2008

Temperature and interconnect aware unified physical and high level synthesis

Vyas Krishnan
University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation
Krishnan, Vyas, "Temperature and interconnect aware unified physical and high level synthesis" (2008). Graduate Theses and Dissertations.
http://scholarcommons.usf.edu/etd/347

This Dissertation is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Temperature and Interconnect Aware Unified Physical and High Level Synthesis

by

Vyas Krishnan

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

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

Date of Approval: October 24, 2008

Keywords: behavioral synthesis, power-aware design, thermal analysis, interconnect-centric design, stochastic interconnect estimation, layout-aware synthesis

© Copyright 2008, Vyas Krishnan
ACKNOWLEDGEMENTS

I would like to express my gratitude to my major professor, Dr. Srinivas Katkoori, for his guidance, encouragement, and support throughout my doctoral degree program. I thank him immensely for providing me the opportunity to work in this interesting area. I would like to thank Prof. Nagarajan Ranganathan, Dr. Hao Zheng, Dr. Sanjukta Bhanja, and Dr. Greg McColm for being on my Ph.D. committee. I take this opportunity to acknowledge the intellectual and moral support provided by current and past members of the VCAPP lab at USF. I would like to acknowledge the help provided by the Computer Science Tech Support team led by Daniel Prieto. Most importantly, I would like to thank my family for steadfastly supporting me in all my endeavors.
Note to Reader: The original of this document contains color that is necessary for understanding the figures and data. The original dissertation is on file with the USF library in Tampa, Florida.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>iv</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
</tbody>
</table>

## CHAPTER 1  INTRODUCTION

1.1 Need for Interconnect-Awareness  3
1.2 Need for Temperature-Awareness   5
1.3 Need for Unifying Physical-Level and High-Level Synthesis  8
1.4 Contributions of the Dissertation  10
   1.4.1 Layout-Aware Resource Binding for 3-D Integrated Circuits  10
   1.4.2 Net Topology Aware Resource Binding for Wire Delay Minimization  11
   1.4.3 Stochastic Wirelength Estimation Based Design Space Exploration  12
   1.4.4 Temperature-Aware Layout-Driven High-Level Synthesis  14
   1.4.5 A Genetic Algorithm for High-Level Design Space Exploration  15
1.5 Organization of the Dissertation  16

## CHAPTER 2  BACKGROUND AND RELATED WORK

2.1 An Overview of High-Level Synthesis  17
2.2 An Overview of Physical Synthesis   20
2.3 Impact of Elevated Chip Temperatures on Performance and Reliability  23
   2.3.1 Effect on Circuit Reliability  24
   2.3.2 Effect on Circuit Performance  25
   2.3.3 Effect on Power Dissipation  27
2.4 Related Work on Interconnect-Aware High-Level Synthesis  31
2.5 Related Work on Temperature-Aware High-Level Synthesis  34

## CHAPTER 3  LAYOUT-AWARE BINDING FOR THREE-DIMENSIONAL INTEGRATED CIRCUITS

3.1 Introduction  38
   3.1.1 3-D Integration Technologies  40
   3.1.2 3-D Inter-Layer Vias  42
   3.1.3 Benefits of 3-D Integration  43
   3.1.4 Related Work  44
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4.2</td>
<td>Temperature Reduction through Temperature-Aware HLS</td>
<td>111</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Comparison with Related Work</td>
<td>119</td>
</tr>
<tr>
<td>5.5</td>
<td>Conclusions</td>
<td>124</td>
</tr>
<tr>
<td>CHAPTER 6</td>
<td>STOCHASTIC WIRELENGTH ESTIMATION BASED HIGH-LEVEL DESIGN SPACE EXPLORATION</td>
<td>125</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction and Motivation</td>
<td>125</td>
</tr>
<tr>
<td>6.2</td>
<td>Related Work</td>
<td>132</td>
</tr>
<tr>
<td>6.3</td>
<td>Rent Rule-based Stochastic Wirelength Estimation</td>
<td>134</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Experimental Validation of Stochastic Wirelength Estimation</td>
<td>141</td>
</tr>
<tr>
<td>6.4</td>
<td>Iterative Binding for High-Level Design Space Exploration</td>
<td>144</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Estimating Placement of RTL Modules</td>
<td>148</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Clock Period Estimation</td>
<td>150</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Iterative Binding Algorithm</td>
<td>153</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Runtime-vs-Accuracy Trade-off of Estimation Algorithm</td>
<td>156</td>
</tr>
<tr>
<td>6.5</td>
<td>Experimental Results</td>
<td>158</td>
</tr>
<tr>
<td>6.6</td>
<td>Conclusions</td>
<td>163</td>
</tr>
<tr>
<td>CHAPTER 7</td>
<td>A GENETIC ALGORITHM FOR HIGH-LEVEL DESIGN SPACE EXPLORATION</td>
<td>164</td>
</tr>
<tr>
<td>7.1</td>
<td>Motivations for this Work</td>
<td>165</td>
</tr>
<tr>
<td>7.2</td>
<td>Related Work</td>
<td>167</td>
</tr>
<tr>
<td>7.3</td>
<td>Design of the Genetic Algorithm</td>
<td>171</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Overall Structure</td>
<td>171</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Solution Encoding</td>
<td>175</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Crossover Operators</td>
<td>177</td>
</tr>
<tr>
<td>7.3.3.1</td>
<td>One-Point Topological Crossover</td>
<td>177</td>
</tr>
<tr>
<td>7.3.3.2</td>
<td>Crossover Operation for Resource Allocation Sub-string</td>
<td>179</td>
</tr>
<tr>
<td>7.3.4</td>
<td>Mutation Operators</td>
<td>179</td>
</tr>
<tr>
<td>7.3.4.1</td>
<td>Mutation Operation for Node Priority Sub-string</td>
<td>180</td>
</tr>
<tr>
<td>7.3.4.2</td>
<td>Mutation Operation for Resource Allocation Sub-string</td>
<td>181</td>
</tr>
<tr>
<td>7.3.5</td>
<td>Solution Cost</td>
<td>182</td>
</tr>
<tr>
<td>7.4</td>
<td>Experimental Results</td>
<td>188</td>
</tr>
<tr>
<td>7.5</td>
<td>Analysis and Discussion</td>
<td>193</td>
</tr>
<tr>
<td>7.6</td>
<td>Summary and Conclusions</td>
<td>196</td>
</tr>
<tr>
<td>CHAPTER 8</td>
<td>CONCLUSIONS AND FUTURE WORK</td>
<td>198</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>ABOUT THE AUTHOR</td>
<td>End Page</td>
<td></td>
</tr>
</tbody>
</table>
**LIST OF TABLES**

<table>
<thead>
<tr>
<th>Table 3.1</th>
<th>Results for 3-D Layout-Aware Binding</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Comparison of Interconnect Delay Estimates with and without Accounting for Net-Topology</td>
<td>66</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Comparison of Net Delays for 70nm Technology</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Comparison of Total Wirelength for 70nm Technology</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Comparison of Chip Area for 70nm Technology</td>
<td>76</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Pertinent Design Data and Execution Times</td>
<td>77</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Benchmark Set - 1</td>
<td>102</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Benchmark Set - 2</td>
<td>103</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Comparison of Peak Chip Temperature using Power-Unaware (Method-A) and Power-Aware (Methods B and C) Techniques</td>
<td>106</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Comparison of Average Power Dissipation using Power-Unaware (Method-A) and Power-Aware (Methods B and C) Techniques</td>
<td>107</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Comparison of Peak Module Power using Power-Unaware (Method-A) and Power-Aware (Methods B and C) Techniques</td>
<td>108</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>Comparison of Peak Chip Temperature using the Temperature-Aware Synthesis Methodology (TABS) and Power-Aware Techniques (Methods B and C)</td>
<td>115</td>
</tr>
<tr>
<td>Table 5.7</td>
<td>Percentage Improvements in Peak Chip Temperature using the Temperature-Aware Synthesis Methodology (TABS) over Power-Aware Techniques (Methods B and C)</td>
<td>116</td>
</tr>
<tr>
<td>Table 5.8</td>
<td>Percentage Area Overhead of the Temperature-Aware Synthesis Methodology (TABS) over Power-Aware Techniques (Methods B and C)</td>
<td>117</td>
</tr>
<tr>
<td>Table 5.9</td>
<td>Comparison with Related Work</td>
<td>122</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.10</td>
<td>CPU Runtimes using TABS</td>
<td>122</td>
</tr>
<tr>
<td>6.1</td>
<td>Characteristics of HLS Benchmarks</td>
<td>139</td>
</tr>
<tr>
<td>6.2</td>
<td>Estimation Errors of Stochastic Wirelengths with Respect to Measured Wirelengths of Standard-Cell Placers</td>
<td>143</td>
</tr>
<tr>
<td>6.3</td>
<td>Clock Period Improvement by the Iterative Binding Algorithm</td>
<td>159</td>
</tr>
<tr>
<td>6.4</td>
<td>CPU Runtimes for the Iterative Binding Algorithm</td>
<td>162</td>
</tr>
<tr>
<td>7.1</td>
<td>Example Schedule by Modified List-Scheduling Heuristic</td>
<td>184</td>
</tr>
<tr>
<td>7.2</td>
<td>GA Results for DSP Benchmarks</td>
<td>190</td>
</tr>
<tr>
<td>7.3</td>
<td>Comparison of Proposed GA-Based Method with Other Scheduling Methods</td>
<td>192</td>
</tr>
<tr>
<td>7.4</td>
<td>Comparison of Proposed GA-Based Method with Other Heuristic HLS Techniques</td>
<td>193</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Interconnect Delay Trend with Technology Scaling</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Floorplan-Level Thermal Distribution of an Integrated Circuit</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>A Conventional High-Level Synthesis Flow</td>
<td>18</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>(a) A Scheduled Dataflow (DFG) Graph for a Behavioral Description,</td>
<td>19</td>
</tr>
<tr>
<td></td>
<td>(b) Output of High-Level Synthesis - An RTL Netlist</td>
<td></td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Typical Physical Synthesis Flow</td>
<td>21</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>(a) Transistor Threshold Voltage at Different Operating Temperatures,</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>(b) Sub-Threshold Leakage Current at Different Threshold Voltages,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(c) Variation of Leakage Power with Temperature</td>
<td></td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Example of the Implementation of a 3-D IC using Through-Silicon Vias</td>
<td>38</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>Wirelength Reduction with 3-D Integration</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>(a) Planar Circuit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(b) 3-D Integrated Circuit</td>
<td></td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Wirelength Distribution of 3-D IC as a Function of Number of Device Layers</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>used (from [134])</td>
<td></td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Motivating Example of Resource Binding in a 3-D Integrated Circuit</td>
<td>47</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>3-D Chip-Package Structure (from [179])</td>
<td>47</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Overall Flow of 3-D Layout-Aware Binding for High-Level Synthesis</td>
<td>50</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>Wirelength Trends for HLS Benchmarks with 1, 2, 3, and 5 Floorplan Layers</td>
<td>56</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>Cumulative Wirelength Distributions for HLS Benchmarks with 1, 2, 3, and 5</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>Floorplan Layers</td>
<td></td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>Improvement in Wirelength over 3-D Layout-Unaware Binding</td>
<td>59</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>Wirelength Distributions for 3-D Layout-Aware and Layout-Unaware Bindings</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>for HLS Benchmarks with Three Floorplan Layers</td>
<td></td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Improvement in Inter-Die Via Count over 3-D Layout-Unaware Binding</td>
<td>60</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>A Motivating Example</td>
<td>65</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Traditional Physically-Driven High-Level Synthesis</td>
<td>69</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Proposed Net-Topology Aware Synthesis Algorithm</td>
<td>72</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Cost Function Computation</td>
<td>73</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Comparison of Net Delay Distributions for HLS Benchmarks</td>
<td>78</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>ITRS 2006 Projections for High-Performance ASIC Designs</td>
<td>82</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Relative Cost of Heat Removal from a Microprocessor (from [64])</td>
<td>84</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Plot of Peak Chip Temperature of Twenty-Five Datapath Designs for the COSINE-2 HLS Benchmark Synthesized using a Low-Power Design Methodology that Minimizes Average Power</td>
<td>86</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>Post-Synthesis Thermal Profiles of Two Different Designs with Identical Resource Allocations and Power Dissipations but with different HLS Schedules, Module Bindings, and Floorplans</td>
<td>87</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Impact of HLS Sub-Tasks on Chip Power and Temperature</td>
<td>88</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Effect of Floorplan on Module Temperature</td>
<td>89</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>Overview of TABS, the Proposed Approach for Temperature-Aware Unified Physical-Level and High-Level Synthesis Algorithm</td>
<td>91</td>
</tr>
<tr>
<td>Figure 5.8</td>
<td>Multi-Stage Annealing Schedule</td>
<td>94</td>
</tr>
<tr>
<td>Figure 5.9</td>
<td>Solution Sub-Spaces Sampled at Different Stages of Search by Multi-Stage Simulated Annealing</td>
<td>94</td>
</tr>
<tr>
<td>Figure 5.10</td>
<td>Multi-Stage SA Controller</td>
<td>95</td>
</tr>
<tr>
<td>Figure 5.11</td>
<td>Solution Cost Function Evaluation</td>
<td>99</td>
</tr>
<tr>
<td>Figure 5.12</td>
<td>Floorplan-Level Thermal Distribution for the interpolate-aux MediaBench Benchmark Synthesized using the Power-Unaware Method-A</td>
<td>109</td>
</tr>
<tr>
<td>Figure 5.13</td>
<td>Floorplan-Level Thermal Distributions for the interpolate-aux MediaBench Benchmark Synthesized using the Low-Power Methods B and C</td>
<td>110</td>
</tr>
<tr>
<td>Figure 5.14</td>
<td>Floorplan-Level Thermal Distributions for the interpolate-aux MediaBench Benchmark Synthesized using the Low-Power Method-B and the Proposed Temperature-Aware Synthesis Method TABS</td>
<td>113</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>5.15</td>
<td>Comparison of Floorplan Areas using the Proposed Temperature-Aware Synthesis (TABS), and the Power-Aware Synthesis Methods (A and B)</td>
<td>118</td>
</tr>
<tr>
<td>5.16</td>
<td>Convergence Plot of the Peak On-Chip Temperature for the \textit{IDCT_col} MediaBench Benchmark using the Proposed Approach</td>
<td>123</td>
</tr>
<tr>
<td>6.1</td>
<td>A Representative Flow for High-Level Synthesis</td>
<td>126</td>
</tr>
<tr>
<td>6.2</td>
<td>Conventional Wirelength-Driven High-Level Synthesis Flow</td>
<td>128</td>
</tr>
<tr>
<td>6.3</td>
<td>Proposed Wirelength-Driven High-Level Synthesis Flow</td>
<td>130</td>
</tr>
<tr>
<td>6.4</td>
<td>Typical Rent Characteristic of a Digital Circuit</td>
<td>135</td>
</tr>
<tr>
<td>6.5</td>
<td>Stochastic Wirelength Estimation for High-Level Synthesis</td>
<td>136</td>
</tr>
<tr>
<td>6.6</td>
<td>Dynamic Rent Parameter Extraction</td>
<td>138</td>
</tr>
<tr>
<td>6.7</td>
<td>Rent Characteristic Plots for HLS Benchmarks</td>
<td>140</td>
</tr>
<tr>
<td>6.8</td>
<td>Comparison of Wirelength Estimates with Post-Placement Wirelengths of DRAGON, CAPO, FengShui, and Silicon Ensemble</td>
<td>142</td>
</tr>
<tr>
<td>6.9</td>
<td>Overall Flow of Iterative Binding Algorithm for Wirelength and Clock Period Improvement</td>
<td>145</td>
</tr>
<tr>
<td>6.10</td>
<td>Cost Function Computation in the Iterative Binding Algorithm</td>
<td>146</td>
</tr>
<tr>
<td>6.11</td>
<td>Layout-Area Partitioning for Estimating Gate Placements</td>
<td>149</td>
</tr>
<tr>
<td>6.12</td>
<td>Determining Coordinates and Dimensions of RTL Modules</td>
<td>149</td>
</tr>
<tr>
<td>6.13</td>
<td>Datapath Model for Clock Period Estimation</td>
<td>151</td>
</tr>
<tr>
<td>6.14</td>
<td>Clock Period Estimation</td>
<td>152</td>
</tr>
<tr>
<td>6.15</td>
<td>Estimating Module-to-Module Wire Delays</td>
<td>152</td>
</tr>
<tr>
<td>6.16</td>
<td>Iterative Binding Algorithm</td>
<td>155</td>
</tr>
<tr>
<td>6.17</td>
<td>Accuracy of Wirelength Estimation as a Function of Partitioning Depth of Gate-Level Netlist</td>
<td>157</td>
</tr>
<tr>
<td>6.18</td>
<td>Convergence Plots of the Iterative Binding Algorithm for IIR and EWF Benchmarks</td>
<td>160</td>
</tr>
<tr>
<td>6.19</td>
<td>Convergence Plots of the Iterative Binding Algorithm for DCT and FIR Benchmarks</td>
<td>161</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>6.20</td>
<td>Runtime Comparison between HLS Design Space Exploration with Traditional Place &amp; Route and the Proposed Method</td>
<td>162</td>
</tr>
<tr>
<td>7.1</td>
<td>Framework of Proposed High-Level Synthesis System</td>
<td>172</td>
</tr>
<tr>
<td>7.2</td>
<td>Structure of Genetic Algorithm for High-Level Synthesis</td>
<td>173</td>
</tr>
<tr>
<td>7.3</td>
<td>Solution Encoding used in Genetic Algorithm</td>
<td>175</td>
</tr>
<tr>
<td>7.4</td>
<td>One-Point Topological Crossover</td>
<td>178</td>
</tr>
<tr>
<td>7.5</td>
<td>Precedence Preserving Shift Mutation</td>
<td>181</td>
</tr>
<tr>
<td>7.6</td>
<td>Resource Allocation Sub-string Mutation</td>
<td>182</td>
</tr>
<tr>
<td>7.7</td>
<td>Solution Fitness Evaluation Procedure</td>
<td>183</td>
</tr>
<tr>
<td>7.8</td>
<td>Modified List-Scheduling Heuristic</td>
<td>184</td>
</tr>
<tr>
<td>7.9</td>
<td>Left Edge Algorithm</td>
<td>185</td>
</tr>
</tbody>
</table>
TEMPERATURE AND INTERCONNECT AWARE UNIFIED PHYSICAL AND HIGH LEVEL SYNTHESIS

Vyas Krishnan

ABSTRACT

Aggressive scaling of nanoscale CMOS integrated circuits has created significant design challenges arising from increasing power densities, thermal concerns, and rising wire delays. The main contribution of this dissertation is the development of unified physical and high-level synthesis techniques for the design of ASICs with optimal chip temperatures and interconnect delays.

