid}[j] = k$;
14:          update $MIN \Leftarrow \text{dist}[j][k]$
15:        end if
16:      end while
17:    end while
18: end while
19: while ($k < \text{NumCentroids}$) do
20:  Allocate memory to CentNew equivalent to the size of Graph
21:  while ($j < \text{NumRows}$) do
22:    if ($\text{bestCent}[j] == k$) then
23:      while ($l < \text{NumCols}$) do
24:        $\text{CentNew}[e] += \text{Graph}[j][l]$;
25:        $\text{Count} \Leftarrow 0$
26:      end while
27:    end if
28:    if ($\text{Count} > 0$) then
29:      while ($l < \text{NumCols}$) do
30:        $\text{Cent}[k][l] = \text{CentNew}[l]/\text{tot}$;
31:      end while
32:    else
33:      while ($l < \text{NumCols}$) do
34:        $\text{Cent}[k][l] = \text{Mean}[l]$;
35:      end while
36:    end if
37:  end while
38: end while
39: end while
40: OUTPUT: Set of refined centroids $C^r = \{c^r_1, \ldots, c^r_k\}$ with value points from the set $G$
Chapter 6

Experimental Setup and Results

“People love chopping wood. In this activity one immediately sees results.”
Albert Einstein

This chapter discusses the results obtained during the experiments performed on developed prototype on standard benchmarks. Section 6.1 describes the necessary experimental set-up and a brief description of the standard benchmarks on which the proposed synthesis flow is tested. Subsection 6.2.1 of section 6.2 discusses the results using the criticality factor in the cost metric. Subsection 6.2.2 of section 6.2 discusses the results obtained using different heuristics proposed in this thesis work.

6.1 Experimental Setup

The entire flow is validated on the RTL descriptions of a real-life circuits used in typical network-on-chip architectures, a NoC switch and Serial Peripheral Interface bus. The design description in brief is provided as below.

6.1.1 Description of the Benchmarks

1. Switch_6x6_2_6

The functionality of the module is to analyze the incoming packets, identify the source and the destination addresses and route the packets. The design consists of more than 8K logic gates. It consists of total 852 registers. So that the probability of reducing dynamic power is more. The design is sufficiently large and it consists of 124 clock gating clusters.

2. Serial Peripheral Interface Bus

The main functionality of this design is data transfer in communication centric processors. It provides several communication modes required for efficient data
transfer. It is relatively smaller design and consists of total 1K standard logic gates. It also consists of 71 registers and has potential scopes for improvements. The design consists of total 22 clock gating registers.

### 6.1.2 Description of the Setup

![Experimental Flow of the Tool and Setup](image)

Figure 6.1: Experimental Flow of the Tool and Setup.

Figure 6.1 shows the experimental flow of the tool. It also describes the integration of the developed tool with the commercially available tools which are used for the synthesis and final place and route. The RTL of the design is synthesized using Synopsys PhysicalCompiler and mapped onto a 65nm CMOS standard-cell library provided by STMicroelectronics.

To measure the real-time switching activities at each and every net in the design, an output file (vcd file) is generated by the simulation of the standard test-benches over the gate-level synthesized benchmarks, in Mentor Graphics Modelsim.

A parser is developed to read the vcd file and calculate the static probabilities and the switching for the internal nets. These values are provided to Synopsys PrimeTime for the power estimation phase.

More accurate timing and power estimations are evaluated using the physical information provided with the standard-cell library. An OpenAccess database of the synthesized design is created. As mentioned in chapter 3 section 3.2.1, the analysis phase determines the feasibility of CG/PG integration. Once this is done, the clustering phase uses this information and it generates the necessary bounds for the final placement. The information regarding the bounds is provided to the standard
6.1 Experimental Setup

Placement tool, which uses an interconnect-delay dominating cost-function. With such placement strategy, there is a chance that a bound containing a set of clusters will also contain the cells belonging to other clusters. This can result in multiple interconnects for each bound and, hence, a larger area penalty. The final step is to insert the sleep-transistors. A row-based power-gating strategy as explained in chapter 3 section 3.1.3 is used.

A user-guide provided in appendix D explains the necessary commands to use the prototype and the environment variables required to be set on Unix platform.

![Figure 6.2: Modern Optimization Space for Sub-micron Technologies.](image)

The experiments are performed keeping in mind all 3 optimization parameters referring to figure 6.2. Timing performance and power are the prime concerns and it will be analyzed as a combined optimization factor called *EDP-Energy Delay Product*. It is the product of total consumed energy and propagation delay. The total energy is the product of the total power (leakage power and the dynamic power) and the total simulation time. This calculation requires the knowledge of the time interval when the block of cells is off, which can be obtained from the simulation results. The total off-time of the signal corresponds to the total shut-down time of the block and it is used to calculate the percentage of leakage which is saved. Then, this value is compared to the one obtained using the whole simulation time as time interval, which indicates the total leakage power consumed by the cluster during circuit operation. The idea is to minimize the *EDP*. Area is the actual physical area occupied by the standard cell during the placement and it will be referenced as and when the overhead is significant.

As explained in chapter 4 section 4.1, different values of alpha will be used to vary the dominance of *idleness* and *degree of coupling* to bind the clusters into pre-placed bounds. The EDP defined above varies with the different values of α and total number of bounds. The larger power-gated cluster or the pre-placed bound, the larger the virtual ground capacitance. As the virtual-ground capacitance becomes larger, the amount of energy has to be discharged during the idle-to-active transition of the block is drastically increased. At the same time, if the duration of the idle period reduces, the leakage saving introduced by this approach decreases. Since both
logic size and idle period duration are strongly related to the nature of the circuit under test, a common optimal solution for every kind of implementation does not exist. This means that varying the size of the size of the pre-placed clusters (i.e., number of bounds) or the duration of idleness (i.e., $\alpha$) different leakage savings can be achieved. Keeping this in mind, the experiments are performed with the design-space parameters shown in table 6.1.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>0.1, 0.3, 0.5, 0.7, 0.9</td>
</tr>
<tr>
<td>Bounds</td>
<td>3, 4, 5, 6, 7</td>
</tr>
</tbody>
</table>

Table 6.1: Explored Design Space Parameters

6.2 Results

Logically, there are two perspectives for the analysis of the results. One is related to the introduction of the criticality as described in chapter 4 and compare the results of having criticality as a part of cost function with the results obtained without criticality. On the other side a comparison of the results, obtained using different partitioning techniques proposed in chapter 5 is carried out.

6.2.1 Results: Part 1 "Net-criticality Matters!"

As described before, the concept of net-criticality is introduced to reduce the timing overhead and hence over all EDP. The criticality factor does not affect the energy savings. First of all, to avoid any confusion due to possible power savings achieved by the commercial tools, used in this work for final place and route stage, an experiment is carried out to confirm that having high, medium or low placement efforts in Synopsys Physical Compiler will do not affect on the leakage power in the circuit. Changing the efforts will force the placer to use efficient algorithms to improve the quality of results in terms of area optimization. It does not have any impact on the power consumption as the total number of standard cells will remain the same in any placement effort.

Figure 6.3 shows the effects of various placement efforts on leakage power, dynamic power, area and propagation delay for the switch_6x6_2.6 benchmark. And as it was predicted that the placement efforts do not affect the leakage power consumption in the circuit. The main task of the placement efforts is to reduce the total area, and as it is clear from the figure that the high placement effort, achieves almost 50% of reduction in area overhead than that of the low placement effort.
6.2 Results

Figure 6.3: Effect of Placement Efforts on Various Cost-metric Parameters.

But there is always tradeoff and hence the propagation delay will increase by 16%.
And this saving of area overhead is achieved at the expense of 100% increment in computation time.

A wise choice is to opt for the intermediate option and all the following results are performed using medium placement effort.