Thermal issues are becoming a serious problem in high-performance VLSI circuits, adversely impacting performance, reliability, power consumption, and cooling costs. To address this, we present a temperature-aware behavioral synthesis (TABS) framework that combines power minimization with temperature-aware task scheduling, resource binding, and floorplanning. Compared to conventional low-power synthesis methods, our approach is effective in synthesizing circuits with lower chip temperatures and more uniform thermal distributions, with temperature reductions up to 23% when compared to low-power synthesis.

We propose three techniques to address interconnect delays during high-level synthesis: (1) a simulated annealing (SA) based layout-aware high-level synthesis technique for 3-D integrated circuits, that tightly couples the synthesis tasks of resource binding and 3-D floorplanning. The proposed algorithm significantly outperforms a conventional synthesis flow that separates the binding and floorplanning steps, with improvements in the total wirelength by 29% and of the
longest wirelength by 21%; (2) a floorplan-aware high-level synthesis technique that uses the topology of multi-terminal nets to improve interconnect delay estimates during resource binding. Experiments show that the use of accurate wire delay estimates during binding can reduce wire delays by as much as 49% in 70nm technology; (3) an iterative high-level design-space exploration engine that uses a priori stochastic wirelength estimates to guide binding decisions during high-level synthesis. The proposed approach offers a significant speed-up during design space exploration when compared to approaches that use traditional place-and-route to evaluate candidate solutions.

Finally, we present a genetic algorithm (GA) based approach for high-level synthesis. We propose novel GA encoding, crossover, and mutation operators for the problem. The quality of the results generated by the GA are superior to those of several other techniques reported in the literature.
CHAPTER 1

INTRODUCTION

The exponential scaling in CMOS transistor sizes over the past three decades have enabled spectacular advances in integrated circuit technology, allowing the integration of more than a billion transistors in modern very large-scale integrated (VLSI) circuits. Over the last four decades, transistor scaling has followed Moore's law [1, 2], and according to projections made by the International Technology Roadmap for Semiconductors (ITRS) [3], minimum feature sizes are expected to reach 24nm by 2012. The primary drivers for transistor scaling are the associated benefits of lower system costs, improved performance, and system reliability.

Relentless technology scaling, however, poses some new challenges to VLSI designers. Some of these challenges include poor scaling of interconnect delays [4, 5], increased power consumption [6], rising thermal concerns [7, 8, 9], and manufacturing challenges [10]. The semiconductor industry must overcome these challenges to keep pace with Moore's law [2] and industry projections [3].

Despite these challenges, technology scaling offers tremendous opportunities for designers to cram increasingly more complex designs on a chip, to meet the ever increasing demand for multi-media consumer products, mobile devices, and other “smart” products in today's competitive markets. However, with product cycles getting shorter and typical chip design sizes increasing, the ability to turn out fast, reliable, and cost-effective electronic designs of increasing complexity in the shortest possible time is also proving to be an enormous design challenge for VLSI designers. The rate of increase in semiconductor technology advances has far outpaced the
ability of designers to effectively utilize the silicon real-estate, leading to the well-known designer productivity gap [12]. Consequently, designers must find efficient ways to exploit the full potential of a semiconductor manufacturing process when designing complex VLSI ASICs.

There is a growing consensus among VLSI designers that one of the most effective methods to manage the increasing size and complexity of today's system-on-chip designs is to use computer-aided design (CAD) techniques that start with an abstract behavioral or algorithmic description of a circuit and automatically synthesize a digital circuit that realizes the behavior. This has motivated the development of High-Level Synthesis (HLS) or Behavioral Synthesis [12, 13, 14]. The main advantages offered by high-level synthesis are a reduction in design time and a more efficient design space exploration.

Although high-level synthesis has been the subject of research for two decades, there are two main reasons that have limited the utility and wider acceptance of high-level synthesis:

- The quality of synthesis results is compromised by inadequate accounting of interconnect effects such as increased wire delays, crosstalk, and interconnect power. Effects such as increased power densities and on-chip thermal-hotspots further exacerbate the problem.
- And designers are usually given minimal controllability of the synthesis process and its results.

In this dissertation, we address these two limitations through interconnect-aware and temperature-aware high-level synthesis. The rest of this chapter is organized as follows. Section 1.1 discusses the need for interconnect-awareness and Section 1.2 the need for temperature-awareness during high-level synthesis in current and future deep submicron technology nodes. Section 1.3 discusses the need for layout-awareness during high-level synthesis. Section 1.4 discusses the contributions of this dissertation. The dissertation outline is given in Section 1.5.
1.1 Need for Interconnect-Awareness

As reported by the International Technology Roadmap for Semiconductors (ITRS) [3], the feature size of VLSI devices has scaled down from 130nm in 2001, to 45nm in 2008. The current trend on technology scaling is expected to continue for another decade. With feature sizes continuing to shrink, interconnect delay has become a limiting factor in the performance of VLSI circuits. In keeping with Moore's law, transistor sizes decrease with each technology generation. The reduced transistor size enables higher integration density by providing more transistors in the same silicon area as previous technology generations. With smaller transistors, their switching speeds increase with successive technology scaling generations. The increased switching speed of transistors enables a higher frequency of operation. The improved integration density and the increased speed together enable higher functionality and higher overall functionality of the VLSI circuits.

This increasing functionality of VLSI circuits, enabled by higher transistor counts, leads to higher wiring complexity in these circuits. This is a consequence of the increasing complexity of circuits: the number of wires grows exponentially with the number of gates, according to Rent's rule [18]. Unfortunately, the interconnect delays have not improved at the same rate as transistor delays with technology scaling [4, 5]. Performance improvements gained through transistor scaling may be diminished by the negative effects of technology scaling, since interconnect delay begins to dominate over transistor delay. For example, if devices and interconnects are all scaled down by a factor of $S$, the intrinsic gate delay will decrease by a factor of $S$, and the delay of local interconnects (connecting adjacent gates) remains the same, while the delay of global interconnects increases by a factor of $S^2$. Figure 1.1 shows the ITRS projections for gate and interconnect delays. While local interconnects scale roughly with feature the size, intermediate and global interconnections certainly do not. Thus, the performance of VLSI circuits is increasingly limited not by the transistor delay, but rather by the interconnect delay. This poses a
Figure 1.1  Interconnect Delay Trend with Technology Scaling

major challenge in the future system-level design, implying that “interconnect-driven architectures” are becoming critical. Thus, with scaling of wire dimensions in deep submicron technologies, many interconnect issues which could be safely ignored in older technologies have become prominent now.

With technology scaling, wires are placed closer to each other and the aspect ratio increases. This leads to larger coupling capacitance, which causes crosstalk noise and excessive signal delay. With higher clock frequencies, faster transistor rise/fall times, and longer signal wires, inductance of interconnects, and the noise generated because of this inductance has become important in design.

Given the dominating importance of interconnects in current and future generations of VLSI designs, interconnect design and optimization needs to be considered and emphasized at all stages of the design process [5]. This is in contrast to conventional high-level synthesis flows in
older technologies, where most of the emphasis was given to the design and optimization of logic (i.e., functional units such as adders, multipliers, registers, and multiplexers), and wire delays could be ignored without loss of accuracy. All these factors have made interconnect-centric design one of the most significant problems for high-performance ASICs.

Traditionally, high-level synthesis consists of the three main steps – scheduling, allocation, and binding, which are typically performed in a sequential manner. Interconnect-awareness during high-level synthesis adds an additional step: interconnect optimization. An optimal solution may not be found by applying these steps sequentially because the optimizations are not independent. Consequently, because interconnect optimization at this level requires a detailed understanding of the target architecture, heuristics are needed to synthesize such architectures in an interconnect-optimized way under simultaneous consideration of scheduling, allocation, binding, and floorplanning.

1.2 Need for Temperature-Awareness

With decreasing feature sizes and increasing transistor counts, power density in VLSI circuits has increased dramatically. Since the heat generated by a VLSI circuit is proportional to its power density, the corresponding rise in on-chip temperatures adversely impacts reliability, circuit performance, and cooling costs. According to ITRS, thermal management is projected as one of the most challenging issues in the design of future high-performance integrated circuits [3]. Power-aware design alone fails to adequately address thermal issues, thus creating a need for temperature-aware low-power design at all system levels, including behavioral synthesis.

Though temperature-aware design makes use of power-management techniques, it is significantly different from traditional power-aware design. Traditional low-power synthesis techniques typically use overall power budgets in their optimization objectives. These low-power
techniques typically do not account for spatial on-chip power density variations, and hence cannot accurately estimate on-chip thermal distributions.

Different functional units in a circuit can have different switching activity rates, leading to uneven power dissipation among the various logic blocks on the chip. Due to low thermal conductivity of silicon, the rate of lateral heat propagation in a chip is slow, causing localized heating to occur much faster than chip-wide heating. This can result in an uneven temperature distribution on the chip, creating on-chip thermal gradients, and “thermal hot spots” caused by localized areas of high power densities. Figure 1.2 shows an example of a floorplan-level thermal distribution on a VLSI chip. In the figure, the rectangles represent circuit modules color-coded by their peak temperatures. From Figure 1.2, it can be seen that there exists a non-uniform temperature distribution across the chip caused by differences in the power consumptions of modules.
Traditional low-power behavioral synthesis techniques that minimize average power do not take on-chip thermal distributions into account during synthesis. Thus, traditional low-power synthesis methods cannot prevent the occurrence of localized areas of high on-chip temperatures (caused by localized areas of high power density) and large thermal gradients across a chip. With power densities and thermal gradients projected to increase rapidly in future technologies [3, 9], there is a need for temperature-aware low-power design techniques that directly target the spatial nature of on-chip temperature distributions. This makes temperature-awareness during synthesis an important requirement in low-power behavioral synthesis.

Elevated temperatures have several adverse effects on a circuit. Higher temperatures adversely impact performance due to decreased transistor switching speeds and increased wire resistance, which can lead to timing violations [9]. On-chip thermal gradients cause the thermal profile of interconnects to be non-uniform, adversely affecting clock signal and interconnect performance [96]. Thermal gradients also impact layout-level optimization schemes such as gate-sizing, wire-sizing, and buffer insertion schemes, since gate and interconnect delays are dependent on temperature [96]. In addition, non-uniform temperature distribution impacts the integrity of the power/ground network in a chip [8]. Sub-threshold leakage currents increase exponentially with temperature, leading to an exponential increase in static power dissipation in CMOS circuits [19]. This coupling between static power and rising temperature can lead to thermal runways if not adequately managed [9]. Higher temperatures also cause reliability problems since they exacerbate electro-migration in metal wires, hot-carrier effects in transistors, and thermal-stress related gate-oxide breakdown in transistors [8]. In addition, chip packaging costs increase with higher thermal budgets.

Temperature-aware high-level synthesis is challenging because of the multi-objective nature of the problem, with several conflicting objectives – throughput rate, power consumption, peak temperature, and chip area. During high-level synthesis, decisions regarding scheduling and
assigneed of tasks to functional units are made. Decisions made during scheduling and binding
determine the static and dynamic power of datapath units. Functional units with higher switching
activity have higher power dissipation, and correspondingly higher temperatures. An increase in
temperature increases sub-threshold leakage, leading to further increases in temperature.
Resource allocation and sharing also affects the resulting power dissipation and temperature.
Allocating more resources reduces power density and temperature, whereas more resource
sharing may increase switching power and temperature. High-level synthesis thus has a
significant impact on the power density and thermal distribution of a design.

Floorplanning also significantly affects on-chip thermal distribution. The temperature of a
functional unit is determined not only by its power density but also by the temperature of other
functional units in its vicinity. Changing functional unit positions on a floorplan to balance
power density and thereby reduce chip temperatures may increase chip area. Conversely, placing
highly connected functional units bound to timing critical tasks, close to each other to minimize
wire delay and power, could lead to localized areas of high chip temperatures, especially if the
modules also have high switching activity.

An effective temperature-aware design technique must tightly couple floorplanning and high-
level synthesis, with some form of feedback from thermal analysis.

1.3 Need for Unifying Physical-Level and High-Level Synthesis

In a high-level synthesis flow, the RTL netlist forms the interface between HLS and physical
synthesis. High-level synthesis systems have traditionally optimized the RTL structure, but the
real cost of the synthesized design will ultimately be measured in the physical domain in terms of
circuit delay, chip area, and power dissipation. The high-level synthesis system therefore
optimizes the final cost of the solution indirectly.
A design may need to be iterated through the HLS and physical synthesis steps several times if design constraints such as timing, area, and power are not met. Due to the separation of physical synthesis from behavioral synthesis, often the impact of design decisions taken during high-level synthesis will only be known after physical synthesis, necessitating numerous design iterations to achieve design closure. To address this, feedback paths between HLS and physical synthesis steps must exist to incorporate physical-level information during HLS. Use of feedback improves the interaction between HLS and physical synthesis and refines the quality of the solution generated at each stage. Feedback measures employed during HLS typically use floorplan or place and route information to guide high-level design decisions. Alternatively, the post-synthesis results from physical synthesis, could be used as feedback to guide HLS decisions.

Estimating interconnect delay between the modules in a design requires physical information of the placement of these modules. The wirelength between modules depends on the location of the modules of the design on the floorplan and the routing structure between them. Similarly, estimating on-chip thermal distribution requires information about module power consumptions as well as their locations on the floorplan.

The three main steps of high-level synthesis viz., scheduling, allocation, and binding, are typically performed in a sequential manner. Interconnect or temperature awareness during high-level synthesis adds an additional step: interconnect or temperature optimization. An optimal solution may not be found by applying these steps sequentially because the optimizations are not independent. Consequently, heuristics are needed to synthesize architectures that are interconnect and temperature optimized under simultaneous consideration of scheduling, allocation, binding, and floorplanning.

The synthesis techniques proposed in this dissertation tightly integrate physical-level and high-level synthesis steps, to find interconnect and temperature optimized designs during HLS design space exploration.
1.4 Contributions of the Dissertation

Continuous device and interconnect scaling trends in deep submicron designs has created new challenges for integrated circuit designers such as increased interconnect delays due to rising parasitic resistance and capacitance of on-chip wiring, increased on-chip power densities, and performance and reliability problems posed by on-chip thermal gradients and thermal-hotspots. Thus, the major challenge is in achieving reliable, high-performance system implementations, all the way from the micro-architecture level down to the layout level. In order to realize such an implementation, a unified physical-level and high-level synthesis method becomes paramount, to ensure predictability of HLS design flows and minimize design iterations. The main contributions of this dissertation are summarized below.

1.4.1 Layout-Aware Resource Binding for 3-D Integrated Circuits

Three-dimensional (3-D) integrated circuit technology is a new technology that has the potential to alleviate many of the performance and power related issues raised by interconnects in conventional planar (2-D) integrated circuits. In a planar 2-D technology, floorplanning and layout constraints may force two connected circuit modules to be physically separated, thus requiring long global wires for communication. In a 3-D integrated circuit, these circuit modules may be stacked on top of each other, thus replacing long global wires by short vertical interconnects. 3-D integration provides increased device density, reduced latency, and lower power [45, 46]. The relative benefits of 3-D integration technology will increase in future technology generations [46], making it a very attractive option for future designs. There is a pressing need for electronic design automation (EDA) tools and methodologies to fully exploit the potential of 3-D technology. Xie et al. [47] have identified have two different categories of EDA tools essential for 3-D microarchitecture designs: early design analysis tools and physical design tools. While there has been a great deal of on-going research in the academic community
on 3-D EDA tools [48, 49], very little work has been done in the area of interconnect-aware high-level synthesis for 3-D technology. We address this issue in this dissertation by proposing a novel 3-D layout-aware binding algorithm for high-level synthesis of 3-D integrated circuits [50].

Physical synthesis for 3-D integrated circuits is substantially different from traditional planar 2-D integrated circuits due the presence of additional constraints posed by the need to place circuit blocks in multiple silicon die. To realize the full potential offered by 3-D integrated circuits, high-level synthesis of these circuits must take layout-related issues unique to 3-D technology into account during synthesis. In this dissertation, we propose a 3-D layout aware binding algorithm for high-level synthesis [50], that tightly integrates the synthesis tasks of resource binding, assignment of datapath functional units to multiple silicon layers, 3-D floorplanning, and through-silicon via minimization. Since floorplanning and resource binding are inter-dependent, we demonstrate that the proposed algorithm significantly outperforms a conventional synthesis flow that separates the binding and floorplanning steps. Compared to a 3-D layout-unaware approach, our experiments show an improvement in the total wirelength of 29% on average, while the longest wirelength is reduced by 21%. In addition, the number of through-silicon vias are reduced by 27%. These optimizations were achieved with no penalty in chip area.

1.4.2 Net Topology Aware Resource Binding for Wire Delay Minimization

Most of the existing interconnect-aware high-level synthesis approaches [15, 16, 17] base their interconnect estimates on simple net-length models such as half-perimeter bounding box, center-to-center, or 2-terminal nets, on a floorplan of RTL modules. However, estimates based on these cannot accurately model interconnect delays in deep submicron technologies [44], thus leaving room for improvements in solution quality. In this dissertation, we develop accurate net-topology based interconnect estimation models [44], and show that these can significantly
improve solution quality in an interconnect-aware high-level synthesis algorithm. No prior work exists in literature that accounts for the topology of nets resulting from resource binding decisions during high-level synthesis. We propose a novel floorplan-aware high-level synthesis technique that uses accurate net topologies and distributed wire-delay models to guide resource allocation and binding decisions during design-space exploration. The proposed approach tightly integrates a floorplanner with a high-level synthesis binding algorithm. The location of data path modules in the floorplan is used to determine the minimal length Rectilinear Steiner Minimum Tree (RSMT) of every net, to which the delay model is applied to accurately estimate delays of multi-terminal nets. Our results show that, when compared to previous approaches, our method reduces wire delays by as much as 48.9% in 70nm technology, with an average improvement of 38.6%, and an overhead of only 3.6% in chip area.

1.4.3 Stochastic Wirelength Estimation Based Design Space Exploration

Accounting for layout information during high-level synthesis design space exploration has the potential to address many of the problems associated with the quality of synthesized results from conventional high-level synthesis. A straight-forward way to account for layout information is to actually go through a back-end physical synthesis procedure every time a candidate solution is generated. However, this would be computationally inefficient if thousands of candidate solutions need to be evaluated during design space exploration. Since the main objective of high-level synthesis is to quickly explore the large design space and identify potential candidate solutions that meet design constraints, interconnect-aware high-level synthesis algorithms use estimates of the layout that can be computed efficiently, and be reasonably accurate.

In this dissertation, we show that stochastic wirelength estimation is a viable technique for evaluating the interconnect complexity of designs explored during high-level synthesis [39]. Stochastic wirelength estimation uses analytical models to estimate wirelength distributions of
logic gate netlists [18, 40, 41]. These models are based on an empirical law called Rent's rule [18] that relates the number of terminals in a logic block to the number of gates in the block, and is expressed by the relation

\[ T = kN^p \]

where, \( p \) is known as the Rent's constant, and \( k \) is called the Rent's coefficient. The Rent's constant, whose value lies in the range \( 0 < p < 1 \), provides an indication of the wiring complexity of a circuit, while the Rent coefficient (\( k \)) indicates the average number of terminals per gate. Davis et al. [41] proposed a stochastic model that computes the wirelength distribution of a gate-array in terms of its gate-count and Rent parameters. During HLS design space exploration, often hundreds of designs with different netlist structures are examined, in an attempt to find an optimal solution that meets all design constraints. We propose an efficient technique to dynamically extract the Rent parameters of a gate-level netlist [39], and use it to estimate the wiring complexity of the datapath netlists examined during design space exploration. We also develop an iterative HLS design-space exploration engine that uses this information to guide module and register binding decisions during high-level synthesis, with goal of synthesizing a design with the minimal achievable clock period [42, 43]. To the best of our knowledge, this is the first work to apply stochastic wirelength estimation to interconnect-aware high-level synthesis. The key advantage to using our approach is that it can be applied to non-hierarchical (\( i.e. \) flattened) gate-level layouts of standard-cells or gate arrays. In contrast, most existing approaches to interconnect-aware HLS derive their wirelength estimates from a floorplan of RTL macro-cells, making their approach only applicable to macro-cell based layouts. In addition, our algorithm is an order-of-magnitude faster than a traditional synthesis technique that uses a full place-and-route as part of the design space exploration process.
1.4.4 Temperature-Aware Layout-Driven High-Level Synthesis

With increasing device counts and operating frequencies, total power and power density is becoming a serious problem in high-performance VLSI circuits. In these circuits, power can be unevenly distributed, leading to thermal-hotspots with significantly greater temperatures than surrounding regions. Elevated chip temperatures have an adverse impact on performance, reliability, power consumption, and cooling costs. Thermal-hotspots lead to serious reliability issues such as thermal-runaways [8]. To ensure adequate thermal management, all phases of the design flow must account for thermal effects on their design decisions.

In the dissertation, we propose an integrated approach to power and thermal management during high-level synthesis [51, 52], through the use of a two-stage simulated annealing-based synthesis technique that combines power minimization with temperature-aware scheduling, binding, and floorplanning. In the proposed method, the first-stage of the simulated annealing algorithm creates a low-power solution, which is then iteratively improved by the second stage to minimize estimated on-chip peak temperature, using ISAC [53], an accurate module-level temperature estimation tool. We show through extensive experiments that minimizing average power alone does not guarantee minimal peak temperatures. However, our approach consistently finds solutions that have lower on-chip peak temperatures and uniform on-chip temperature distributions, compared to a traditional low-power synthesis methodology that minimizes average power. Our method reduces peak temperatures by an average of 15% and up to 23%, compared to traditional low-power synthesis that minimizes average power. These improvements in chip-level temperature distributions are achieved with a modest increase in chip area of under 9% on average.
1.4.5 A Genetic Algorithm for High-Level Design Space Exploration

High-level synthesis consists of interdependent sub-tasks such as scheduling, allocation, and binding. Each of these sub-tasks is NP-hard [12], and are therefore usually performed sequentially [2]. Thus for example, the tasks in input behavioral description are first scheduled, followed resource allocation, and then resource binding. However, performing these steps concurrently, instead of sequentially, could lead to better solutions [12].

For today’s VLSI designs, the cost of solving the combined scheduling, allocation, and module selection problem by exhaustive search is prohibitive. However, to meet design objectives, an extensive design space exploration is often critical to obtaining superior designs. In the dissertation, we present a Genetic Algorithm (GA) based approach for combined scheduling, allocation, and binding during high-level synthesis of datapaths during design space exploration [54]. The genetic algorithm uses a multi-chromosome representation to encode datapath schedules and module allocations and efficient heuristics to minimize functional and storage area costs, while minimizing circuit latencies. Our technique offers several advantages over traditional high-level synthesis methods.

Our approach combines the global search capability of genetic algorithms with fast local search heuristics for scheduling and register allocation. This combination allows our synthesis system to search a large, multi-modal design space, in an efficient manner, in order to find the best possible solution within reasonable CPU times. Our approach provides the flexibility to perform resource-constrained scheduling, time-constrained scheduling, or a combination of the two, using a simple and fast list-scheduling technique. A graded penalty function is used as an objective function in evaluating the quality of designs to enable the genetic algorithm to quickly reach areas of the search space where designs meeting user specified criteria are most likely to be found. Since genetic algorithms are population-based search heuristics, a unique feature of our
framework is its ability to offer a large number of alternative datapath designs, all of which meet
design specifications but differ in module, register, and interconnect configurations.

1.5 Organization of the Dissertation

The remainder of this dissertation is organized as follows. In Chapter 2, we provide requisite
background on the main issues presented in this dissertation, followed by a discussion of related
work. Chapter 3, provides a background on three-dimensional integrated circuit technology and
describes our 3-D layout-aware binding algorithm for high-level synthesis. Chapter 4 motivates
the need for net-topology aware interconnect delay estimation for high-level synthesis, and
develops an algorithm for minimizing wire delays by net-topology aware binding during
floorplan-driven high-level synthesis. In Chapter 5 we define the problem of temperature-aware
behavioral synthesis, and develop an integrated approach to power and thermal management
during high-level synthesis, through the use of a two-stage simulated annealing-based synthesis
technique that combines power minimization with temperature-aware scheduling, binding, and
floorplanning. In Chapter 6 we show that stochastic wirelength estimation is a viable technique
for evaluating the interconnect complexity of designs explored during high-level synthesis. In the
chapter, we propose an iterative HLS design-space exploration method that uses a Rent-
parameter based stochastic wirelength estimation to guide module and register binding decisions
during high-level synthesis, with goal of synthesizing a design with the minimal achievable clock
period. In Chapter 7, we introduce a Genetic Algorithm-based high-level synthesis method that
simultaneously searches the sub-spaces of HLS schedules, allocations, and bindings, for an
efficient design-space exploration of digital datapaths. The concluding remarks and future work
related to the dissertation are given in Chapter 8.
CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter provides an overview of high-level and physical-level synthesis, and related work on layout-aware high-level synthesis techniques proposed in the literature. Section 2.1 introduces the basics of high-level synthesis, followed by Section 2.2 which discusses the steps involved in physical synthesis. Section 2.3 discusses the impact of chip temperature on circuit performance and reliability. Sections 2.4 surveys related work on interconnect-aware high-level synthesis, and Section 2.5 describes related work on temperature-aware high-level synthesis.

2.1 An Overview of High-Level Synthesis

Hardware synthesis is the process of taking the a set of specifications for the required behavior of a system, together with a set of constraints to be met, and finding a hardware structure that implements the behavior while meeting the constraints [12]. The “behavior” of a system describes the functionality of the system and its interaction with its environment. In the behavioral domain we are primarily interested in what a design does and not how it is built, treating the design as one or more black boxes with a specified set of inputs and outputs. Examples of behavioral descriptions include Boolean expressions, timing diagrams, and state diagrams. The “structure” of a system refers to the set of interconnected components realizing the system. Examples of structural descriptions include gate-level netlists and register-transfer level (RTL) netlists. The structural description is, in turn, mapped into a “physical” design that implements the specified behavior. Examples of physical descriptions include printed circuit
boards and transistor layouts. A structural representation bridges the behavioral and physical representations. It is a one-to-many mapping of a behavioral representation onto a set of components and their interconnections, under constraints such as cost, area, delay, and power.

High-level Synthesis (HLS) is the process of synthesizing an RTL design from an algorithmic or behavioral description of a digital circuit. Figure 2.1 sketches a conventional high-level synthesis flow. A high-level synthesis flow takes an input behavioral or algorithmic description of a digital circuit, described in a high-level programming language (e.g., C, C++), together with
a set of design constraints (such as chip area, performance, power dissipation, reliability, and testability), and automatically synthesizes a Register-Transfer Level (RTL) structure of the circuit realizing the behavior [12]. Logic and layout synthesis tools are then used to assemble the RTL-netlist, thus providing a route from algorithmic behavior to physical chip layout.

In a typical high-level synthesis flow, the application described in a high-level programming language is processed by a compiler, which performs several optimizations such as constant propagation, loop unrolling, and function in-lining. The compiler generates an intermediate representation for the application, capturing the control flow and data dependencies within the
application. A commonly used model for this intermediate representation is a directed acyclic graph called a dataflow graph (DFG). The nodes of the DFG represent computational tasks (i.e., operations) in the behavioral description and the edges represent the data dependencies among these tasks. A high-level synthesis stage follows the compiler stage and takes the DFG as input and generates a RTL netlist of the design. Figure 2.2(a) shows an example of a DFG consisting of 1 multiplication operation, 1 add operation, and 1 subtraction operation. The edges between the DFG operations set the precedence constraints between the operations. Figure 2.2(b) shows the corresponding RTL netlist.

High-level synthesis consists of three inter-related steps – scheduling, allocation, and binding. The scheduling step assigns time-stamps to the DFG operations, corresponding to the discrete time steps at which the tasks are completed, such that all data dependencies in the DFG are met. Scheduling fixes the order in which DFG operations will be executed.

Allocation is the process of determining the amount of hardware resources required to meet design specifications in the synthesized circuit. The allocation step determines the number of functional units of each type for performing operations, memory units (registers) for storing data values, and interconnect resources (buses, multiplexers, wires) for data transfers. The binding step determines the actual mapping of DFG nodes to functional units, DFG data values to storage, and data transfers to interconnects. The output of HLS is an RTL netlist similar to that shown in Figure 2.2(b).

2.2 An Overview of Physical Synthesis

Physical synthesis consists of a series of steps that transform a structural description of a circuit to a layout of the circuit that can be used to generate masks for fabrication. This is accomplished through several stages such as partitioning, floorplanning, placement, routing, and compaction, as shown in Figure 2.3. A brief discussion of each of these stages follows.
Figure 2.3 Typical Physical Synthesis Flow
**Partitioning** - A typical VLSI chip design may contain millions of gates. Therefore, the design is often partitioned into sub-circuits. These sub-circuits are called blocks. The output of the partitioning step is a set of blocks and the interconnections required between the blocks. In large circuits, the partitioning process is hierarchical and at the top-most level a chip may have about two dozen blocks. Each of these blocks is then partitioned recursively into smaller blocks.

**Floorplanning and Placement** - This step selects good layout alternatives for each block as well as the entire chip. The area of each block can be estimated after partitioning based on the number and type of components in that block. In addition, interconnect area required within the block must also be considered. Floorplanning is a critical step in physical synthesis as it sets up the groundwork for a good layout.

During placement, the blocks are exactly positioned on the chip. The goal of placement is to find a minimum area arrangement for the blocks that allows completion of interconnection between the blocks, while meeting the performance constraints. The quality of the placement will not be evident until the routing phase is completed. Good routing and circuit performance depend heavily on a placement algorithm. This is due to the fact that once the position of each block is fixed, very little can be done to improve the routing and the overall circuit performance. Late placement changes lead to increased chip area and lower quality designs.