Figure 6.4: Comparison of Area and Power for Switch and SPI with Different Combinations of Clock and Power Gating Techniques.

An analysis is done to measure the overall savings achieved by this approach of merging clock and power gating techniques. The figure 6.4 shows normalized values of the possible overheads in total power and area for three different configurations. Without clock gating and power gating configuration values are taken as reference points and the values for with clock gating and without power gating, and with combined clock and power gating approach are normalized. As it can be seen that for combined approach the best possible power reduction achieved is upto 21% over the first configuration. But this is at the expense of 59% of area overhead for switch
and for SPI the power reduction and area overhead are 16% and 78% respectively. These results show the effectiveness of the proposed technique.

![EDP vs ALPHA](image1)

Figure 6.5: Plot of EDP to $\alpha$ for the Switch$_{6 \times 6, 2, 6}$.

![EDP vs ALPHA](image2)

Figure 6.6: Plot of EDP to $\alpha$ for the SPI.

The figure 6.5 shows the plot of energy delay product for different values of $\alpha$ and different values of bounds for the switch benchmark with criticality metric. From the figure it is clear that, maximum savings can be achieved for the total number of bounds 7. The bars in light blue color shows the values of EDP for different values of $\alpha$.

A similar observation of benchmark SPI is shown in figure 6.6. In this particular case, for different values of $\alpha$ the total desired bounds are 6 for optimum EDP. The bars in magenta color shows the values of EDP for different values of $\alpha$ for bounds 6.

For the best bound of the benchmarks switch$_{6 \times 6, 2, 6}$ and SPI, figures 6.7 (left) and 6.8 (left) shows a comparison between EDP for different values of $\alpha$, and fig-
6.2 Results

Figure 6.7: EDP and Propagation Delay for different values of $\alpha$ for 7 Bounds of the Switch.

Figure 6.8: EDP and Propagation Delay for different values of $\alpha$ for 6 Bounds of the SPI.
The foremost observation in both the figures is that, irrespective of the values of $\alpha$, the values of EDP and Propagation Delays with criticality model are smaller than the values without criticality model. This perfectly satisfy with the prediction made above.

The introduction of the criticality factor, has achieved an improvement upto 33% in the EDP and the results also indicate a significant improvement of 24% in the total timing overhead for the switch benchmark and upto 14% of total improvement in both EDP and timing overhead for SPI benchmark. For larger designs like switch, the difference is larger in the case of lower values of $\alpha$, since less preference will be given to overlapped clusters during the partitioning process. And if such clusters are not merged, it could result in longer interconnects among the overlapped clusters. For sufficiently larger values of $\alpha$ this problem does not occur.

Tables 6.2 and 6.3 summarizes the values of area overhead, leakage power, total dynamic power and propagation delay for different values of $\alpha$, for Switch and SPI benchmarks respectively.

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Area [$\mu.m^2$]</th>
<th>Leakage Power [$\mu W$]</th>
<th>Dynamic Power [$\mu W$]</th>
<th>Propagation Delay [ns]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>4.502E+04</td>
<td>5.838</td>
<td>7.412</td>
<td>5.7948</td>
</tr>
<tr>
<td>0.3</td>
<td>4.538E+04</td>
<td>5.838</td>
<td>7.47</td>
<td>5.84652</td>
</tr>
<tr>
<td>0.5</td>
<td>4.646E+04</td>
<td>5.838</td>
<td>7.753</td>
<td>6.2402</td>
</tr>
<tr>
<td>0.7</td>
<td>4.586E+04</td>
<td>5.838</td>
<td>7.791</td>
<td>5.87346</td>
</tr>
<tr>
<td>0.9</td>
<td>4.586E+04</td>
<td>5.838</td>
<td>7.791</td>
<td>5.87346</td>
</tr>
</tbody>
</table>

Table 6.2: Area, leakage and timing specifications for bounds = 7 with criticality model. Benchmark Switch_6x6_2_6.

It is important to mention here that increasing the total number of bounds from 5 to 7 will increase the total area overhead by 2% for switch and for SPI this overhead is negligible. And hence, this area overhead is not significant compared to the improvements mentioned above.

This supports the intuition that larger the designs, better the scopes for the improvements.
6.2 Results

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>Area [$\mu m^2$]</th>
<th>Leakage [\mu W]</th>
<th>Dynamic Power [\mu W]</th>
<th>Propagation Delay [ns]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>6.696E+03</td>
<td>449.5</td>
<td>709.5</td>
<td>3.18017</td>
</tr>
<tr>
<td>0.3</td>
<td>6.786E+03</td>
<td>449.5</td>
<td>714.5</td>
<td>3.22886</td>
</tr>
<tr>
<td>0.5</td>
<td>6.648E+03</td>
<td>449.5</td>
<td>712.9</td>
<td>3.19973</td>
</tr>
<tr>
<td>0.7</td>
<td>6.784E+03</td>
<td>449.5</td>
<td>709.2</td>
<td>3.22716</td>
</tr>
<tr>
<td>0.9</td>
<td>6.784E+03</td>
<td>449.5</td>
<td>709.2</td>
<td>3.22716</td>
</tr>
</tbody>
</table>

Table 6.3: Area, leakage and timing specifications for bounds = 6 with criticality model. Benchmark Spi_top.

6.2.2 Results: Part 2 “Effects of Heuristics during Optimization.”

In this section the results obtained by the greedy heuristic described in chapter 5 section 5.3 and the heuristic based upon centroids with $k$-mean refinement technique as discussed in 5.4 from the same chapter are compared.

To analyze the heuristics for specific application which is to achieve better partitions resulting in improvements in area, power and timing overheads, the execution time and quality of partitions are considered as the prime criteria. As a measure of quality the EDP- Energy Delay Product is taken. The lesser the EDP the better is the quality.

Similar experiments as performed in section 6.2.1 are done with the same experimental set-up and benchmarks.

Figure 6.9: Comparison of EDP using Greedy and Centroid Based Heuristics for switch_6x6_2_6.

Figures 6.9 and 6.10 shows the comparison of EDP using greedy and centroids based partitioning methodologies for Switch and SPI benchmarks respectively. For the ease of readability the comparison is shown only between the minimum number of bounds(#B=3) and the optimum number of bounds which are 7 and 6 for the Switch and the SPI benchmarks respectively. The results show that in the centroids
Experimental Setup and Results

Figure 6.10: Comparison of EDP using Greedy and Centroid Based Heuristics for SPI

Based on the observation, there is a very little variation in EDP by changing the \( \alpha \) or number of bounds. This observation agrees with the prediction that the final partitions depend on the initial partition.

Figure 6.11: Comparison of EDP using Greedy and Centroid Based Heuristics, Considering the Best Partitions for Switch and SPI

Figure 6.11 displays the plot of EDP for the best partitions for Switch and SPI benchmarks which are 7 and 6 respectively, for two different techniques of greedy partitioning and centroid based partitioning. It is clear that the greedy heuristic outperforms the heuristic based upon the concept of centroids. One of the possible reasons could be the lack of good quality of initial partition for centroids based approach.

Table 6.4 provides a summery of the achieved performance compared to total execution time.
In the table 6.4, the total execution time is the algebraic sum of the clustering time and the buffer time. The buffer time is the time required to build the necessary data structure used for partitioning. It can be noticed that the fastest heuristic among the two is the one based on the concept of the centroids with \textit{kmean} refinement technique which has an improvement of 94\% in the clustering time over the greedy heuristic. The main reason behind this is that in centroid based approach, each vertex is considered as an independent entity and the partition time directly proportional to the number of vertices irrespective of how the vertices are interconnected. While in greedy partitioning the partitioning time proportional to the number of edges mapping the attraction among the vertices. However, greedy heuristic improves the quality of partition by 46\% over the centroid based approach. The greedy algorithm provides better quality partition because of the reason that the quality of end partition depends upon the initial partition. The greedy approach generates coarser initial partition. Also in centroid based approach the current implementation is done with 2-D \textit{Euclidean space}, in this case there is a possibility of missing some solution points. Hence, the greedy approach provides better results.
Chapter 7

Conclusions and Future Developments

“It always seems impossible until its done.”
Nelson Mandela

This chapter provides the concluding remarks in section 7.1 on the work done in this thesis. Section 7.2 discusses the future directions for the improvements and enhancements in the current state of the work. It also provides a list of few articles for the better understanding of the concept and necessary motivation to follow up.

7.1 Conclusions and Discussions

The work of this thesis is centered around the new aspect of integration of clock and power gating for the power reduction methodology as proposed in [11]. The technique seems promising from the point of view of the achievable results, however the combination of clock-gating and power-gating is not mature enough to be applicable to real-life circuits. In particular, several aspects related to the support of integrated CG/PG in EDA frameworks are still open.

The methodology presented in [12] lacks the industrial aspect of the design flow. It did not take into account the critical paths and resulting timing violations. The timing overhead seen in [12] are unrealistic and hence the methodology cannot be applicable and integrable with the industrial design flow.

Some improvements were claimed in [13] but the aspect of matching the timing criteria was still missing and the timing overheads were 100% which makes the flow unrealistic. And also the modeling based explanation that is provided in [13] lacks an efficient implementation for the correct synthesis.

This thesis overcomes both the limitations, making the flow more realistic. In this work the cost-function proposed in [13] is correctly engineered. To overcome the bottleneck of observed timing overhead, a new concept representing the criticality of the interconnects is introduced. This new aspect is exploited in the clustering phase.
of the synthesis flow. The tool which implements this methodology is flexible enough
to work seamlessly with commercially available EDA tools for 65nm technology. The
results we have obtained show significant improvements in terms of both the energy-
delay product and the timing overhead with respect to circuits in which CG and
PG are integrated without considering interconnect criticality information during
clustering. The results also show that the integrated approach provides significant
improvements over the traditional approaches.

Efforts are also made to improve the partitioning techniques. The imple-
mented greedy heuristic is costly in terms of the execution time and to improve the
total execution time, a new heuristic is proposed which is based upon the concept
of centroids taken from classical mechanics. Necessary mathematical theories and
models are applied for efficient partitioning process. The results also compare the
two heuristics implemented under the work of this thesis. As expected the approach
based upon the centroids improves the partitioning time by 95% but at the expense
of reduced quality of partitions.

We strongly believe that achieving ultra low power designs and power opti-
mization will be the key driving force in future generation CMOS circuits due to ever
increasing demand of portable electronics and devices which need to be deployed in
environments where long battery life is of primary importance.

7.2 Future Directions

The implementation of the working prototype and the synthesis flow is com-
plete in the sense of integrability with the current industrial synthesis tools. But
we believe that automatic CG/PG integration requires more investigation. There
are several aspects where the future improvements may be performed. These will
help in reducing the total run-time and memory usage, which will make the flow
applicable to larger circuits. The list is as follows,

1. The proposed flow uses row based power gating techniques[67]. This technique
is optimum for sufficiently large designs. To make the flow work efficiently for
smaller designs as well, other coarser power gating techniques such as block or
cell based [62, 75] may be more convenient.

2. Some improvements can be done to estimate the reactivation energy and tim-
ing costs more precisely during the power-gating[62, 18, 60]. In this approach
power gating will be applied only to the clock gated registers. During the
experiments we have also observed the number of non-power gated cells which
is significantly large. This directly depends on the current estimation method-
ology which forces the registers not to be power gated for optimum savings.
Hence, improvements in estimation methodology will yield more improvements
in the final leakage savings. This will also help in the maximum exploitation of the idleness of the clock gating.

3. As mentioned earlier, sizes and compositions of the clusters may not be optimal anymore since gate grouping is constrained by the existing CG structure. This has implications on the choice of the sizes of the sleep transistors. The size of the sleep transistor is a prime factor to reduce the area overheads observed in current implementations. A better sleep transistor sizing and insertion technique as proposed in [22, 58] will make the tool more realistic. Integrating completely different approach like [75] may help the flow to be more ideal and effective under certain conditions.

4. The identification of the critical path in the circuit and utilizing its attributes is always tricky. The current methodology, uses a generalized commercial tool to collect the information regarding the critical path and finally provides the information regarding the critical factor defined in this thesis. There is a possibility to improvise this net criticality factor by using more specific and efficient tools and techniques to extract more accurate information of the criticality at each and every net. There is also possibility to mathematically model this criticality factor to make it more accurate.

5. In the current implementation the graph partitioning process is very traditional. There are several other modern and efficient graph partitioning heuristics available [38, 76]. In [49], a multilevel partitioning process is proposed and it can be paralleled. This process is extremely efficient in terms of execution time for larger graphs and hence larger designs.

6. One of the key improvements required is that the current implementation does not work properly with the larger designs driven by multiple clocks. The problem with such designs is that in these designs more than one clock gating registers are enabled by the same net. Some modification in the analysis part of the design will eliminate this current limitation.
Appendix A

Leakage Power Components

“In order to make an apple pie from scratch, you must first create the universe.”
Carl Sagan

This appendix introduces several leakage current components contributing in the static power on a CMOS.

As shown in figure A.1 in a CMOS total 4 components of the currents contribute to the leakage power.

1. Reverse Bias Junction Leakage ($I_{REV}$) : $I1$: 
2. Sub-threshold Leakage ($I_{SUB}$) : $I2$: 
3. Gate Leakage ($I_{GATE}$) : $I3$: 
4. Gate Induced Drain Leakage ($I_{GIDL}$) : $I4$: 

![Figure A.1: Leakage Currents in Bulk CMOS][34]

A.1 PN Junction Reverse-Bias Current

This component is indicated by I1 in the figure A.1. To minimize the effect of barrier lowering it is essential to increase the channel doping at source-body and
A Leakage Power Components

Figure A.2: Leakage Currents in a CMOS Inverter in ON and OFF State [54].

Drain-body junctions along with the scaling which causes the reduction in channel length. This doping is also known as halo doping. This increase doping near the junctions will increase the junction tunneling leakage. It is well known that the reverse biased junctions that have heavy doping on both sides result in direct tunneling across these junctions. This tunneling phenomenon is known as band-to-band tunneling (BTBT). Band to band to tunneling takes place when a significant amount of current flows through the junction due to tunneling of electrons from the valance band of p type to the conduction band of the n type due to hight electric field across the reverse biased p-n junction as shown in the figure A.3

Figure A.3: BTBT in Reverse Bias [60].

A.2 Sub-Threshold Leakage

When the gate voltage is below the $V_{th}$, weak inversion current flows in MOS devices occurs. It is also known as sub-threshold current. It is shown as $I_2$ in figure A.1.

The figure A.4 explains the weak inversion phenomenon, the linear region with the small minority concentration, of the curve referring to the semilog plot, is the weak inversion region. It is important to model and implement methods to control
this leakage in today’s technologies as weak inversion dominates in the idle state of deep submicron devices. Unlike the strong inversion region where the drift current dominates, the sub-threshold conduction is dominated by the diffusion current. The exponential dependence between driving voltage on the gate and the drain current is a straight line in a semi-log plot of $I_d$ vs $V_g$. Sub-threshold leakage depends on various factors and short channel effects further increase this component of the leakage current. This leads to reduced threshold voltage and hence exponential increase in sub-threshold leakage current.

Figure A.4: MOSFET current to Gate Voltage in Logarithmic (Left) and Normal Scales (Right) [17].