**Routing** - The objective of the routing phase is to complete the interconnections between the blocks according to the specified netlist. Routing is typically performed in two phases: *global routing* and *detailed routing*. In global routing, connections are completed between the blocks of the circuit disregarding the exact geometric details of each wire and pin. The global routing specifies the different regions in the routing space through which a wire should be routed. Global routing is followed by detailed routing which converts the global routes for the nets into geometrical information such as the location and spacing of the wires and their layer assignments, location of the pins etc.
Compaction - Compaction is the task of compressing the chip layout in all directions such that the total area is reduced. By making the chip smaller, wire lengths are reduced, which in turn reduces the signal delay between components of the circuit. At the same time, a smaller area may imply more chips can be produced on a wafer, which in turn reduces the cost of manufacturing.

Extraction and Verification - After placement and routing, a design-rule check is performed on the layout to ensure that all the geometric patterns in the layout meet the design rules imposed by the fabrication process. After checking a layout for design rule violations, the functionality of the layout is verified by Circuit Extraction. The extracted netlist is compared with the input netlist to verify its correctness, performance, and reliability.

Physical design is iterative in nature and many of these design steps are repeated several times to obtain a good layout. In addition, the quality of results obtained in a step depends on the quality of solutions obtained in earlier steps. For example, a poor quality placement cannot be fixed by high quality routing. As a result, earlier steps have more influence on the overall quality of the solution. In this sense, partitioning, floorplanning, and placement play a significant role in determining the overall area and chip performance.

2.3 Impact of Elevated Chip Temperatures on Performance and Reliability

Elevated temperatures have an adverse impact on nearly all key performance metrics of a VLSI circuit, including packaging/cooling costs, circuit and system reliability, circuit speed, power dissipation, and integrity of an integrated circuit's power/ground network. Recent work suggests that, in high-performance integrated circuits, peak chip temperatures can rise up to 160°C in 90nm technology node, and is expected to rise further in future technologies [92]. Data shows that temperature-related failures currently account for more than 50% of all integrated circuit failures [62]. Thermal issues will have a much greater impact on the reliability and performance of
integrated circuits in future nanometer CMOS technologies due to increasing variability of key process parameters [66]. The impact of chip temperature on these metrics is discussed next.

### 2.3.1 Effect on Circuit Reliability

High temperatures can cause reliability problems in a VLSI circuit since they exacerbate electro-migration in metal wires, hot-carrier effects in transistors, and thermal-stress related gate-oxide breakdown in transistors [66].

Electromigration in metal wires occur when the current density in the wires becomes too high. Wires in a circuit that typically carry high current loads (such as the power/ground and clock networks) are particularly susceptible to electromigration failures. Electromigration is caused by the gradual migration of metal ions in a wire, induced by the current flowing through the wire. This migration of metal ions creates voids in the “upward” direction of the current flowing through a wire, and an accumulation of metal ions in the “downward” direction of current flow to form features called “hillocks” and “whiskers” [88]. The presence of voids in a wire creates “open-circuit” failures, while the accumulation of metal ions in wires leads to shorts between adjacent wires in a circuit.

The mean-time-to-failure (MTTF) due to electromigration is computed using the following equation [89]

\[
MTTF = AJ^{-n}e^{Q/kT}
\]

where \( A \) is a constant that is dependent on the process and layout geometry, \( J \) is the average current density, exponent \( n \sim 2 \) under typical operating conditions, \( Q \) is the activation energy for grain-boundary diffusion and is equal to \( \sim 0.7eV \) for Al–Cu, \( k \) is Boltzmann's constant, and \( T \) denotes the metal temperature.
From equation 2.1, we can observe that for a given current density $J$, the MTTF decreases exponentially with rising temperature. Conversely, the higher the chip temperature becomes, the lower is the tolerable current density. Temperature also limits the maximum allowable current density in wires to limit the temperature increase due to “self-heating” in wires. Though chip designers would typically design circuits to operate correctly and meet specifications within a tolerable temperature range, localized areas in a chip could heat up, creating thermal “hot-spots,” where the chance of electromigration-induced failures could increase exponentially, greatly reducing the lifetime of a chip.

In addition to its impact on the reliability of wires, elevated integrated circuit temperatures can also have an adverse effect on the reliability of transistors in nanometer CMOS technologies. The gate oxide thickness for sub-100nm CMOS is typically in the range of 1.2–1.6 nm [90, 91]. Such thin gate-oxides are particularly vulnerable to thermally-induced mechanical stresses, caused by on-chip thermal gradients, that lead to oxide breakdown and circuit failure [66]. This problem gets more severe with transistor scaling.

### 2.3.2 Effect on Circuit Performance

At the circuit level, elevated on-chip temperatures are known to increase transistor and interconnect delays and thereby degrade circuit performance [66, 93].

Transistor drive currents and switching speeds decrease with increasing junction temperatures [61, 94]. This decrease is due to carrier mobility degradation with rising temperatures. Mobility decreases as temperature increases mainly because of the increase in carrier scattering caused by thermal vibrations of the semiconductor crystal lattice [94].

Carrier mobility in a semiconductor is a function of electric field, doping concentration, and temperature. Transistor carrier mobility ($m$) at commercial integrated circuit operating temperatures ($25^\circ C$ ~ $100^\circ C$) can be modeled by equation 2.2.
where, \( n \approx 1.3 \) for electrons, and \( n \approx 1.2 \) for holes. It can be seen from equation 2.2 that for room temperature operation, carrier mobilities decrease exponentially with increasing temperatures. A direct consequence of this is a decrease in the drive current of transistors with rising temperatures. Transistor drive currents are very sensitive to substrate temperatures, significantly decreasing with rising temperatures. Simulations using HSPICE have shown that gate delays for basic logic gates (NAND, NOR, XOR) in the 90-nm technology node, can increase by as much as 12% for a temperature rise of 60°C [95].

On-chip temperature rise is known to increase interconnect delays, thereby degrading circuit performance [93]. The resistivity of a metal wire increases linearly with increase in its temperature, thereby increasing interconnect resistance (and consequently the signal propagation delay). In addition, the increase in the resistivity of the interconnect leads to an increase in interconnect Joule heating that causes an additional temperature rise in the interconnect [93]. This trend is likely to have a larger impact in the future, since wire cross-sections decrease with scaling, resulting in a corresponding increase in wire resistances and the effect of elevated chip temperatures on interconnect delays.

The existence of large on-chip thermal gradients can create nonuniform temperature profiles along the length of global interconnect lines, leading to nonuniform interconnect resistance profiles. The nonuniform resistance profiles of global interconnects will strongly impact many aspects of interconnect design and optimization, such as, wire sizing and spacing, buffer insertion, layer assignment, crosstalk effects, buffer insertion, and clock skew control in clock-trees [96].
2.3.3 Effect on Power Dissipation

In an integrated circuit, the main sources of heat generation are the power dissipation in transistors, and power dissipation from Joule heating (or self-heating) caused by the current flow in the interconnects. Typically, the power dissipation in transistors is the main contributor to total power dissipation. However, the current density in some parts of the interconnect network can be large enough to cause a significant temperature rise due to Joule heating, creating thermal-hotspots and on-chip temperature gradients [96], giving rise to timing failures and reliability concerns.

The power dissipation in the logic gates of a CMOS circuit can be expressed as

\[ P_{\text{total}} = P_{\text{dynamic}} + P_{\text{short-circuit}} + P_{\text{static}} \]

where \( P_{\text{dynamic}} \) denotes the dynamic power consumption that occurs when the output signal of a CMOS logic gate makes a transition. \( P_{\text{short-circuit}} \) represents the power dissipated when both n- and p-transistors are simultaneously conducting, creating a direct path between supply power and ground. \( P_{\text{static}} \) is the static power dissipation caused by static current drawn from the power supply, due to sub-threshold conduction current and direct gate current (collectively referred to as transistor leakage current) in CMOS transistors.

Dynamic power is due to signal switching activity at the output of a CMOS logic gate. Dynamic power in a CMOS circuit can be modeled as

\[ P_{\text{dynamic}} = a C_{\text{eff}} V_{\text{dd}}^2 f \]

where \( a \) is the switching activity factor, \( C_{\text{eff}} \) is the total effective output load capacitance of the circuit, and \( f \) denotes the operating frequency. The dynamic power component dominates during the active mode of a CMOS logic gate, and increases with increasing switching activity and
operating frequency. High dynamic power (resulting from the current trend of increasing transistor counts and operating frequencies) can significantly increase on-chip temperatures.

Short-circuit power is due to direct current flow from the power supply to the ground, and occurs when the pull-up and pull-down transistor networks in a CMOS circuit are conducting simultaneously. Short-circuit current depends on the transistor sizes, supply voltage level, and the duration of the simultaneous on-times of the pull-up and pull-down networks [97]. The short-circuit current is found to depend on the carrier mobility and threshold voltage of the transistors both of which vary with temperature [94]. Short-circuit power can be minimized through careful circuit design [97].

Static power dissipation is due to sub-threshold conduction current (referred to as sub-threshold leakage current) and direct gate current (referred to as gate-leakage current). Dynamic and static power are the two main sources of power consumption in CMOS circuits. In many high-performance CMOS circuits, leakage power forms a significant fraction of the total chip power consumption. For example, the contribution of leakage power to the total power dissipation of a high-performance circuit in 90-nm CMOS technology can be up to 40% or higher [98]. Sub-threshold leakage current increases rapidly with technology scaling due to continuous reduction in supply voltage, which necessitates reduction of the threshold voltage to meet performance requirements. In addition, gate leakage current increases with technology scaling due to a reduction of gate-oxide thickness with scaling. Simulation results indicate that leakage current in CMOS transistors increases by a factor of three to five per technology generation [99].

In nanometer-CMOS, the major leakage components are [100]:

- sub-threshold leakage,
- gate-oxide tunneling leakage, and
- reverse-bias drain-substrate and source-substrate band-to-band tunneling leakage.
There are other leakage components such as gate-induced drain leakage (GIDL), and punch-through current, but their contribution to the total leakage current is not significant at normal modes of operation. In current CMOS technologies, the sub-threshold leakage current is much larger than the other leakage current components.

Sub-threshold leakage current is a function of transistor threshold voltage and substrate temperature. According to the BSIM3v3.2 MOSFET transistor model [101], the sub-threshold drain current of a transistor in the normal “off” state can be modeled as:

\[ I_s = I_0 \left( 1 - e^{\frac{V_{ds}}{V_T}} \right) e^{\frac{V_{gs} - V_{th} - V_{offset}}{nV_T}} \]  

\[ I_0 = k_{tech} \left( \frac{W}{L} \right) V_{th}^2 \]  

\[ V_T = \frac{kT}{q} \]

where \( V_{th} \) is the transistor threshold voltage, \( V_{ds} \) is the drain-to-source voltage, \( V_{gs} \) is the gate-to-source voltage, \( V_T \) is the thermal voltage, \( V_{offset} \) is the offset voltage, \( n \) is the sub-threshold voltage swing coefficient, \( k_{tech} \) is a CMOS technology-related parameter, and \( W/L \) is the transistor aspect ratio. From equations 2.5, 2.6, and 2.7, we get the equation 2.8, that relates the variation in sub-threshold leakage current to substrate temperature:

\[ I_s \propto T^2 e^{\frac{qV_{th}}{kT}} \]

Equation 2.8 shows that the sub-threshold leakage current varies exponentially with the temperature and transistor threshold voltage.

The threshold voltage of a transistor is also a function of temperature. Equation 2.9 models the threshold voltage of a CMOS transistor when \( V_{ds} \) is small,
\[ V_{th} = V_{fb} + 2Q_b + k_{tech} \]  \hspace{1cm} \text{(2.9)}

where, \( V_{fb} \) represents the flat-band voltage, and \( Q_b \) is the potential difference between the Fermi level and the intrinsic level, and \( k_{tech} \) is a CMOS technology-related parameter, determined by permittivity of silicon, gate-oxide capacitance, and donor (or acceptor) concentration. For large \( V_{ds} \), however, \( V_{th} \) decreases further due to the effect of drain-induced barrier lowering.

In equation 2.9, both \( V_{fb} \) and \( Q_b \) are temperature dependent. \( V_{th} \) typically decreases at the rate of 1 mV per °K rise in temperature. Figure 2.4(a) illustrates the change in the threshold voltage of a 45-nm NMOS transistor [90] at different operating temperatures.

Figure 2.4(b) shows the relationship between leakage current and threshold voltage \( V_{th} \) for a 45-nm NMOS transistor [90]. The leakage current is normalized to the leakage when \( V_{th0} \) is fixed.

Figure 2.4  (a) Transistor Threshold Voltage at Different Operating Temperatures, (b) Sub-Threshold Leakage Current at Different Threshold Voltages, (c) Variation of Leakage Power with Temperature
at 0.25 V. We observe from Figure 2.4(b) that leakage current in a transistor increases rapidly with a decrease in $V_{th}$.

Rising chip temperatures significantly impact the static power dissipation of a CMOS circuit, primarily due to two reasons. First, transistor sub-threshold leakage current increases exponentially with temperature and transistor threshold voltage. Secondly, the threshold voltage itself decreases with rising temperatures, further increasing the sub-threshold leakage current.

Figure 2.4(c) plots the transistor sub-threshold leakage current as a function of temperature. In the figure, the leakage current is normalized to the leakage current at 40°C.

Sub-threshold leakage current increases exponentially with increase in temperature, leading to an exponential increase in static power dissipation in CMOS circuits. This coupling between static power and rising temperature can lead to thermal runways [66].

2.4 Related Work on Interconnect-Aware High-Level Synthesis

The primary focus of high-level synthesis in the past was optimizing logic (i.e., functional units such as adders, multipliers, registers, and multiplexers), and wire delays could be ignored without loss of accuracy. However, with process scaling, wire delays have become significant, shifting the focus of VLSI design from transistors to interconnects [5]. For example, in [57, 116], the authors show that interconnect delays can contribute an additional 20% to the clock period of a HLS design, and conclude that high-level synthesis tools must take physical effects into consideration if they are to produce high-quality designs.

Taking interconnect costs into consideration during high-level synthesis has attracted significant attention in HLS research. In some of the early work on HLS, a simple estimate of the interconnect cost was determined by counting the number of wires and multiplexers required by a design [30, 31, 32, 33, 35]. These estimates did not use any physical-level information and were used mainly to compare the wiring complexity of different datapath design alternatives.
However, when interconnect delays began to be comparable to (and even exceed) gate delays, these simple estimates were inadequate, since it was no longer possible to accurately predict the performance of a design without first knowing enough about its floorplan and the structure of its interconnect. This dramatically complicated both design and synthesis.

Traditionally, high-level synthesis consists of the three main steps – scheduling, allocation, and binding, which are typically performed in a sequential manner. Interconnect-awareness during high-level synthesis adds an additional step: interconnect optimization. An optimal solution may not be found by applying these steps sequentially because the optimizations are not independent. Consequently, because interconnect-aware high-level synthesis requires a detailed understanding of the target architecture, heuristics are needed to synthesize such architectures under simultaneous consideration of scheduling, allocation, binding, and floorplanning.

A number of researchers have proposed high-level synthesis techniques, that utilize some form of layout information, such as the chip floorplan, to guide synthesis [15, 16, 17, 34, 37, 38, 55, 56, 57, 59, 125, 126]. In these approaches, the floorplanning information is used to provide an estimate of the chip area, interconnect length, delay or power.

BUD [55] and Fasolt [56] use floorplans to analyze the area impact during high-level synthesis. Chippe [125] is based on a constraint-driven expert system that predicts wire delays from structural RT-level descriptions. However, all of these techniques separate the floorplanning and high-level synthesis steps.

Weng and Parker [57] attempt to minimize interconnect delay by assigning DFG operations to modules that are close to the modules bound to their predecessors. The approach is greedy in that the operations on the critical path are scheduled and bound first. In addition, they do not consider the cost of registers or multiplexers.

Tarafdar and Leeser [34] incorporated layout into scheduling and binding of data transfers in high-level synthesis. The binding process is based on a branch-and-bound heuristic. Um et al.
transform the binding and module placement problem into a problem of rectangle packing with distance constraints, and use a linear programming (LP) formulation to solve the problem.

Other approaches have combined some or all of the HLS steps along with a floorplanner, into one optimization loop, such as simulated annealing. These approaches improve a preliminary design by applying a set of moves, and evaluating the impact of these moves using some form of a floorplanner. Fang and Wong [15] apply a set of floorplanning moves and use a binding heuristic to assign DFG operations to floorplan modules before evaluating the cost. Prabhakaran et al. [16] apply moves changing the schedule and the binding. Before evaluating the cost function, they perform a floorplanning step during each iteration. Zhong and Jha [17] use allocation and binding moves followed by a floorplanning step for cost estimation. Stammerman et al. [37] include allocation, binding, and floorplanning moves into their optimization heuristics.