*Effect of Operating Temperature:*
Temperature dependence of the sub-threshold leakage current is important since digital circuits usually operate at elevated temperatures due to power dissipation in the circuit. $\log(I_d)$ versus $V_g$ shows a linear change in sub-threshold slope with temperature. Two parameters increase the sub-threshold leakage as temperature is raised.

1. Linear increment of the slope of the sub-threshold $\log(I_d)$ versus $V_g$ curve, with temperature.

2. Decrease in the $V_{th}$.

Hence, it is clear that the sub-threshold leakage current varies exponentially with respect to temperature. This makes it dominant considering the working temperatures of modern processors.

*Punch Through:*
This phenomenon can be observed in short channel devices. In such devices depletion regions at the drain-substrate and source-substrate extend into the channel resulting in an increase in reverse bias across the junctions which pushes the junctions closer. Now, punch through occurs when the combination of channel length and the reverse bias will merge the depletion regions. This results in increased sub-threshold current. Furthermore, punch-through degrades the sub-threshold slope. One way of controlling this punch through effect is to use additional implants.

A.3 Tunneling into and Through Gate Oxide and Hot Carrier Injection from Substrate to Gate Oxide

This is another implication of the tunneling effect. Tunneling into and through gate oxide reduction of gate oxide thickness results in an increase in the electric field across the oxide. The gate oxide tunneling current shown as I3 in figure A.1 occurs when the high electric field is coupled with the oxide thickness resulting in substrate to gate tunneling of the electrons through the gate oxide. When a positive bias is applied to the gate of a MOS transistor, due to the small oxide thickness, which results in a small width of the potential barrier, the electrons at the strongly inverted surface can possibly tunnel into or through the $SiO_2$ layer and hence gives rise to the gate current.

Injection of Hot carriers from substrate to Gate Oxide

Hot-carrier injection is an effect mostly seen in short channel devices when due to high electric field near the $Si - SiO_2$ interface, electrons or holes can gain sufficient energy from the electric field to cross the interface potential barrier and enter into the oxide layer. Electrons are more prone to this effect than the holes.

A.4 Gate-Induced Drain Leakage - GIDL

For deep-submicron CMOS devices, $I_{GIDL}$ is the most dominant leakage-current component and must be minimized. When the gate is biased to cause an accumulation layer to form at the silicon surface, the silicon surface under the gate has about the same potential as the p-type substrate. The depletion region at the surface becomes narrower compared to any other regions because of the accumulation of holes at the surface, it behaves like a heavily doped p-region than the substrate. This narrowing of the depletion layer at or near the intersection of the p-n junction and the $Si - SiO_2$ interface causes denser fields. When the negative bias is large enough, the n+ region under the gate can become depleted, and even inverted. In this case, the gate and the n+ region behave like a MOS capacitor with a heavily doped n-type "substrate". The field density increases even further, and hence, the peak field increases. As the electric field in and around the gated p-n junction
increases, by the gate voltage, all the high field effects, such as avalanche multiplication and band-to-band tunneling may increase. When it occurs in the drain junction of a MOSFET, the increased junction leakage current is called gate-induced drain leakage or GIDL shown as I4 in figure A.1.
Appendix B

State of the Art : Leakage
Power Reduction Mechanisms

“"A youth who had began to read geometry with Euclid, when he had learnt the first proposition, inquired, "What do I get by learning these things?" So Euclid called a slave and said "Give him three pence, since he must make a gain out of what he learns."." 

Euclid

This appendix introduces several leakage power reduction techniques adapted by the designers. There have been numerous techniques devised to reduce this leakage current component ranging from device level solutions to circuit level techniques to architecture level leakage management policies.

B.1 Design Time Techniques

B.1.1 Dual $V_{th}$

The main idea of this technique is to separate the transistors lying on the critical path and non-critical path by assigning low $V_{th}$ and high $V_{th}$ respectively, to match with the desired timing constraints for critical path and to match the power constraints by limiting the sub-threshold leakage for non-critical path. The technique is particularly attractive as its implementation is feasible with existing dual-Vth MOSFET process, making it ideally suited to achieve high performance and low power simultaneously. Dual-Vth technique helps in reducing active and stand-by leakage power[62].

It is static technique that means only two threshold voltages available in the technology are used, however there are techniques which proposes dynamic variations in $V_{th}$ by biasing the body of the transistors. This technique achieves both high performance and low leakage simultaneously, costing zero overhead in terms of
additional circuits.

Figure B.1: Conceptual Network with Critical and Non-critical Cells Replaced by Low and High $V_{th}$ Cells.

Figure B.1 shows an example gate network to which dual-$V_{th}$ technique is applied. The synthesis step will take care of the replacement of the gates on non-critical paths (grey colored) with the high-$V_{th}$ cells from the standard cell libraries.

Many design modifications in the standard technique have been proposed. One of the famous technique is to increase the size of the high $V_{th}$ transistor in dual $V_{th}$ design to improve performance\cite{57} at the cost of switching power and die area that can be traded off against using a low $V_{th}$ transistor that increases leakage power.

A typical dual $V_{th}$ domino logic\cite{70} for low leakage and noise free operations is shown in figure B.2 as domino logic is easily influenced by the leakage power and noise. In domino logic the transition directions are fixed and hence it is very easy to allocate low $V_{th}$ to all the transistor switching in evaluate mode and high $V_{th}$ to the ones that made transition in precharge mode. As it is clear from the figure that in stand by mode the driving clock to the domino logic must be high to turn off the high $V_{th}$ components P1, I2 PMOS and I3 NMOS of the figure. It is also necessary to maintain the internal nodes at constant absolute NULL voltage, and hence the initial inputs to the domino gate must be set to "1" so that the high $V_{th}$ keeper and I1 NMOS should be turned off.

So far, the techniques discussed represents the changes in the channel doping
profiles, instead of this it is also possible to adapt higher $t_{ox}$ and obtain a high $V_{th}$ device for dual $V_{th}$ circuits. Multiple $t_{ox}$ CMOS (MoxCMOS) can optimize the power consumption due to subthreshold leakage, gate oxide tunneling leakage as well as switching power.

### B.1.2 Multiple Supply Voltage

In multiple power supplies $V_1 > V_2 > V_3.. > V_n$ power dissipation is given by

$$P_n = f \left\{ \left( C_1 - \sum_{i=2}^{n} C_i \right) V_1^2 + \sum_{i=2}^{n} C_i V_i \right\}$$ (B.1)

where, $C_i$ is total capacitance of circuits and interconnections that will operate under $V_i$ and $f$ is an operating frequency.

A rough rule of thumb for optimum $V_{DD}$ is derived\[53],

For $\{V_1, V_2\}$, $\frac{V_2}{V_1} = 0.5 + 0.5 \frac{V_T}{V_1}$ (B.2)

For $\{V_1, V_2, V_3\}$, $\frac{V_2}{V_1} = \frac{V_3}{V_2} = 0.6 + 0.4 \frac{V_T}{V_1}$ (B.3)

For $\{V_1, V_2, V_3, V_4\}$, $\frac{V_2}{V_1} = \frac{V_3}{V_2} = \frac{V_4}{V_3} = 0.7 + 0.3 \frac{V_T}{V_1}$ (B.4)

The figure B.3 proves, that this rule of thumb gives almost optimum under which power is reduced to the point that is within 1% difference from the precise minimum. It is also understood from figure that the more number of available voltages the less power, but the effect will be saturated. The power reduction effect will also be diminished as the power supply voltage is scaled and as is higher. A better approximation would be,

![Figure B.3: Power Dissipation Ratio in a Configuration with 3 $V_{DD}$ with $V_1 = 1.5V$ and $V_t = 0.3V\[17]](image)

$$\frac{P_n}{P_1} = 0.3 + 0.7 \frac{V_T}{V_1}$$ (B.5)
B.2 Run Time Standby Leakage Reduction Techniques

B.2.1 Transistor Stacking

Stacking Effect: It can be defined as when the sub-threshold current flows through the series of the transistors, eventually the current will reduced when more than one transistors in the stack is turned off.