Davoodi and Srivastava [38] propose an iterative algorithm that combines binding and floorplanning to minimize interconnect delays. They use a probabilistic wirelength estimation model to account for inaccuracies in traditional wirelength estimation methods that use half-perimeter bounding box or minimum spanning tree to model nets interconnecting floorplan modules.

All of these interconnect-aware high-level synthesis approaches use a loosely-coupled floorplanner, where the floorplanning and synthesis decisions are made independent of each other. However, Gu et al. [59] have shown that tightly coupling high-level and physical synthesis and simultaneously performing HLS and floorplanning, shows significant improvements in run times and solution quality. In light of this, in this dissertation, we propose unified incremental physical-level and high-level synthesis techniques for interconnect-aware high-level synthesis.

Previous work on interconnect-aware high-level synthesis only handled 2-D circuits. Very little has been reported on high-level synthesis systems aimed at 3-D layouts. To the best of our knowledge, only Mukherjee and Vemuri [127] have addressed high-level synthesis for 3-D
integrated circuits. However, their approach separates the high-level synthesis tasks from the floorplanning step. In Chapter 3, we present a unified physical-level and high-level synthesis algorithm, which enables maintaining physical-level properties across consecutive physical estimations during behavioral synthesis. Since the physical design space of 3-D layouts is much larger than that of planar layouts, tightly integrating the high-level and layout-level phases of synthesis is necessary to ensure convergence of the synthesis flow. The benefits of this integration increases with increasing problem size and complexity.

No prior work exists in literature that accounts for the topology of nets resulting from resource binding decisions during high-level synthesis. In Chapter 4, we propose a novel floorplan-aware high-level synthesis technique that uses accurate net topologies and distributed wire-delay models to guide resource allocation and binding decisions during design-space exploration.

2.5 Related Work on Temperature-Aware High-Level Synthesis

Several researchers have addressed the problem of thermal modeling and temperature-aware design. Micro-architecture level thermal models were proposed in [69, 70]. The authors of [71, 72] consider thermal effects during processor-based micro-architectural design space exploration. Thermal issues have also been considered during physical level design such as standard-cell placement [73, 74], floorplanning [75], and routing [76].

Recently, researchers have addressed thermal issues during high-level synthesis. Related works in high-level synthesis have considered thermal effects during resource binding [77, 78] and scheduling [80]. Gu et al. [81] use floorplanning and voltage islands to alleviate thermal hot-spot formation during high-level synthesis.

Mukherjee et al. [77] propose an analytical model that relates the switching activity of datapath functional units to their expected heat dissipation, and use this in binding algorithms for temperature-constrained resource minimization and resource-constrained temperature minimization.
tion. The analytical model used in their approach does not utilize any geometry or floorplan information to guide temperature estimation of functional units. This makes their approach inaccurate since ignoring floorplan information when estimating module temperatures fails to capture lateral heat dissipation among the modules through the silicon substrate.

In [78], Mukherjee et al. start with an optimal low-power binding solution, and apply an iterative rebinding algorithm to create a solution with uniform switching activity distribution among the datapath functional units. Their work is based on the observation that a traditional low-power binding could lead to an uneven distribution of switching activity among the datapath functional units, resulting in some functional units switching more than others, dissipating more power, and potentially becoming thermal-hotspots. Their temperature-aware binding algorithm redistributes tasks among the functional units with the goal of creating an even power dissipation among the resources. The main drawback of their approach is that they base all their re-binding decisions on functional unit switchings instead of module temperatures, which could lead to significant inaccuracies. In addition, their algorithm does not use any floorplan or module geometry information to guide their temperature minimization moves, making their approach inaccurate.

The work proposed by Mukherjee and Memik in [80] integrates task scheduling, resource allocation and assignment, and post-floorplan thermal simulation into a thermal-aware high-level synthesis framework. This work addresses some of the deficiencies of [77] and [78] by using floorplanning information during temperature minimization. This work starts with an initial low-power solution, which is then used to create a thermal-aware floorplan using HotFloorplan [102], a thermal-aware floorplanner. A series of re-scheduling and re-binding moves are then applied to this initial solution, in an attempt to redistribute the switching activity among the functional units and mitigate thermal-hotspots. After each move, on-chip temperature profiles are obtained using the HotSpot simulator [70]. Though the authors of [80] use accurate estimates of module
temperatures, they restrict their search to the HLS sub-space of task schedules and resource bindings, without modifying the initial floorplan. Since on-chip thermal profiles can change significantly with changes in the floorplan, by not exploring different floorplans, the algorithm proposed in [80] could miss many opportunities for temperature minimization.

In [81] Gu et al. propose a thermal-aware floorplanning high-level synthesis system that makes use of integrated high-level and physical-level thermal optimization techniques. They use multiple voltages and voltage partitioning of the tasks in order to reduce a design's power consumption and peak temperature. However, much of the peak temperature minimizations in their technique were obtained through the use of multiple voltages for power minimization.

Our work differs from these in several respects. Our approach tightly integrates high-level and physical-level synthesis steps at all stages of synthesis, to provide accurate estimates of thermal distribution. We concurrently explore the sub-spaces of operation scheduling, resource binding, and floorplanning, in an attempt to minimize module temperatures during behavioral synthesis. During search, we also minimize peak and average switching power of modules, to mitigate thermal hotspots. Unlike the works of [77], [78], and [80], in addition to minimizing module switching power, we also explore the impact of changes to the floorplan, in an effort to minimize chip temperatures.
CHAPTER 3

LAYOUT-AWARE BINDING FOR THREE-DIMENSIONAL INTEGRATED CIRCUITS

Recent progress in the fabrication of three-dimensional integrated circuits has opened up the possibility of exploiting this technology to alleviate performance and power related issues raised by interconnects in nanometer CMOS. Physical synthesis for three-dimensional integrated circuits is substantially different from traditional planar integrated circuits due to the presence of additional constraints of placing circuit blocks in multiple die. To realize the full potential offered by three-dimensional integrated circuits, high-level synthesis (HLS) of these circuits must take layout-related issues unique to 3-D technology into account during the synthesis process. This chapter presents a 3-D layout aware binding algorithm for high-level synthesis that tightly integrates the synthesis tasks of resource binding, assignment of modules to multiple die, 3-D floorplanning, and inter-die via minimization. Since floorplanning and resource binding are interdependent, the algorithm can significantly outperform traditional high-level synthesis flows that separate these tasks. This chapter is organized as follows.

Section 3.1 provides an introduction to three-dimensional (3-D) integrated circuit technology and outlines the related work. Section 3.2 discusses the impact of 3-D integration on layout-aware high-level synthesis. Section 3.3 describes the problem of HLS for 3-D integrated circuits. Section 3.4 discusses the 3-D layout-aware binding algorithm proposed in this chapter. Section 3.5 presents the experimental results and Section 3.6 concludes the chapter.
3.1 Introduction

Aggressive scaling of process technologies has enabled feature sizes to shrink to nanometer regimes. While the performance of gates has been improving, interconnects have become a major performance bottleneck and a major source of power consumption because global interconnects do not scale accordingly. To keep delays of global interconnects within bounds, repeaters and flip-flops are inserted to prevent performance degradation. However, use of these additional components increase interconnect power dissipation. Consequently, interconnect-centric design methods have garnered much attention in recent years. While there have been significant advances in interconnect materials such as the use of copper and low-K dielectrics, other technologies have been actively explored to address the interconnect problem, such as the development of three-dimensional (3-D) integrated circuits [45, 46, 48].

In a three-dimensional integrated circuit, multiple device layers are stacked together with direct vertical interconnects interfacing the device layers. Figure 3.1 shows an example of a 2-layer 3-D integrated circuit. The direct vertical interconnects are called die-to-die vias (d2d), inter-strata vias, or 3-D inter-layer vias. Compared to a traditional 2-D integrated circuit, a key

Figure 3.1 Example of the Implementation of a 3-D IC using Through-Silicon Vias
Figure 3.2  Wirelength Reduction with 3-D Integration (a) Planar Circuit   (b) 3-D Integrated Circuit

Figure 3.3  Wirelength Distribution of 3-D IC as a Function of Number of Device Layers Used (from [134])
benefit of 3-D integration is a reduction of global wirelengths. Figure 3.2 illustrates this through a simple example. Figure 3.2(a) shows 2-D circuit with a global wire connecting two blocks. Figure 3.2(b) shows a 3-D implementation of this circuit where the planar global interconnect has been replaced by a short inter-layer via. Rahman and Reif [134] have performed a stochastic analysis of wirelength distributions in a general circuit and found that for a wide class of 3-D integration technologies, the wirelength distribution shifts in response to an increase in the number of device layers as shown in Figure 3.3. The leftward shift in wirelength distribution is an indication of the wirelength reduction trend that can be achieved through the use of multiple active layers in 3-D integration.

3.1.1 3-D Integration Technologies

A number of fabrication techniques for three-dimensional integrated circuits have been proposed recently. They span the continuum - from integrating multiple transistor layers such as multi-layer buried structures [129], to 3-D bonding technologies such as wafer-to-wafer, wafer-to-die, and die-to-die bonding [130, 131]. All of these fabrication techniques involve stacking multiple device layers (i.e., silicon dies) and forming interconnects between the layers. Various vertical interconnect technologies have also been studied, including wire bonded, micro-bump, contact-less (capacitive and inductive), and through-silicon interconnects. A survey of these interconnect technologies can be found in [45]. Among these, through-silicon vias (3-D vias) offer the greatest vertical interconnect density and therefore the most promising among the vertical interconnect technologies.

Multi-layer buried structures (MLBS) 3-D technology involves successively (sequentially) fabricating multiple device layers on a silicon substrate. Layer-to-layer connections are made from either inter-layer vias or from direct interconnects between transistors. The main advantage of MLBS technology is that the 3-D inter-layer vias can potentially scale with feature size due to
use of local polysilicon wires for connection [132]. However, MLBS technology requires extensive changes to existing manufacturing processes [47].

Currently, 3-D bonding technology is the most promising approach to fabrication of three-dimensional integrated circuits since it is compatible with standard lithography, processing, and wafer handling techniques [47]. In this approach, each active device layer is processed separately, using conventional fabrication techniques. Then multiple device layers are stacked to build up a 3-D IC, using techniques such as wafer-bonding [47]. Different wafer-bonding techniques have been proposed, such as oxide-to-oxide bonding [133], copper-to-copper bonding [47], and dielectric adhesive bonding [133]. After each wafer is processed individually, wafers can be bonded face-to-face (F2F) or face-to-back (see Figure 3.1).

Figure 3.1(a) shows a 3-D integrated circuit using a face-to-face wafer bonding technology. A typical copper-to-copper bonding approach to create the 3-D structure of Figure 3.1(a) involves depositing 3-D vias on the top metal layers of each of the two wafers, aligning the two wafers, and bonding them together using thermo-compression. Under thermo-compression, the copper vias fuse together providing both the die-to-die interconnection as well as a physical mechanism to hold the 3-D structure together. After bonding, one of the wafers is thinned with chemical-mechanical polishing (CMP) down to about 10um allowing low impedance backside vias to be etched through, to provide signal and power/ground connections. From an interconnect perspective, an advantage of face-to-face bonding is that the 3-D inter-layer vias can be made very small since they do not go through a thick silicon device layer, allowing for very high via densities.

Figure 3.1(b) shows a 3-D integrated circuit using a face-to-back bonding topology. The face-to-back bonding requires etching vias through the backside of a silicon layer (backside vias). This causes the cross-sectional area and length of the backside vias to increase relative to face-to-face vias. In addition, the backside vias must pass through the active region of a device layer,
which may disrupt the layout of transistors. Hence, a face-to-back structure cannot support the same levels of via densities that the face-to-face structure can potentially offer.

When extending 3-D integration to more than two active layers, the bonding process may be repeated in combinations of face-to-face, face-to-back, and back-to-back arrangements.

### 3.1.2 3-D Inter-Layer Vias

One of the key technology parameters in 3-D integration is the size of the vertical 3-D vias. In MLBS technology the 3-D inter-layer vias can potentially scale with feature size due to use of local polysilicon wires for connection [132].

When using wafer-bonding, the dimensions of the 3-D vias are not expected to scale at the same rate as feature size, because wafer-to-wafer alignment tolerances during bonding pose limitations on the scaling of the vias [47]. In wafer-bonding, the arrangement of the device layers in a 3-D structure (F2F or F2B) determines the pitch and quality of the 3-D vias implementable between the layers, and whether the vias interrupt the device layers. In a F2F organization, the 3-D vias only need to cross the distance separating the top two metal layers on adjacent dies, with typical via heights less than 5um, depending on the technology. As mentioned earlier, the individual die in a 3-D IC are thinned to approximately 10um to reduce the distance 3-D vias must cross to connect adjacent active layers. Due to this thinning, even in a F2B arrangement, the 3-D via heights are typically less than 25um [132]. A 3-D inter-layer via is much smaller than the planar interconnect it replaces, thus reducing both the interconnect resistance ($R$) and capacitance ($C$). This leads to significant improvements in signal propagation delays due to reduced interconnect $RC$ characteristics of 3-D inter-layer vias. For example, the authors of [135] report that for simulations using a 70nm technology, the $RC$ interconnect delay for driving a signal through a 10um long F2F 3-D via was only 8ps. This is in contrast to driving a signal through a 1mm minimum width metal wire on a planar IC, where they report a delay of 225ps.
Current implementations of 3-D integrated circuits support F2F 3-D via pitches in the range of about 3um to 10um. As alignment technologies continue to improve, via sizes smaller than 1um may soon be practical. The backside vias (used in F2B bonding) require etching through the bulk silicon substrate and hence require larger structures in current technologies. Typical F2B via sizes vary from 1um x 1um to 10um x 10um [47].

3.1.3 Benefits of 3-D Integration

The reduction in the wirelength due to 3-D integration can bring the following two benefits:

*Performance improvement* – Higher performance can be achieved due to reduced average interconnect length and critical path length.

*Power Reduction* – Interconnect power consumption becomes a large portion of the total power consumption as technology scales. The wirelength reductions from 3-D integration translates to corresponding savings in interconnect power.

3-D integration allows higher packing density and enables significant reductions in circuit footprints. This reduction has implications in the yield of the integrated circuits. For example, smaller footprints enable a larger number of dies to be fabricated on a single wafer, resulting in higher yield per wafer.

Another benefit of 3-D integration is the ability to support the realization of mixed-technology chips by combining dies fabricated with different processes into a 3-D wafer stack. For example, rather than optimizing a single process technology to fabricate both logic and RF circuits on the same die, separate dies containing the logic and RF circuits could be fabricated using processes optimized individually and the dies could then be stacked using 3-D integration.
3.1.4 Related Work

The prior work related to this chapter can be divided into three groups: system-level analysis tools for 3-D integrated circuits, physical synthesis tools for 3-D integrated circuits, and architectural-level exploration tools. A brief overview of these works follows:

System-level analysis tools developed analytical and numerical models to predict the impact of 3-D integration. The works [134, 136] examine wirelength reductions and performance improvements of 3-D ICs. The impact of 3-D integration on interconnect power was studied in [137]. The performance, energy, and thermal impact of 3-D designs under a given timing constraint was studied in [138].

Physical synthesis tools for 3-D ICs have been proposed for floorplanning, standard-cell placement, and routing in 3-D integrated systems. Das et al. [48] have proposed custom layout and placement tools for 3-D ICs. Thermal driven floorplanning, placement, and routing algorithms for 3-D ICs have been proposed by various groups [74, 75, 128, 139]. The use of thermal-vias to manage thermal issues in 3-D ICs was studied in [140]. Reliability-aware floorplanning for 3-D ICs was examined in [141].

Several groups have studied the impact of 3-D integration at the microarchitecture level, especially the trade-offs in microprocessors. Cong et al. [142] proposed a design flow for 3-D microarchitecture evaluation. Black et al. [143] examined the benefits of implementing an Intel IA32 processor in 3-D technology. Loh et al. [135] studied the architectural issues involved in the design of processors in 3-D die stacking technologies.

Previous work on layout-aware high-level synthesis only handled 2-D circuits [15, 16, 17, 34, 36, 37, 38]. Very little has been reported on high-level synthesis systems aimed at 3-D layouts. To the best of our knowledge, only Mukherjee and Vemuri [127] have addressed high-level synthesis for 3-D integrated circuits. However, their approach performs the tasks of high-level
synthesis and floorplanning sequentially. Since high-level synthesis and floorplanning are strongly inter-dependent, this separation of the tasks makes their approach less than optimal.

In this chapter, we propose a unified physical-level and high-level synthesis algorithm that performs floorplan-aware resource binding for interconnect minimization. One of the key features of our work is the tight coupling between behavioral and physical synthesis, which enables maintaining physical-level properties across consecutive physical estimations during behavioral synthesis. Since the physical design space of 3-D layouts is much larger than that of planar layouts, tightly integrating the high-level and layout-level phases of synthesis is necessary to ensure convergence of the synthesis flow. The benefits of this integration increases with increasing problem size and complexity.

3.2 Impact of 3-D Integration on High-Level Synthesis

The additional freedom of placing circuit blocks in the z-direction makes 3-D layouts substantially different from traditional 2-D layouts. In addition to the traditional optimization objectives of chip area and wirelength, 3-D floorplanning must also address issues of optimally placing circuit blocks on multiple die, minimizing the number of die-to-die vias, and ensuring that the final packed areas of all dies in a 3-D integrated circuit match. These additional constraints increase the complexity of floorplanning 3-D layouts, when compared to floorplanning of planar integrated circuits.

Attributes of a 3-D layout that affect the performance metrics and interconnect delays of a design are

- number of die vertically stacked in a 3-D IC,
- the length and sizes of die-to-die vias used,
- the die to which interconnected modules are assigned, and
- the floorplanning of the modules in each die.
In deep submicron CMOS technologies, wire delays dominate gate delays. The location of individual modules on a 3-D floorplan determine interconnect delays, which in turn are determined by the placement of modules on the 3-D floorplan, and the 3-D fabrication technology used.

The assignment of modules to individual die in a 3-D IC has a significant impact on circuit performance. Interconnect delays associated with nets connecting vertically stacked modules placed in two adjacent die are much smaller than those of nets connecting modules placed far apart on the same die, or between modules separated by several intermediate die in the 3-D integrated circuit. Module placement determines the vertical overlap of modules on different die. To fully exploit the advantages offered by 3-D technology, an efficient 3-D placer must maximize vertical stacking among interconnected modules on the critical path.

In this chapter we investigate the the impact of 3-D layout technology on datapath designs during high-level synthesis. Since the resource binding subtask of high-level synthesis and the placement of RTL modules on a layout are strongly interdependent (as shown in section 3.2.1), it is reasonable to direct our attention to resource binding. Our algorithm simultaneously performs the tasks of floorplanning, layer assignment, and resource binding, tightly integrating these tasks.

### 3.2.1 Motivating Example

Figure 3.4(a) shows a dataflow graph (DFG) fragment, and the dashed lines show the two critical paths in the design. If operations 1 and 4 are both bound to the same multiplier $M1$, then by placing $M1$ in one silicon-layer and the adders $A1$ and $A2$ in an adjacent silicon-layer (as shown in Figure 3.4(a)), it is possible to reduce interconnect lengths significantly and also reduce the number of 3-D vias. On the other hand, exchanging the resource bindings for operations 4 and 6 increases the number of 3-D inter-layer vias, which may be a disadvantage with backside vias.
Figure 3.4 Motivating Example of Resource Binding in a 3-D Integrated Circuit

Figure 3.5 3-D Chip-Package Structure (from [179])
Figure 3.5 illustrates a typical 3-D IC chip and package design. One side of the 3-D IC is connected to the carrier layer, that is attached to the circuit board. The other side of the IC is connected to a cooling solution such as a heat-sink. The primary heat dissipation path is from the silicon layers through the heat-sink to the ambient. Modules in the silicon layers closer to the heat-sink have lower thermal resistance to the ambient. Hence, for better thermal management in 3-D ICs, modules with higher power consumption must be placed in silicon layers closer to the heat-sink.

For the motivating example in Figure 3.4, since multipliers consume more average power than an adder or a subtracter, the assignment of modules to silicon layers as shown in Figure 3.4(c) could result in higher temperatures in layers farther from the heat-sink, leading to problems created by elevated chip temperatures, as discussed in Chapter 2. For the arrangement in Figure 3.4(b), if executing operations 1 and 4 on same multiplier leads to a higher average switching power, it is thermally more efficient to bind these operations to multiplier \( M1 \) rather than \( M2 \). On the other hand, if executing operation 6 consumes more power, it should be bound to multiplier \( M1 \) instead of \( M2 \), for a better thermal profile in the 3-D IC.

3.3 Problem Description

The input to the algorithm is a scheduled dataflow graph (DFG), and an allocated set of resources. It is assumed that a library of components to be used for implementing the datapath is available. The output of the algorithm is an RTL binding and its corresponding 3-D floorplan. The algorithm concurrently optimizes the following objectives:

- footprint area,
- the total wirelength,
- the difference in floorplan dimensions among individual dies in the 3-D IC, and
- the number of die-to-die vias.
The footprint area of a 3-D floorplan is computed by determining the maximum dimension (width and height) among all dies in the 3-D integrated circuit. The need for matching the dimensions of all the dies in a 3-D integrated circuit fabricated with wafer-stacking technology necessitates minimizing the difference in floorplan dimensions among individual dies, to avoid penalties of chip area. For example, assuming two layers, L1 and L2, if the height of L1 is larger than L2, and similarly if the width of L2 is larger than that of L1, the need for matching layer dimensions to aid manufacturing would result in a significant portion of the silicon area to be wasted.

In 3-D integrated circuits, minimizing the number of inter-die vias is important because inter-die vias impact chip yield, routing congestion, and signal delays.

### 3.4 Layout-Aware Binding Algorithm

The overall structure of the framework within which our algorithm operates is shown in Figure 3.6. The system accepts a scheduled data flow graph (DFG) and a resource allocation for the schedule. A compatibility graph for each resource type is then extracted from these two, and provided as an input to a Simulated Annealing (SA) based 3-D layout-aware binding algorithm.

Our technique is an SA-based iterative improvement algorithm that simultaneously performs a search for optimal module bindings and 3-D floorplans. We chose an SA-based approach primarily due to its superior performance when applied to floorplanning [75]. Since our algorithm aims to tightly integrate binding and floorplanning, the search for optimal bindings was also implemented as moves in the SA-framework. The SA uses two interleaved sequence of moves that alternate between a chain of binding moves and floorplanning moves. This allows the algorithm to perform a neighborhood search of the binding space, followed by a neighborhood search of floorplan space. The number of floorplanning and binding moves attempted by the SA
at every temperature can be set independently, and their values are chosen based on the size the input data flow graph, and the number of modules in the floorplan.

Changes in bindings can significantly affect the netlist topology in a datapath, and thus the resulting floorplan and wirelength statistics. Since our algorithm performs a neighborhood search of the floorplan space immediately following a neighborhood search of the binding space, any

Figure 3.6 Overall Flow of 3-D Layout-Aware Binding for High-Level Synthesis
changes in the netlist topology are immediately reflected in the resulting floorplan. Due to this incremental floorplan update feature, the SA need only modify the current floorplan with every binding move sequence, without the need for building a floorplan from scratch, thus making the algorithm very efficient.

3.4.1 Solution Representation

This section details the representation used by our SA to simultaneously search the HLS space of RTL bindings and the layout-level space of floorplans.

3.4.1.1 Representation of 3-D Floorplans

The floorplanner used in this work is based on the sequence pair representation proposed by Murata et al. [118]. We extend their representation to handle 3-D floorplans. The RTL modules of the datapath are distributed over the k floorplan layers available in a 3-D structure. To perform the 3-D placement, we maintain a multi-level sequence pair data structure, with one sequence pair for each floorplan layer of the 3-D integrated circuit.

Five floorplan perturbation operators are used in our algorithm, as described below:

- Rotate a Module,
- Move a module within the same floorplan layer,
- Swap two modules within the same floorplan layer,
- Move a module from one floorplan layer to another,
- Swap two modules located in different floorplan layers.

The first three moves are designed for a traditional 2-D floorplanning problem, and hence only affect the floorplan in a single layer. The last two moves are used to explore the space of 3-D floorplans.
3.4.1.2 Representation of Module Bindings

For this work, we assume point-to-point multiplexer based interconnections among the data path modules. The module bindings are determined by the compatibility graph for each RTL module type derived from the scheduled DFG, and the allocated number of RTL resources. While the number of allocated resources is determined prior to module binding, the number and types of multiplexers can only be determined after the binding step. The number and types of multiplexers change with different bindings. Since the area and wiring overheads of the multiplexers can be significant, the binding and floorplanning steps are strongly inter-dependent. In 3-D layouts, the layer assignments of modules and their bindings also strongly influence the number of through-die vias needed to interconnect the modules. To determine feasible bindings for a given schedule, the SA maintains a compatibility graph for each resource type. All binding related SA moves are guided by this compatibility graph, to ensure that all bindings determined by the SA are legal.

Five binding moves are used to explore the HLS binding search space:

• Reassign binding of a DFG operation,
• Reassign binding of a DFG variable,
• Swap compatible DFG operation bindings,
• Swap compatible DFG variable bindings,
• Swap inputs of a commutative DFG operation.

In all of these moves, if the number of sources at the input of a resource (module or register) changes as a result of a change in the binding, multiplexers can vanish, appear, or change their input sizes. Any changes in the number and types of multiplexers as a result of a binding move is immediately reflected in the floorplan.
3.4.2 Cost Function

Our floorplan-driven binding algorithm concurrently optimizes three different metrics, namely, the chip area, the total wirelength, and the number of through-die vias. In addition, the final packed area of each floorplan layer in the 3-D integrated circuit must match, dictated by the need for the dimensions of all dies in a 3-D integrated circuit to match. To ensure this, we use the concept of dimension deviation $d_{ev}(F)$ from [15]. Here, $d_{ev}(F)$ represents the deviation of the upper-right hand corner of a floorplan layer from the average $A_{vx}$, $A_{vy}$ values. The $A_{vx}$ value is computed as $\sum u x(f_i) / k$, where $u x(f_i)$ is the x-coordinate of the upper right-hand of floorplan $i$ and $k$ represents the number of layers. $A_{vy}$ is calculated in a similar manner. Thus, $d_{ev}(F)$ is formulated as $\sum_{i=1}^{N} (|A_{vx} - u x(f_i)|^2 + |A_{vy} - u y(f_i)|^2)$.

The following cost function is used to evaluate solutions:

$$\alpha * (A_{Norm} + d_{evNorm}) + \beta * W_{Norm} + \gamma * V_{Norm}$$

where, $A_{Norm}$ is the normalized footprint area of the 3-D floorplan, $W_{Norm}$ is the normalized total wirelength, and $V_{Norm}$ is the normalized inter-layer via count. The terms are normalized with respect to the best footprint area, total wirelength, and via count values seen during search, which are dynamically updated during search, where $\alpha$, $\beta$, $\gamma$ are user defined normalizing coefficients.

Using a representative HLS benchmark, we performed an extensive set of experiments to determine a suitable combination of values for the weights used in our cost function. The values for each of these weights were changed in increments of 10, and the combination of weights that yielded the best results over an average of five runs was used for the rest of our experiments. In our experiments, we set $\alpha = 25$, $\beta = 50$, $\gamma = 30$.

To estimate the total wirelength, we use the widely used method of half-perimeter bounding box model. Minimizing the total wire length is complementary to finding minimal area.
implementations. This also guides the SA to exploit the additional placement freedom afforded
by the availability of multiple layers in a 3-D floorplan. Since wirelengths along the $z$-plane are
significantly smaller than average wire lengths on the $xy$-plane, the wire length minimization
metric is a very useful search parameter in minimizing average delays between RTL modules,
assuming that the delay introduced by through-die vias are small.

3.5 Experimental Results

The proposed algorithm was implemented in C and executed on a SunOS 5.9, UltraSPARC-4
650MHz, 256MB workstation. The RTL modules, used in our module library were created from
behavioral Verilog descriptions and converted to structural Verilog using Cadence BuildGates.
They were then mapped to a 180nm, 6-metal standard cell library, and placed and routed by
Cadence First Encounter. The netlists extracted from these layouts were then analyzed for timing
delays using Synopsis PrimeTime. The areas and delays from the actual layouts of these
characterized RTL macro-cells served as the inputs to our algorithm.

The proposed algorithms were validated on four benchmarks drawn from DSP applications,
namely - a 16-point FIR filter, a 8-point IIR filter, an Elliptic Wave (EWF) filter, and a 1-point
8 x 8 DCT filter. Each of these benchmarks was specified as a scheduled dataflow graph (DFG),
capturing the behavioral description of the architecture to be synthesized.

3.5.1 Results from 3-D Layout-Aware Binding

The 3-D implementations of the synthesized datapaths have the potential to substantially
reduce lengths of global nets, with corresponding reductions in both the latency and power
associated with long wires. In the results below, we compare the total wirelength, average
wirelength, and the length of the longest global net of a baseline 2-D/planar implementation with
that of 3-D implementations comprising of 2, 3, and 5 floorplan layers.
Table 3.1 presents the results of datapaths synthesized using our algorithm for 2-D and 3-D floorplans. In the table, column 1 lists the HLS benchmarks tested, while column 2 shows the number of device layers in a 3-D IC. Column 3 lists the footprint area of the 3-D floorplans, and column 4 shows the total wirelength of the 3-D placement of datapath modules. Column 5 reports the average wirelength of a design, and column 6 shows the longest net length in a 3-D floorplan. Column 7 shows the count of the number of 3-D inter-layer vias needed in the 3-D floorplan.

As the number of floorplan layers in a 3-D IC increases, there is corresponding reduction in the footprint area required to implement a design. A benefit of this reduced footprint area is a corresponding decrease in the length of nets on each die. Since wafers in a 3-D wafer-stack are

<table>
<thead>
<tr>
<th>Benchmark</th>
<th># of device layers</th>
<th>Footprint Area (mm²)</th>
<th>Total Wirelength (mm)</th>
<th>Average Wirelength (microns)</th>
<th>Maximum Wirelength (microns)</th>
<th># of 3-D inter-layer vias</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>1</td>
<td>1.82</td>
<td>465.2</td>
<td>338</td>
<td>1114</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1.02</td>
<td>218.8</td>
<td>246</td>
<td>916</td>
<td>896</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.72</td>
<td>284.4</td>
<td>202</td>
<td>612</td>
<td>1504</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.43</td>
<td>235.8</td>
<td>171</td>
<td>437</td>
<td>3392</td>
</tr>
<tr>
<td>EWF</td>
<td>1</td>
<td>0.64</td>
<td>256.2</td>
<td>286</td>
<td>864</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.29</td>
<td>178.8</td>
<td>207</td>
<td>571</td>
<td>992</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.24</td>
<td>163.5</td>
<td>197</td>
<td>437</td>
<td>2144</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.18</td>
<td>134.8</td>
<td>145</td>
<td>338</td>
<td>2272</td>
</tr>
<tr>
<td>IIR</td>
<td>1</td>
<td>1.11</td>
<td>250.2</td>
<td>312</td>
<td>612</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.68</td>
<td>150.8</td>
<td>205</td>
<td>465</td>
<td>448</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.39</td>
<td>146.5</td>
<td>199</td>
<td>451</td>
<td>864</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.29</td>
<td>119.9</td>
<td>163</td>
<td>388</td>
<td>1248</td>
</tr>
<tr>
<td>DCT</td>
<td>1</td>
<td>1.24</td>
<td>691.7</td>
<td>328</td>
<td>1229</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0.63</td>
<td>481.1</td>
<td>232</td>
<td>834</td>
<td>2624</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.47</td>
<td>444.7</td>
<td>214</td>
<td>821</td>
<td>4480</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.28</td>
<td>365.4</td>
<td>171</td>
<td>569</td>
<td>7712</td>
</tr>
</tbody>
</table>
Figure 3.7  Wirelength Trends for HLS Benchmarks with 1, 2, 3, and 5 Floorplan Layers
usually thinned to about 10um, vertical wires running between dies are very short. Hence, having more floorplan layers in a 3-D integrated circuit leads to successive reductions in wirelengths, a fact borne out by our experimental results shown in Table 3.1. Though 3-D technology in itself leads to significant reductions in wirelengths, we show in Section 3.5.2 that by a judicious choice of layout-aware binding, further reductions in wirelengths can be accomplished. Though using more floorplan layers in general leads to a reduction in wirelength, it also results in an increase in the number of die-to-die vias.
Figure 3.7 illustrates the variation in total wirelength, average wirelength, length of the longest global net, and the number of through-die vias, for varying number of floorplan layers in a 3-D integrated circuit. When compared to a baseline 2-D floorplan, the total wirelength of a 3-D floorplan decreases by 38% on average with two floorplan layers, and by 49% on average with five floorplan layers. Similarly, the length of the longest net decreases by 28% on average with two floorplan layers, and by 54% with five floorplan layers.