![Figure B.4: Stacking Effect in Two NMOS.](image)

Figure B.4 shows a two transistor NMOS stack. Because of the stack effect, when all the transistors in the network are given "0" as the input, the drain voltages as it can be seen in the figure, goes up and this will result in reduced leakage power in stand by mode. This reduction is due to the following reasons.

1. As shown in the figure due to the voltage $V_x$, the gate-source voltage $V_{gs}$ of the upper transistor goes negative reducing the sub-threshold leakage drastically.

2. Due to the source voltage of the Y transistor being higher than its body voltage, the threshold voltage of this transistor increases due to body effect and hence further reducing the sub-threshold current.

3. Since $V_x > 0$, the drain-source voltage of the upper transistor is reduced and hence threshold voltage of this transistor increases and hence suppressing the sub-threshold current.

This effect is also called *self-reverse biasing* of transistor. The self-reverse bias effect can be achieved by turning off a stack of transistors. The leakage of a 2 transistor stack is an order of magnitude lower than that of a single transistor. There are several ways of exploiting the stacking factor to reduce the leakage.
B.2 Run Time Standby Leakage Reduction Techniques

B.2.2 Sleep Transistor Insertion

The sleep transistor insertion method is well described in chapter 2 and chapter 3.

B.2.3 VTCMOS

VTCMOS-Variable Threshold voltage CMOS method controls the leakage by the means of substrate bias control. Figure B.5 shows the concept of the VTCMOS technology. VTCMOS technology sets the leakage current below 10mA in active mode and below 10µA during standby independently from the processed $V_T$. The analytical model, device design, and scaling scenario for VTCMOS technology are found in [26].

![Figure B.5: VTCMOS](image)

Figure B.6 shows the leakage current variations in a DSP application chip which adapts VTCMOS technology [29].

Some of the enhancements in the traditional VTCMOS techniques are enlisted as below,

1. To achieve different threshold voltages, a self-substrate bias circuit is used to control the body bias. In the active mode, a zero body bias is applied. While in standby mode, an intense reverse body bias is applied to increase the $V_{th}$ and cut off the leakage current. [6, 29, 27]

2. Forward substrate bias can be used to increase the circuit speed in operational mode resulting in reduced short channel effects. The reverse bias, helps reducing the circuit leakage by 3 times of the magnitude for $0.35 - \mu m$ technology nodes at the cost of little area overhead caused by the body bias grid [56, 71].

3. Due to the exponential increase in band-to-band tunneling leakage at the source/substrate and drain/substrate p-n junctions due to halo doping in deep
submicron technologies, the effectiveness of reverse body bias to reduce the standby current decreases. The possible solution for this problem is to adapt forward body bias along with reverse body bias to achieve better current drive with less short channel effects [28].

B.3 Active Leakage Reduction

B.3.1 Dynamic Voltage Scaling - DVS

Dynamic supply voltage reduces the cost by adapting the concept of producing more than one supply voltage from a single source of voltage unlike multiple voltage source technique. And this dynamic variation depends on the performance requirements. The highest supply voltage delivers the highest performance at the fastest designed frequency of operation. When performance demand is low, supply voltage and clock frequency are lowered, just delivering the required performance with substantial power reduction. [64] DVS is very easy to implement and it is very economical, implementing DVS in a general-purpose microprocessor system includes three key components:

1. An operating system that can intelligently vary the processor speed.

2. A regulation loop that can generate the minimum voltage required for the desired speed.

3. A microprocessor that can operate over a wide voltage range.
B.3 Active Leakage Reduction

B.3.2 Dynamic $V_{th}$ Scaling - DVTS

In sub-micron generations where leakage power accounts for a large fraction of the total power consumption even in the active mode, a method similar to the dynamic $V_{DD}$ scaling (DVS) scheme, can be used to reduce the active leakage power. This method adapts dynamic scaling of $V_{th}$.

During the period of relaxed workload a lower clock frequency can be assigned to the operating system running on the system. Just like DVS based upon the lower bound of the operating frequency the DVTS increases the $V_{th}$ by varying the reverse body bias to reduce the runtime leakage power dissipation\[19\]. In cases when there is no workload at all, the $V_{th}$ can be increased to its upper limit using appropriate variations in body biasing, to significantly reduce the standby leakage power\[31\]. It is also possible to exploit the optimal $V_{th}$ to deliver only required amount of throughput, this will result in very low leakage but sluggish circuit behavior.

![Figure B.7: Total Power Compared with Clock Frequency in DVS and DVTS\[60\].](image)

The figure\[B.7\] compares the two techniques by plotting a graph of total power consumption against the operating frequency. Foremost observation says that total power reduces linearly with decreasing operating frequency as the dynamic power linearly varies with frequency. Leakage power does not depend on the frequency as at "0" frequency almost 52% of the total power (mainly leakage power) is wasted. At this technology (70 nm) DVTS achieves similar results as DVS. And at the 30% of the maximum operation frequency, 92% total energy savings can be achieved using DVTS.

Some points worth mentioning for DVTS are as follows,

1. Adapts very simple hardware. To boost the voltages when the current require-
ment is low charge inserters (they are used to generate the body bias voltages, which are outside the supply rail) can insert the necessary charges. No external inductors are needed and hence, power consumption is very low compared to the mechanisms used for DVS systems \cite{39, 25}.

2. Transition energy overhead. Results VTCMOS demonstrate that the energy overhead for a 120K-transistor test chip in 0.3\(\mu\)m triple well technology consumes 10 nJ per \(V_{th}\) transition. In case the \(V_{th}\) transition occurs frequently, transition energy overhead for DVTS systems becomes non negligible \cite{51}.

3. This method is prone to increased substrate noise. In the absence of external inductors, charge pumps generate an unregulated body bias voltage and any fluctuation in body bias will easily induce noise in logic.

4. As there is a specific requirement of independent body biases for PMOS and NMOS from the rest of the system to have a clear reference in DVTS controllers, the fabrication process demands complex methods. The overall cost penalty by using these advanced processes is less than 5\% \cite{51}.
Appendix C

Proof of Minimum k-cut Theorem

"No human investigation can be called real science if it cannot be demonstrated mathematically."
Leonardo da Vinci

This appendix provides the proof of the minimum k-cut problem which is to support the arguments discussed in chapter 5. In the first section a concept of the Gomory-Hu trees is explained which is required in the proof of the minimum k-cut theorem[76].

C.1 Gomory-Hu Tree

A Gomory-Hu tree for $G = (V, E)$ is a tree $T$ on the same set of vertices $V$. The edges of $T$ are not necessarily in the set $E$ and have a new weight function $w'$ associated with them. In addition, each edge $e \in T$ which partitions $T$ into components $S$ and $(T - S)$ is said to represent the cut associated with separating $S$ and $(T - S)$ in $G$.

The conditions for a Gomory-Hu tree are as follows:

1. For every pair of vertices $u$ and $v$, the weight of the minimum cut between $u$ and $v$ is the same in both $G$ and $T$.