Figure 3.8 illustrates the cumulative wirelength distributions of interconnects extracted from the layouts of these designs. The cumulative wirelength distribution function gives the total number of interconnects that have a length less than or equal to a value $l$. In the figure, the $y$-axis represents the total number of wires in a floorplan that have a length less than or equal to a corresponding wirelength on the $x$-axis. It can be observed from Figure 3.8 that as the number of floorplan layers increases, the cumulative wirelength distribution plots tend to shift to the left, due to a decrease in the average wirelength of the global nets.

### 3.5.2 Comparison with 3-D Layout-Unaware High-Level Synthesis

One of the essential features of our algorithm is that it implements a layout-driven binding algorithm for 3-D integrated circuits. Migrating a 2-D implementation of a design to a 3-D implementation naturally leads to significant reductions in wirelengths. However, we argue that to derive the full benefit of 3-D integrated circuit technology for potential reductions in wirelengths, high-level synthesis must be 3D-layout aware. Previous work has shown the advantages of using a layout-aware synthesis flow for high-level synthesis of 2-D/planar integrated circuits. We contend that with the increased layout constraints for 3-D integrated circuits, such as module layer assignment, module overlap, die-area matching, and minimizing through-die vias, it is imperative to make high-level synthesis for these systems layout-aware. To verify this claim, we compared the results of our algorithm with those produced by a traditional
Figure 3.9 Improvement in Wirelength over 3-D Layout-Unaware Binding

layout-unaware high-level synthesis flow that uses an efficient clique-partitioning based binding heuristic (that is layout-unaware) to create an RTL datapath, which is then passed on to a simulated annealing based 3-D floorplanner that is identical to the floorplanning engine used in our algorithm.

When compared to a layout-unaware flow, our algorithm was able to achieve significant improvements in wirelength distribution and in the number of through-die vias needed.

Figure 3.9(a) compares the percentage improvement in total wirelength obtained with our layout-aware binding algorithm. Figure 3.9(b) compares the improvement in the longest wirelength using our approach. The layout-aware binding technique results in an average total wirelength improvement of 29%, while the longest wirelength is improved by 21% on average. These improvements in wirelengths are due to better wirelength distributions resulting from a binding that takes the unique features of a 3-D layout into account during the binding process.

Figures 3.10 compares the wirelength distributions for the four benchmarks, obtained using both approaches, on 3-D floorplans consisting of three floorplan layers. From the figure, it is apparent that with layout-aware binding, a larger percentage of wires in the layout have a smaller
Figure 3.10  Wirelength Distributions for 3-D Layout-Aware and Layout-Unaware Bindings for HLS Benchmarks with Three Floorplan Layers

Figure 3.11  Improvement in Inter-Die Via Count over 3-D Layout-Unaware Binding
average wirelength when compared to layouts created by a traditional flow. A similar trend was also observed for floorplans with 2, 4, and 5 floorplan layers.

A layout-aware binding algorithm also leads to a significant reduction in the number of die-to-die vias introduced in a 3-D floorplan. Figure 3.11 illustrates the percentage reduction in the inter-die via count, when compared to a traditional layout-unaware binding scheme. An average of about 26% reduction was observed using our technique. This reduction is particularly significant when the number of floorplan layers in a 3-D integrated circuits increases.

3.6 Conclusions

In this chapter we addressed the problem of 3D-layout aware binding for three-dimensional integrated circuits, as part of a physical aware behavioral synthesis flow. We outlined a simulated annealing based formulation for the combined binding and floorplanning problem. Using our algorithm we obtained reductions of 29% in total interconnect lengths on average, compared to a traditional layout-unaware synthesis of 3-D layouts with 2 to 5 device layers. In addition, the worst-case net length in a given design also decreased by 21% on average, and a reduction in the number of inter-die vias by 27%, compared to a layout-unaware synthesis.
With shrinking feature sizes in deep sub-micron technologies, interconnect delays play a dominant role in the cycle time of digital circuits. It is essential to consider the impact of physical design during high-level synthesis. No prior work exists in literature that accounts for the topology of nets resulting from binding decisions during high-level synthesis. This chapter presents a novel unified incremental physical-level and high-level synthesis technique that uses accurate net topologies and distributed wire-delay models to guide resource allocation and binding decisions during design-space exploration. The proposed approach tightly integrates a floorplanner with a high-level synthesis binding algorithm. The location of data path modules in the floorplan is used to determine the minimal length RSMT of every net, to which the delay model is applied to accurately estimate delays of multi-terminal nets.

4.1 Introduction and Related Work

With deep sub-micron technologies, interconnect delays between datapath modules is increasingly becoming a major part of the cycle time [5]. Interconnect delays strongly depend on the number and sizes of the modules used in a datapath, and the relative locations of these modules in the floorplan. The area of the floorplan is determined by the number of modules present in the floorplan, and their positioning, which is partly determined by the binding decisions taken during high-level synthesis. Thus, it is important to consider binding information during floorplanning. Conversely, during binding, the effect of floorplanning should be
considered. The floorplan information is necessary to accurately predict the structure of the interconnects in a data path.

Taking interconnect costs into account during high-level synthesis has attracted significant attention. In some of the early work on HLS [30, 31, 32, 33, 34, 35], a simple estimate of the interconnect cost was determined by counting the number of wires and multiplexers required by a design. These estimates did not use any physical-level information and were used mainly to compare the wiring complexity of different datapath design alternatives. However, when interconnect delays began to be comparable to (and even exceed) gate delays, these simple estimates were no longer adequate, since it is not possible to accurately predict the performance of a design without first knowing enough about its floorplan and the structure of its interconnect.

In response to this, a number of researchers have considered the impact of physical details (such as floorplanning information) on high-level synthesis [15, 16, 36, 55, 56, 57, 58]. Others have used floorplan-level information for interconnect-driven HLS [17, 37, 38, 42]. Most of these approaches use a loosely-coupled floorplanner, where the floorplanning and binding decisions are made independent of each other. However, Gu et al. [59] have shown that tightly coupling high-level and physical synthesis and simultaneously performing HLS and floorplanning, improves their combined performance. In this chapter, we propose a unified incremental physical-level and high-level synthesis technique that uses accurate net topologies and distributed wire-delay models to guide resource allocation and binding decisions in high-level synthesis.

The rest of this chapter is organized as follows. In Section 4.2, we describe the nature of the problem addressed in this paper. In Section 4.3, we introduce the timing model used to estimate net delays. In Section 4.4, we describe our wire-topology aware binding algorithm. Section 4.5 presents the experimental results, and in Section 4.6 we concludes the chapter.
4.2 Estimating Wire Delays during High-Level Synthesis

The primary motivation for using floorplan-level information during high-level synthesis is to more accurately predict the interconnect structure of a data path. Wire length estimates from derived floorplans are used to drive the HLS synthesis. The underlying premise is that minimizing wire length often leads to corresponding reductions in interconnect delays. To simplify layout-driven high-level synthesis algorithms, most of the previous work have used simple interconnect models to estimate interconnect delays such as linear wire-delay models applied to two-terminal nets [15, 16, 35, 36, 37, 57, 58].

Resource sharing is a commonly used technique to reduce hardware requirements in high-level synthesis resulting in multi-terminal nets. A common simplification made in previous work is to break multi-terminal nets into two-terminal nets (point-to-point nets). This simplification has three major drawbacks:

- it ignores the net topology of a multi-terminal net during delay estimation,
- it assumes that all source-to-sink delays on a multi-terminal net are separable and independent, and
- it ignores the downstream capacitances during the source-to-sink delay computation.

Another common simplification in most of previous work is to use simple linear delay models for the wires using lumped $R$ and $C$ values for the wire parasitics. This simplification results from the fact that interconnects are treated as simple two-terminal point-to-point nets.

4.2.1 An Illustrative Example

Treating multi-terminal nets as equivalent to a set of two-terminal nets often leads to an under-estimation of the net delays, especially, when wire parasitics can no longer be ignored in deep submicron technology. This can be illustrated with a motivating example (Figure 4.1) that
Figure 4.1 A Motivating Example
Table 4.1  Comparison of Interconnect Delay Estimates with and without Accounting for Net-Topology

<table>
<thead>
<tr>
<th>Sink Module</th>
<th>Arrival time Point-to-point delay model</th>
<th>Arrival time Net topology-based delay model</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>543 ps</td>
<td>784 ps</td>
</tr>
<tr>
<td>R2</td>
<td>568 ps</td>
<td>787 ps</td>
</tr>
<tr>
<td>R3</td>
<td>534 ps</td>
<td>779 ps</td>
</tr>
</tbody>
</table>

shows the importance of accounting for the wire topology of a multi-terminal net during delay estimation. The dataflow graph in Figure 4.1(a) is mapped to an RTL datapath with one multiplier and three registers, with the corresponding floorplan shown in Figure 4.1(b).

Table 4.1 shows the estimated net delays for the data path in Figure 4.1(b) using ITRS technology parameters from [3] for the 70nm technology. The table compares delays of datapath nets modeled as traditional two-point nets, with delays obtained by taking the topologies of the nets into account. The Elmore delay model with distributed wire parasitics is used to compute the net delays for both the techniques. Please note that most of the earlier work use a simpler linear delay model for the wires using lumped $R$ and $C$ values for the wire parasitics. The table clearly demonstrates that

- to correctly estimate the signal arrival times at all the sink nodes of a multi-terminal net requires an estimation of the topology of the net, and the use of this information in an accurate delay estimation model,
- one cannot treat the different sink paths on a multi-terminal net as being “separable” and independent, and
- ignoring the presence of other downstream loads on a net while computing the arrival time on a sink pin can result in significant errors.
Buffers can be used on different sink paths on a net, thus allowing them to be treated as separable. However, buffers consume silicon real estate and dissipate power. A design flow that accounts for the wire parasitics of an entire multi-terminal net when computing the signal arrival times, could potentially reduce the need for extensively buffering signal nets to achieve timing closure.

4.3 Timing Model

To estimate net delays we first determine the topology of each net from its minimal wirelength Rectilinear Steiner Minimal Tree (RSMT), and then compute the wire $RC$ delays using a distributed $\pi$-model [11] for the wire segments in a net. The main advantage of this model is that it allows us to accurately estimate the delay between the source pin and each sink pin of a net. This is important since sink pin delays among pins on a net can differ significantly, especially for long nets, or for nets with large fan outs. The computation of individual delays results in much greater fidelity between estimated net delays and actual wire delays obtained after detailed routing, improving success in timing-closure.

The topology of each net is determined from its Rectilinear Steiner Minimal Tree, computed from the coordinates of the pins on a net (obtained from the floorplan). From the net topology, the lengths of all wire segments in every source-to-sink-pin signal path can be determined. These are then used to compute the equivalent $\pi$-model $RC$ circuit of the wire segments connecting each of the sink pins to the source pin. The module delay is modeled as the sum of the intrinsic module delay, and a load dependent delay that is linearly proportional to the external load capacitance. The RSMT net model leads us to an $RC$-tree, to which we apply the Elmore delay model, to compute wire delays. The advantage of using Elmore delays is that it allows us to compute signal delays in linear time.
The Elmore delay from a source pin \( p_0 \) to a sink pin \( p_i \) in an RC-tree is given by

\[
t_{p_0, p_i} = r_0 \cdot C_0 + \sum_{e_i \in \text{path}(p_0, p_i)} r_j \cdot \left( \frac{C_j}{2} + C_j \right)
\]

where, \( p_0 \) is the source pin of the RC-tree,
\( e_i \) is the edge from node \( n_i \) to its parent,
\( r_i \) is the resistance of edge \( e_i \),
\( c_i \) is the capacitance of edge \( e_i \),
\( C_i \) is the total capacitance of a tree rooted at node \( n_i \),
\( c_i \) is the driver resistance at source.

Assuming uniform wire width, the resistance and capacitance of an edge (i.e., the \( r_i \) and \( c_i \) of an edge \( e_i \)) is proportional to its length. Approximating each of the wire segments in an RC-tree with its equivalent \( \pi \)-model, the Elmore delay equation implies that the delay from source pin to a sink pin is proportional to the square of the length of the wire segments between the pins. This quadratic dependency suggests that in order to minimize the Elmore delay for a sink pin, the length of the path between the source and the sink pins should be minimized. In the Elmore delay formulation, the load capacitances of the pins are multiplied by the resistance of the wire segments on the path between source and sink pins. Therefore, pins with larger capacitance should be closer to the source. Further, to minimize delay to any sink pin, the total tree capacitance \( C_0 \) seen at the source pin should be minimized.

During high-level synthesis, decisions made during binding determine the number of sink pins driven by a source pin. For example, a register shared by a large number of variables in the scheduled DFG may require the register to drive a large number of data path modules that consume this variable. Binding also determines the number and types of multiplexers needed in a data path. The lengths of the wire segments to different sink pins in multi-terminal nets are
determined by the data path floorplan. To be effective, a timing-driven binding algorithm for high-level synthesis must consider the impact of binding, floorplanning, net fan outs, and net topologies, on the estimated net delays.

4.4 Net-Topology Aware Binding for High-Level Synthesis

A traditional floorplan-driven high-level synthesis technique, as proposed in previous approaches, is shown in Figure 4.2. In these approaches, simple wire delay models (such as wire delays with lumped $R$ and $C$ values applied to 2-terminal nets) derived from floorplans are used to guide HLS decisions. However, as shown in the illustrative example in Section 4.2.1, these models often under-estimate net delays, thus providing either incorrect or overly constrained time-bounds to the physical synthesis step that follows HLS. This often leads to either over-design, or long convergence times with multiple design re-spins, or in the worst case, a failure to achieve timing closure.

![Figure 4.2 Traditional Physically-Driven High-Level Synthesis](image)

Figure 4.2 Traditional Physically-Driven High-Level Synthesis
To provide better bounds for net delays during physical synthesis, it is necessary to determine the topology of multi-terminal nets present in a design, to more accurately estimate net delays that correlate better with actual delays present in the final layout. To the best of our knowledge, none of the previous work found in the literature has addressed this during HLS.

This work proposes a net-topology aware binding algorithm for HLS. The algorithm used in this work is illustrated in Figure 4.3. A Simulated Annealing (SA) based iterative improvement is used to tightly couple the HLS binding and floorplanning phases of synthesis. This enables a simultaneous search of the design spaces of module bindings, register bindings, floorplans, and net topologies, for solutions with smaller net delays. A unique feature of our algorithm is the use of two interleaved chains of moves that alternate between a sequence of bindings and floorplan moves. This allows the algorithm to perform independent neighborhood search of the binding and floorplan search spaces.

The binding algorithm accepts a scheduled data flow graph and a resource allocation for the data path. In our experiments, a force-directed list scheduler [30] is used to obtain schedules for the input dataflow graphs. A compatibility graph for each resource type is maintained to ensure that the algorithm always generates a legal resource binding. The algorithm returns the best RTL binding and floorplan found by the simulated annealing algorithm.

### 4.4.1 Solution Representation

The solution encoding used by the simulated annealing algorithm consists of two parts – (i) a set of DFG node-lists (or variable-lists) specifying the tasks (or variables) bound to each datapath resource (datapath unit or register), and (ii) a sequence pair [118] representing the relative location of datapath resources on a chip floorplan. SA moves that explore the HLS search space of resource bindings perturb the node-list and variable-list portions of a solution encoding. The search space of chip floorplans is explored via SA moves defined on the sequence pair (S1, S2).
4.4.2 Simulated Annealing Moves

Simulated Annealing neighborhood moves are defined in both, the HLS search space (module and register bindings), and the layout search space (chip floorplans). Five SA moves are defined on the sequence pair representing a solution's floorplan:

- Rotate datapath module,
- Shift datapath module in S1 string,
- Shift datapath module in S2 string,
- Swap two datapath modules in S1 string,
- Swap two datapath modules in S1 string.

Five SA moves are used to explore the HLS binding search space:

- Reassign binding of a DFG operation,
- Reassign binding of a DFG variable,
- Swap compatible DFG operation bindings,
- Swap compatible DFG variable bindings,
- Swap inputs of a commutative DFG operation.