2. For each edge $e \in T$, $w'(e)$ is the weight of the cut represented by $e$ in $G$.

We can compute the Gomory-Hu tree on a graph $G$ by maintaining a tree $T$ on a collection of vertex sets $S = \{S_1, S_2, ..., S_t\}$, beginning with the single set $S_1 = V$. At each step, we take one of these sets $S_i$ for which $|S_i| > 1$, and choose $u, v \in S_i$. 
After rooting $T$ at $S_i$, we create a new graph by starting with $G$ and then collapsing each subtree of $S_i$ into a single supernode. We then run a minimum cut between $u$ and $v$ in this new graph, dividing $V$ into $V_1$ containing $u$ and $V_2$ containing $v$. Our graph $T$ is now modified by breaking $S_i$ into $S_{i1} = S_i \cap V_1$ and $S_{i2} = S_i \cap V_2$, with an edge between them representing the minimum cut we just calculated. As for the subtrees of $S_i$, we connect a subtree to $S_{i1}$ if its supernode was in the same partition as $u$ in the minimum cut, and connect it to $S_{i2}$ otherwise. Performing this step $k - 1$ times will result in a tree in which each node in $T$ represents a single vertex from $G$. This final tree will satisfy the properties of the Gomory-Hu tree.

### C.2 Minimum k-cut

We will achieve our approximation for minimum k-cut using $T$, the Gomory-Hu tree for $G$, with the associated weight function $w'$. Our approach will be to take the $k - 1$ lightest cuts from the $n - 1$ cuts associated with the edges in $T$. We can prove that removing these cuts from $G$ will leave us with at least $k$ connected components.

**Theorem:** Removing $l$ cuts associated with $l$ edges in the Gomory-Hu tree for $G$ results in a new graph $G'$ with at least $l + 1$ connected components.

**Proof:** Let $V_1, V_2, ..., V_{l+1}$ be the connected components which are left in $T$ after the removal of the $l$ edges. For any $u \in V_i$ and $v \in V_j$ with $i \neq j$, we must have removed some edge in $T$ which disconnects $u$ and $v$. Recall that this edge has a cut associated with it in $G$ which must disconnect $u$ and $v$ in $G$ as well. Thus removing the edges in $T$ must disconnect each set $V_i$ from each of the other sets $V_j$, and it follows that there are at least $l+1$ connected components in $G$.

Now as with our proof for the multiway cut, let $A$ be the optimal k-cut for $G$, let $V_1, V_2, ..., V_k$ be the components which $A$ divides $V$ into, and for each $V_i$ let $A_i$ be the set of edges between $V_i$ and $V - V_i$. Thus each edge in $A$ is contained in two of the $A_i$ sets, and again we have

$$\sum_{i=1}^{k} w(A_i) = 2w(A)$$ (C.1)

Consider modifying $T$ by shrinking the vertices corresponding to each $V_i$ into a supernode, then removing edges until the graph becomes a new tree $T'$. Assume without loss of generality that $A_k$ has the highest weight of all the cuts, and root $T'$ at the the supernode $V_k$.

Now we can put every supernode $V_i$ (except for $V_k$) in correspondence with the edge between $V_i$ and its parent. Let this edge be $(u_i, v_i)$ for some supernode $V_i$. Recall that since the edge $(u_i, v_i)$ was in $T$, it represents the minimum cut between $u_i$ and $v_i$. Therefore, since $A_i$ represents a cut between $u_i$ and $v_i$, we must
have $w(A_i) \geq w'(u_i, v_i)$.

And hence,

\[ \sum_{i=1}^{k-1} w'(u_i, v_i) \leq w(A_i) \leq 2 \left( \frac{k-1}{k} \right) w'(A) = \left( 2 - \frac{2}{k} \right) w(A) \quad \text{(C.2)} \]

Note that $(u_i, v_i)$ are arbitrary edges in $T$, so $\sum_{i=1}^{k-1} w'(u_i, v_i)$ is at least as large as the sum of the $k - 1$ lightest edges in $T$ and our proof is complete.
Appendix D

User Guide to the Prototype CGPG Tool

“To accomplish great things, we must not only act, but also dream; not only plan, but also believe.”
Anatole France

This appendix provides the necessary information for successful execution of the developed tool. It explains the necessary environment variables to be set to interface the tool with the commercially available synthesis tools and final place and route tools.

D.1 Using the CGPG tool

The tool is composed of the following files:

1. First part (tool_analysis): read_design.cpp (main), read_design.h, analisi_design.cpp and analisi_design.h

2. Second part (tool_clustering): clustering_design.cpp and clustering_design.h

It is important to note that the tool has been divided in two main blocks namely:

1. Analysis and

2. Cluster and Bound Generation.

During the execution of the first part, the tool is able to read and extract the physical information of the design. In more details, the tool identifies the clock tree, the activation function of the clock-gated registers (CGregister), the fanout associated to every CGregister, the set of all cells connected to more than one fanout and as well computes the virtual ground and the leakage power associated to every CGregister. Moreover, the tool generates two main structures containing the
information necessary to perform the optimization and the bound generation of the design.

Finally, during the second part, the tool calculates the cost-function and generates the graph to be partitioned. Lately, in the same phase, using different heuristics the tool tries to merge the initial clusters, identified during the first part, and performs the generation of the bounds according to the optimization parameters given.

### D.2 Compiling the CGPG tool

The tool is successfully compiled under Unix environment (Linux) with g++v3.3. However it is easy to modify the environment variables by changing the macro.def file in the path /home/upasani/tool_analysis or home/upasani/tool_clustering. Then after enter into the folder where you store the source files of the tool and execute the following command line:

```
setenv OA_DIR /tools/OpenAccess-p052/ in order to set the properly OpenAccess database library.
```

Finally, enter the command `make`. It is also possible to synthesize and place separately by commands `make synthesis` and `make place`. It is important to note that this command is defined by the file Makefile that contains the instructions necessary to compile the tool properly.

### D.3 Configuring the Tool

It is worth mentioning that before running the tool, it is necessary to synthesize as well as to perform an initial placement of the design using the Physical Compiler (Synopsys) using the scripts listed in the previous section of this user guide. The synthesis is made using the 65nm technology libraries from ST Microelectronics.

```
Synthesis and Initial Placement of the Design:
/home/upasani/CGPG_workflow/scripts/synthesis.tcl
```

```
OpenAccess Generation:
/home/upasani/CGPG_workflow/scripts/importOA.tcl
```

The OpenAccess database is generated from the synthesized design using the files .v and .def saved after the synthesis by Physical Compiler. These files are stored into a folder named `saved` that has been automatically created during the synthesis process.

The OpenAccess database is saved into the following folder named `OAlibs` inside `/home/upasani/CGPG_workflow/`. 
Thus, to generate the OpenAccess database it is necessary to execute the following command line from directory home/upasani/CGPG_workflow/:

```
make importOA
```

It is important to highlight that for generating the OpenAccess database it is necessary that the file lib.defs is empty. Afterwards, it is necessary to generate the file .vcd through a simulation of the synthesized design using Modelsim by Mentor Graphics. The .vcd has to be generated using the following commands in Modelsim in the following order:

```
vcd file vcdfile.txt
vcd add -r
run 100000 ns
vcd checkpoint
restart -f
```

The output file has to be copied into the folder ../../../OAlibs/ together with the technology libraries

CORE65LPHVT_cvgnd.dat, CORE65LPSVT_cvgnd.dat, CORE65LPLVT_cvgnd.dat,
CORE65LPHVT_leakage.txt, CORE65LPSVT_leakage.txt, CORE65LPLVT_leakage.txt

These libraries can be found in the folder /home/upasani/CGPG_workflow/OAlibraries.

### D.4 Tool Execution

Enter into the folder /home/upasani/CGPG_workflow/OAlibs/ and execute the following commands:

```
setenv LD_LIBRARY_PATH /tools/OpenAccess-p052/lib/linux_rhel21_32/opt
```

```
../../tool.CGPG/tool OA_switch_6x6_2_6 switch_6x6_2_6 layout
```

Where, tool is the name of the executable file of the first part, OA_switch_6x6_2_6 is the folder of the OpenAccess database, the switch_6x6_2_6 is the name of the design’s top level and layout is the type of view. This is the execution of the first part. After this step there will be files generated which are ,

1. File, output_designPlacement.txt contains the information regarding the physical placement.
2. output_tree.txt contains the extracted clock tree of the design.
3. output_fanout.txt keeps the information of the fanout for each and every cell.
4. output_analysis.txt provides the feasibility analysis.
5. output_clusters.txt stores the information regarding the initial clusters and the information like standard cells in each cluster, leakage of the clusters, total idle period of the clusters, virtual ground capacitance etc.

6. output_FA.txt file contains the information regarding the activation function.

Out of all these keep output_FA.txt and the output_clusters.txt files in the same folder (Do not move them as this information will be used by the second phase.) Enter into the folder /home/upasani/CGPG_workflow/OAlibs/ and execute the following commands:

```bash
csetenv LD_LIBRARY_PATH /tools/OpenAccess-p052/lib/linux_rhel21_32/opt
and
../../tool.CGPG/clustering OA_switch_6x6_2_6 switch_6x6_2_6 layout
```

Where, clustering is the name of the executable file of the second phase,

The output files of the second phase,

1. output_attraction.txt file contains the cost function matrix measuring the attraction among the pair of clusters.

2. output_bounds.txt contains the pre-placed bounds, it contains the list of standard cells, total leakage, total propagation delay, virtual ground capacitance and minimum idle period for each bound.

3. output_edges.txt stores the information of the edge weights.
Appendix E

Tools Used

“If I had eight hours to chop down a tree, I’d spend six hours sharpening my axe.”

Abraham Lincoln

This appendix describes the tools used for the experiments and simulations during the work of the thesis. It also describes some of the important commands used during the simulations.

E.1 Synopsys Physical Compiler

Physical Compiler© provided by Synopsys™ is used for the synthesis and the physical placement and routing of the design. The main advantage of this tool is that it is very accurate and recognized by the EDA industries. Several commands are used for efficient synthesis and placement of the design. The entire process is automated by putting the necessary commands to invoke synthesis and desired placement and routing process in the form of TCL scripts. The simulator takes the benchmark RTL design in Verilog or VHDL as the input and gives the gate-level synthesized files in VHDL and Verilog[45].

E.1.1 Commands for Gate Level Synthesis

Analyze command translates the specified HDL files and stores the intermediate format in the specified library.

Elaborate Builds a design from the intermediate format of a Verilog module, a VHDL entity and architecture, or a VHDL configuration.

The insert_clock_gating command will perform clock gating at all levels of hierarchy in the design.

Ungroup Removes a single level of hierarchy from the current design by exploding the contents of the specified cell or cell instance in the current design.

The physopt command performs simultaneous placement and synthesis on the current design and generates a legalized placed netlist.
The `link_physical_library` command links the physical library with the logic libraries for a Physical Compiler session, linking library cells and pins by name.

### E.1.2 Commands for Physical Placement and Routing

The `create_placement` command performs coarse placement on the current design. As per the placement `effort` specified.

The `legalize_placement` command performs detailed legalized placement on the current design after coarse placement has been done.

The `set_mpc_options` command defines design-level constraints used by the `create_placement -mpc` and `physopt -mpc` commands to automatically create a coarse floorplan for placement prototyping.

The `check_mpc` command checks the user-specified or default `set_mpc_*` design-level physical constraints against a placed design database generated in a placement prototyping flow. The resulting report lists the constraints and whether they are met or not met and why.

### E.2 Modelsim

Modelsim is a design simulator and HDL editor provided by Mentor Graphics. It is used to simulate the synthesized gate-level designs of the benchmarks. Simulations are performed using the test-benches provided along with the benchmarks. The switching activities at each net is stored in a file (Value Change Dump-.VCD) for further process.

The `vcd file` command specifies the filename and state mapping for the VCD file created by a `vcd add` command.

The `vcd add` command adds the specified objects to a VCD file. The allowed objects are Verilog nets and variables and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored).

The `vcd checkpoint` command dumps the current values of all VCD variables to the specified VCD file. While simulating, only value changes are dumped.

The `run` command advances the simulation by the specified number of timesteps.

### E.3 OpenAccess API

OpenAccess provides an infrastructure designed to promote the interoperability of EDA applications and design data. The infrastructure is a C++ API that defines classes and member functions to create, access, and manage databases that describe designs as they evolve through the design process to implementation. The OpenAccess API is structured into a set of packages, each of which provide the implementation of part of the total functionality. Each package has one top level...
header file that a program must include to access the package classes and functions. Each package is distributed in the form of a shared library that contains the implementation of its functions. An application must link against the shared library to call those functions. The API is also used to generate the database of the used 65nm technology libraries provided by STMicroelectronics™.

A brief overview of some of the classes whose attributes are used [55].

An **oaDesign** is a database that holds all the design data that describes a part of a design. All netlists, schematics, layouts and other design representations live as a set of oaDesigns in OpenAccess. It is a container for the connectivity, geometry, hierarchy, parasitic, and floorplanning information about a design.

The **oaInst** class is an abstract base class for block domain instances. An instance represents the inclusion of one design as a part of the contents of another. The design containing the instance is the parent design and the design that is included is the master of the instance.

The **oaNet** class is an abstract base class for scalar and multi-bit nets that are part of the design’s physical description. Nets represent the logical connectivity within a block of a design. Nets connect to terminals, which are the logical connection points on instances of the net’s block. Nets also connect to instTerms that represent connections to the lower-level instances in the net’s block.

The **oaTerm** class is an abstract base class for all scalar and multi-bit terminals. Terminals represent the logical connection points for a block. The pins associated with terminals represent the physical connection points. The nets associated with the terminals are logically exported through the terminals to the next higher level in a design hierarchy. All terminals are required to have a net even if there is nothing else attached to that net.

A quick glance of some functions used,

```c
oaLib* getLib() returns the library name using the specified nameSpace.
oaBlock* getTopBlock() returns the block in this design. It will return null if there is no block in this design. Note that the name of this call reflects the future possibility of having an embedded block hierarchy in a single design. At this time there can only be one block in a design.
oaModule* getTopModule() returns the top module of the module hierarchy in this design. It will return null if there are no modules in this design, or if none of the existing modules have been set to be the top.
void getCellName() returns the cellName using the specified nameSpace.
oaCellType() This function returns the cellType for this design.
```
Appendix F

Source Code

"Two things are infinite: the universe and human stupidity; and I'm not sure about the universe."
Albert Einstein

This appendix contains the main file of the two parts of the tool. In section F.1 the main file of the analysis phase is given. In the section F.2 the main file of the clustering part is shown.

F.1 Analysis Phase

```cpp
int main(int argc, char *argv[]) {
    // Variable declaration
    oaDesignInit();
    oaScalarName libName;
    oaScalarName cellName;
    oaScalarName viewName;
    oaScalarName techNameSVT = oaScalarName(oaNS, "CORE65LPSVT");
    //Technologic library SVT
    oaScalarName techNameLVT = oaScalarName(oaNS, "CORE65LPLVT");
    //Technologic library LVT
    oaScalarName techNameHVT = oaScalarName(oaNS, "CORE65LPHVT");
    //Technologic library HVT
    oaBoolean usage = false;
    oaBoolean traverseHier = false;
    // Opening the output files
    out_designPlacement.open(filename_designPlacement, fstream::out);
    out_tree.open(filename_tree, fstream::out);
    out_fa.open(filename_fa, fstream::out);
```
out_fanout.open(filename_fanout, ofstream::out);
out_test.open(filename_test, ofstream::out);
out_reusedcells.open(filename_reusedcells, ofstream::out);
out_clusters.open(filename_clusters, ofstream::out);

// Reading information from command line
parseArgs(argc, argv, usage, traverseHier, libName, cellName, viewName);
if (usage) {
    printUsage();
    exit(1);
}

// Controlling the design
try {
oaString libPath, techPathSVT, techPathLVT, techPathHVT;
libName.get(libPath);
technNameSVT.get(techPathSVT);
technNameLVT.get(techPathLVT);
technNameHVT.get(techPathHVT);
if (oaLib::exists(libPath)) {
    if (!oaLib::find(libName)) {
        oaLib::open(libName, libPath);
    }
} else {
    cout << "Can not find the library " << libPath << endl;
    exit(1);
}

// Opening the SVT library
if (oaLib::exists(techPathSVT)) {
    if (!oaLib::find(technNameSVT)) {
        oaLib::open(technNameSVT, techPathSVT,
                    oacReadOnlyLibMode);
    }
} else {
    cout << "Can not find the library " << techPathSVT << endl;
    exit(1);
}

// Opening the LVT library
if (oaLib::exists(techPathLVT)) {
    if (!oaLib::find(technNameLVT)) {
        oaLib::open(technNameLVT, techPathLVT,
                    oacReadOnlyLibMode);
    }
} else {
    cout << "Can not find the library " << techPathLVT << endl;
    exit(1);
}
// Opening the HVT library
if (oaLib::exists(techPathHVT)) {
  if (oaLib::find(techNameHVT)) {
    oaLib::open(techNameHVT, techPathHVT,
                oacReadOnlyLibMode);
  }
} else {
  cout << "Can not find the library " << techPathHVT << endl;
  exit(1);
}

oaDesign *view = oaDesign::open(libName, cellName, viewName, 'r');
// Reading the virtual ground
vgnd_read();
// Reading the leakage
leakage_read();
// Starting
cout << " Processing ... " << endl;
processing(view);
view->close();
} catch (oaException &excp) {
  cout << "Error: " << excp.getMsg() << endl;
  exit(1);
}
return 0;

F.2 Clustering Part

// -------------------------------
// MAIN
// -------------------------------

int main(int argc, char *argv[]) {
  string number_bounds;
  int i;
  int n, p, k;
  int magic;
  clock_t start = clock();
  out_attraction.open(filename_attraction, fstream::out);
  out_edges.open(filename_edges, fstream::out);
  out_prebounds.open(filename_prebounds, fstream::out);
  out_bounds.open(filename_bounds, fstream::out);
  out_design.open(filename_design_layout, fstream::out)
oaDesignInit( oacAPIMajorRevNumber,
              oacAPIMinorRevNumber,
              oacDataModelRevNumber );

  // Check number of arguments.
  if ( argc < 4 ) {
    printf(" Use : \n\toadump libraryName cellName viewName\n");
    exit(1);
  }

  char *cvType = NULL;

  // If the 4th argument was supplied, it is the DesignType.
  if ( argc == 5 ) cvType = argv[4];

  // Get the names from the command line
  oaString libstr (argv[1]),
              cellstr(argv[2]),
              viewstr(argv[3]);

  // Treat the names on the command line as belonging to the Native
  // namespace and convert them to oaNames.
  oaNativeNS ns;
  oaDesign *design;
  oaLib *lib;

  const char *pathLibDefs = "/home/upasani/CGPGApril09/OAlibs/lib.defs" ;
  oaScalarName
  libName (ns, libstr ),
  cellName(ns, cellstr),
  viewName(ns, viewstr);

  // Parse the library definitions in the local lib.defs file.
  if ( !oaFile(pathLibDefs).exists() ) {
    printf( "***Must have \%s file.\n", pathLibDefs );
    return 1;
  }

  oaLibDefList::openLibs();

  if ( ! (lib = oaLib::find(libName)) ) {
    printf( "***Either Lib name=\%s or its path
    mapping in file=\%s is invalid.\n",
            (const char *)libstr.pathLibDefs );
    return 2;
  }

  try {
    if ( cvType ) design =
      oaDesign::open( libName, cellName, viewName,
                      oaViewType::get(oaString(cvType)),
                      MODE_READ);

    else design =
      oaDesign::open( libName, cellName, viewName, MODE_READ );
  }
F.2 Clustering Part

```c
catch (oaException &excp) {
    printf("Can’t read Design \%s/\%s/\%s (check lib.defs)\n", 
            (const char *)libstr, (const char *)cellstr, (const char *)viewstr);
    printf("t\%s\n", (const char *)excp.getMsg());
    exit(1);
}

printf( "Reading Design \%s/\%s/\%s\n", 
            (const char *) libstr,(const char *) cellstr,(const char *) viewstr );

try {
    oaBlock *block = design->getTopBlock();
    design->openHier(10000);
    cout << " Processing, Generating OpenAccess database." << endl;
    dumpInst_net (block);
    dumpNet (block);
    dumpShape(block, design);
    dumpRoute(block);
}

catch (oaException &excp) {
    printf("Error: \%s\n", (const char *)excp.getMsg());
    exit(1);
}

/∗ | Step 1: reading the file output_FA.txt that stores the information
about activation function of the CG registers
and storing these information into enableTable */

cont << " Begin the clustering." << endl;
cont << " Processing, step 1 " << endl;
cont << " Reading file output_FA.txt " << endl;
cont << " Generating the structure enableTable. " << endl;
read_inputfile_FA();

// | Step 2: generating the binaryTable from the enableTable
cont << " Processing, step 2. " << endl;
cont << " Generating the structure binaryTable. " << endl;
prefProcessing_binaryTable();

/∗ | Step 3: reading the file output_clusters.txt that stores
the information about the initial clusters and storing
these information into clusterTable structure */

cont << " Processing, step 3 " << endl;
cont << " Reading file output_clusters.txt " << endl;
cont << " Generating the structure clusterTable " << endl;
read_inputfile_clusters();

// | Step 4 : Generating the symbol table
cont << " Processing, step 4." << endl;
cont << " Generating the symboltable for activation nets " << endl;
generating_FAnet_symbolnet();

/∗ | Step 5: computing the idle periods of every cluster
```
and saving the information into idleTable. This information will be used to generate the graph in the successive step */
cout << " Processing, step 5 " << endl;
cout << " Computing the idle periods of every initial cluster " << endl;
idleperiods_generation();
/* |Step 6 : Read the OA database 
Generate the net−list */
cout << "Processing, step 6 " << endl;
cout << "Read the open access database" << endl;
cout << "Reading the output_design.txt file and generating the net−list " << endl;
read_inputfile_design_layout();
read_inputfile_net_criticality();
/* |Step 7 : Count the shared signals and the net−criticality */
cout << "Processing, step 7 " << endl;
cout << "generating shared signals between a pair of clusters." << endl;
signals_shared();
net_criticality();
/* |Step 8 : Compute the cost function */
cout << "Processing, step 8 " << endl;
cout << " Computing the costfunction and generating the graph" << endl;
graph_generation();
/* | Step 9: computing the atraction between every initial cluster and storing this values into structure Edge */
cout << " Processing, step 9 " << endl;
cout << " Generating the Edge structure containing the weights. " << endl;
edge_generation();
// | Step 10: Sorting the edges 
cout << " Processing, step 10" << endl;
cout << " Sorting the edges " << endl;
sort_edges();
// | Step 11: Clustering 
cout << " Processing, step 11 " << endl;
cout << " Begin Clustering. " << endl;
n=N; p=P; k=K;
magic = clustering(Graph, Edge, N, P, K);
// | Step 12: Printing bounds 
cout << " Processing, step 12 " << endl;
cout << " Printing bounds into file output_bounds.txt " << endl;
read_inputfile_prebounds( magic );
printf("Time : \%f\n", ((double)clock() − start )/2000000 );
return 0;}
\end{ANSI[C++]}


Bibliography


[55] OpenAccess. Openaccess v2.2 c++ api documentation.