The binding moves are guided by the resource compatibility graph maintained by the SA. During these binding moves, if the number of sources at the input of a data path resource (module or register) changes as a result of a change in the binding, multiplexers can vanish, appear, or change size. Changes in binding can significantly affect the netlist topology in a datapath, and thus the resulting floorplan and wire length statistics. By tightly integrating the HLS step of resource binding with floorplanning, any changes in the net topology are immediately reflected in the actual floorplan. Due to the concurrent search of the floorplan and HLS search spaces, the SA need not generate a new floorplan from scratch after each move, making our SA very efficient.
Figure 4.3 Proposed Net-Topology Aware Synthesis Algorithm
4.4.3 Simulated Annealing Cost Function

In this work, we treat the layout-driven HLS as an optimization problem with the aim of minimizing wire delays. Figure 4.4 illustrates the steps performed during the SA cost function computation. In addition to the traditional objectives of chip area and wire length, the cost function also minimizes the net delays among RTL modules in the floorplan.

The proposed algorithm is generic enough to allow any technique to determine net topologies and estimate net delays. For example, any of the accurate RSMT estimators from the literature could be used. Similarly, any accurate delay estimation engine (such as higher-moment models [119], \textit{RLC}-models [120], statistical delay models [121]) could be used to determine net delays.

![Figure 4.4 Cost Function Computation](image-url)
The choice of the net topology and net delay estimators is a trade-off between computation efficiency and required accuracy.

In this work, FLUTE [122], a recently proposed RSMT estimation algorithm is used for determining net topologies. FLUTE determines the net topology (RSMT) that is optimal (in terms of wire length) for nets with net-degree up to 7, and to within 5% optimal for nets with higher net-degrees. Additionally, FLUTE is very efficient in terms of run-times when compared to other algorithms. The choice of FLUTE was mainly dictated by its runtime efficiency, since the net-topology estimation engine is used thousands of times within our SA iteration loop.

Once the topology is obtained for each net, the widely used Elmore-delay model based delay calculation engine is used to compute the signal arrival times for all the sink pins on a net. The Elmore-delay model was used primarily for its run-time computation efficiency, and its excellent fidelity with actual net delays [123]. The Elmore-delay model is widely used in VLSI physical synthesis literature as a delay estimator [123, 124].

The following cost function is used to evaluate solutions:

\[
\alpha D_{\text{Norm}} + \beta W_{\text{Norm}} + \gamma A_{\text{Norm}}
\]

where, \(D_{\text{Norm}}\) is the normalized value of the estimated clock period, \(W_{\text{Norm}}\) is the normalized value of total wirelength, and \(A_{\text{Norm}}\) is the normalized chip area. The terms are normalized with respect to the best clock period, total wirelength, and chip-area values seen during search, which are dynamically updated during search.

Using a representative HLS benchmark, we performed an extensive set of experiments to determine a suitable combination of values for the weights used in our cost function. The values for each of these weights were changed in increments of 10, and the combination of weights that yielded the best results over an average of five runs was used for the rest of our experiments. In our experiments, we set \(\alpha = 250, \beta = 70, \gamma = 50\).
4.5 Experimental Results and Discussion

To validate the effectiveness of the proposed approach, an SA-based net-topology aware HLS Binding algorithm was developed and tested on four standard data flow intensive DSP benchmarks drawn from the HLS literature, namely – 16-point FIR filter, 8-point IIR filter, Elliptic Wave (EWF) filter, and 1-point 8X8 DCT filter. Each of these benchmarks was specified as a scheduled data flow graph (DFG), capturing the behavioral description of the architecture to be synthesized. The experiments were performed on a Linux workstation running on a 1.86GHz Intel Core2 Duo CPU processor with 2GB of main memory.

In this set of experiments, the objective was to minimize the clock period of a time-step of a DFG schedule. Two algorithms were compared:

- **Method-1**: an interconnect-centric floorplan-aware HLS synthesis that back-annotates the wire lengths derived from an integrated floorplanner to guide the high-level synthesis decisions (this is similar to that proposed in previous work), and

- **Method-2**: our proposed net-topology driven high-level synthesis flow that uses net topologies to estimate delays computed with an accurate distributed wire-delay model.

Table 4.2 compares the maximum wire delay among all data transfers in a scheduled data flow graph, for 70nm technology, using both methods. From the table, it can be observed that the proposed technique leads to significant improvements in the maximum wire delay, with corresponding reductions in the clock period. An average improvement in wire delays of 38.6% was achieved over a traditional floorplan-driven technique, with a maximum reduction of up to 48.9%.

Tables 4.3 and 4.4 respectively compare the total wire length and chip area of the same benchmarks using both techniques. From the tables it can be seen that the chip area and wirelengths are comparable when using both techniques. These results demonstrate that the
Table 4.2  Comparison of Net Delays for 70nm Technology

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Max. Wire Delay Method-1 (traditional)</th>
<th>Max. Wire Delay Method-2 (proposed)</th>
<th>% improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR</td>
<td>335 ps</td>
<td>224 ps</td>
<td>34.1%</td>
</tr>
<tr>
<td>EWF</td>
<td>540 ps</td>
<td>322 ps</td>
<td>40.4%</td>
</tr>
<tr>
<td>FIR</td>
<td>504 ps</td>
<td>369 ps</td>
<td>31.1%</td>
</tr>
<tr>
<td>DCT</td>
<td>937 ps</td>
<td>569 ps</td>
<td>48.9%</td>
</tr>
</tbody>
</table>

Average improvement: 38.6%

Table 4.3  Comparison of Total Wirelength for 70nm Technology

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Total WL using Method-1 (traditional)</th>
<th>Total WL using Method-2 (proposed)</th>
<th>% difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR</td>
<td>24892</td>
<td>23680</td>
<td>-4.9%</td>
</tr>
<tr>
<td>EWF</td>
<td>36471</td>
<td>32305</td>
<td>-11.4%</td>
</tr>
<tr>
<td>FIR</td>
<td>64213</td>
<td>64800</td>
<td>+9.1%</td>
</tr>
<tr>
<td>DCT</td>
<td>115982</td>
<td>122002</td>
<td>+5.1%</td>
</tr>
</tbody>
</table>

Average improvement: -0.53%

Table 4.4  Comparison of Chip Area for 70nm Technology

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Area using Method-1 (traditional)</th>
<th>Area using Method-2 (proposed)</th>
<th>% difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR</td>
<td>106624</td>
<td>107030</td>
<td>+0.4%</td>
</tr>
<tr>
<td>EWF</td>
<td>71808</td>
<td>74400</td>
<td>+3.5%</td>
</tr>
<tr>
<td>FIR</td>
<td>180840</td>
<td>189680</td>
<td>+4.7%</td>
</tr>
<tr>
<td>DCT</td>
<td>170404</td>
<td>180560</td>
<td>+5.6%</td>
</tr>
</tbody>
</table>

Average improvement: +3.6%
improvements in wire delays were accomplished by our technique with little overhead in terms of increased chip area or wire length.

Table 4.5 lists the execution times for the proposed net-topology driven high-level synthesis methodology. The table compares the execution times for benchmarks of different sizes. The size of each benchmark is measured in terms of the number of operations and variables in a scheduled data flow graph. In column 2 of the table, $|V|$ represents the number of nodes, and $|E|$ represents the number of edges, of a benchmark data flow graph. In addition to the benchmark sizes, the table also lists the number of RT-level modules needed to implement the data path for each benchmark. The RT-level datapath modules include datapath computational units as well as multiplexers used to steer data between these units. The number of data path modules provides a measure of the size of the floorplanning problem instance addressed by the algorithm. The execution times in column 4 is listed in terms of minutes and seconds.

The advantages offered by using wire-topology to estimate net delays during HLS synthesis decisions is evident when we back-annotate the computed module-to-module delays on the floorplan, to the scheduled DFG. The module-to-module delays are used to determine the contribution of the wire delays during execution of each of the DFG operations on the scheduled
Figure 4.5  Comparison of Net Delay Distributions for HLS Benchmarks
dataflow graph. Figure 4.5 illustrates this for the IIR, EWF, DCT, and FIR filter benchmarks. In this figure, the x-axis represents the wire delay between modules for all signal paths in a datapath, and the y-axis represents the number of nets that correspond to these delays. These wire delays are computed using the ITRS projected values for copper interconnect used in intermediate level nets for the 70nm technology.

In our experiments, we assume that DFG operations can be completed in a single clock period (i.e., single cycle operations). During each DFG operation, data transfers occur between several modules (input registers, multiplexers, data path units, and output registers). These wire delays are strongly dependent on the HLS binding and floorplan decisions made during HLS synthesis, and the resulting topologies of the nets created from these decisions.

These experiments demonstrate that incorporating accurate wire delay computations that take the wire topologies into account lead to significant improvements in wire delays and provide more realistic delay constraints to the subsequent physical synthesis steps. The delay distribution plots indicate that using the total wirelength metric and point-to-point net delays to guide HLS binding decisions does not necessarily lead to solutions with optimal clock cycle times. Use of the topologies of the nets, with accurate delay estimates often provide better feedback to the HLS steps during layout-driven synthesis.

The smaller delay values obtained with the proposed net topology-aware synthesis could be explained by examining the way the HLS binding and the floorplanning steps use this information during synthesis. Using information on module locations, the proposed algorithm is able to determine the impact of a HLS binding decision on the resulting topology of the net driven by a datapath module, and the source-to-sink delays for all the pins on the net. This way, the algorithm is better able to recognize binding decisions that lead to nets with large fan outs, or nets with large downstream loads, which a traditional approach may miss altogether, since they only consider two-terminal point-to-point nets. From our experiments, it is evident that
estimating net delays from net topologies results in smaller and more predictable net delays when compared with previous layout-aware high-level synthesis approaches that treat the source-to-sink delays in a multi-terminal net as separable and independent.

4.6 Conclusions

With increasing wire parasitics in deep sub-micron technologies, it is important to use more accurate delay estimation models during high-level synthesis. The signal paths between the source and multiple sink pins in a multi-terminal net are not separable and independent. This must be taken into account when estimating net delays during high-level synthesis. We have presented a new algorithm that computes the wire topology of multi-terminal nets from floorplans during high-level synthesis, and uses an accurate distributed wire delay model to estimate interconnect delays during resource binding. Experimental results show that the use of accurate wire delay estimates during binding can significantly reduce the impact of interconnects in resulting designs. We used this to minimize contribution of wire delays in the clock period of datapaths synthesized during high-level synthesis, with reductions in wire delays of 38.6% on average, when compared to traditional synthesis that only minimizes wire lengths. These improvements were achieved with minimal overhead in terms of chip area or total wirelength.
CHAPTER 5
TEMPERATURE-AWARE UNIFIED PHYSICAL-LEVEL AND HIGH-LEVEL SYNTHESIS

With rising power densities in modern VLSI circuits, thermal effects are becoming important in the design of integrated circuits. Elevated chip temperatures have an adverse impact on performance, reliability, power consumption, and cooling costs. To ensure adequate thermal management, all phases of the design flow must account for thermal effects on their design decisions. This chapter presents a unified physical-level and high-level synthesis technique that combines power minimization with temperature-aware scheduling, binding, and floorplanning. Section 5.1 introduces the problem and discusses the growing importance thermal issues in high-performance VLSI circuits in deep submicron technologies. Section 5.2 outlines the motivations for this work and the main contributions. Section 5.3 describes TABS, our unified physical-level and high-level behavioral synthesis approach. Section 5.4 presents our experimental results and Section 5.5 concludes the chapter.

5.1 Introduction

Relentless scaling of CMOS process technologies over the past three decades have enabled feature sizes to shrink continuously, allowing the integration of millions of transistors in modern VLSI chips. The primary drivers for transistor scaling are the associated benefits of lower system costs, improved performance, and system reliability. With decreasing feature sizes, increasing transistor counts (Figure 5.1(a)), and operating frequencies, power density in VLSI circuits has
increased dramatically (Figure 5.1(b)). According to ITRS, this trend is expected to increase in the future [3]. The heat generated in an integrated circuit is proportional to its power density. With increasing transistor counts and more aggressive frequency scaling in each technology generation, thermal management has been projected as one of the most challenging issues in the design of future high-performance integrated circuits [3, 60, 61]. Power-aware design alone fails to adequately address thermal issues, thus creating a need for temperature-aware design at all levels, including behavioral synthesis. Techniques such as behavioral or high-level synthesis (HLS) must take into account the implications of their design decisions on the thermal distribution in an integrated circuit.

Complex high performance Systems-on-Chip (SoC) designs typically contain different functional blocks with different activity rates (for example, logic vs. memory), and often use
aggressive low-power design methods such as clock/power gating techniques, thus creating non-uniform temperature distributions across a chip. Thermal management is particularly challenging in such designs, especially if they also contain analog circuit blocks, which are particularly susceptible to signal and current mismatches created by thermal gradients, adversely affecting their performance [62].

Though temperature-aware design makes use of power-management techniques, it is significantly different from traditional power-aware design. Different functional units in a circuit can have different activity rates and power dissipations, causing a non-uniform thermal distribution across a chip. This leads to “thermal hot spots” caused by localized heating in a chip, and spatial thermal gradients on a chip. Due to increasing transistor densities with CMOS scaling, spatial thermal variation and local thermal hot-spots are becoming increasingly common. This means that thermal-management techniques, must directly target the spatial behavior of operating temperature. Since many low-power techniques do not specifically address power density in local hot spots, they are not effective in mitigating thermal hot spots [63]. Temperature-aware design is therefore a critical part of modern chip design.

In the past, thermal issues were primarily addressed through packaging and cooling solutions, and were not an important concern during circuit design. To guarantee reliable runtime operation, these package-level cooling solutions were based on worst-case chip power consumption. Chip packaging and cooling costs account for 30% of total system costs in current designs, and the cost of such solutions increases super-linearly with power consumption [6], as illustrated in Figure 5.2. Packages designed for worst-case heating conditions are often an over-design, since a circuit rarely encounters the worst-case conditions during its lifetime, with typical applications dissipating 20% or less power compared to the worst-case [62]. With increases to chip power densities, costs for package-based worst-case cooling solutions can become prohibitively expensive. Package-level solutions such as heat sinks and fans, are not
particularly suited for mitigating localized thermal hot-spots whose temperatures can be significantly higher than chip-wide temperatures [65].

Due to increasing power densities and the need to address localized thermal hot-spots, the traditional approach to thermal management (fans and package-level solution) now faces fundamental difficulties. There is a critical need to simultaneously address thermal issues at each stage of design: micro-architecture design, circuit design, fabrication, and packaging and cooling design. The work described in this chapter addresses thermal issues during micro-architecture level design-space exploration.

5.2 Motivations for this Work

As described in Chapter 2, elevated on-chip temperatures can have several adverse effects on an integrated circuit. Higher temperatures adversely impact performance due to decreased transistor switching speeds and increased wire resistance, which can lead to timing violations. Sub-threshold leakage currents increase exponentially with temperature, leading to an exponential increase in static power dissipation, which could potentially lead to thermal runways. In addition, chip packaging costs increase with higher thermal budgets.
Currently, thermal issues in integrated circuits are typically addressed through packaging and cooling solutions such as use of heat-sinks, fans, etc. Package-level cooling solutions are based on worst-case chip power dissipation and estimated operating chip temperatures under these conditions.

The operating temperature of an integrated circuit can be calculated from the following linear equation:

\[ T_{chip} = T_a + R_{th} \frac{P_{tot}}{A} \]

where \( T_{chip} \) is the average chip temperature, \( T_a \) is the ambient temperature, \( P_{tot} \) is the total power consumption, \( A \) is the chip area, and \( R_{th} \) is the equivalent thermal resistance of the substrate silicon layer plus the package and heat sink. Typically, the worst-case estimated power consumption is used to estimate the average chip temperature, \( T_{chip} \), where the thermal distribution across the integrated circuit is assumed to be uniform.

While traditional cooling solutions designed using estimates of average chip-wide temperatures were adequate in the past, with aggressive scaling, increasing transistor counts, and operating frequencies, power densities in VLSI circuits have increased dramatically. A complex high-performance SoC design can typically contain different functional blocks with different switching activity rates, leading to uneven power dissipation among the various logic blocks on the chip. Due to low thermal conductivity of silicon, the rate of lateral heat propagation in a chip is slow, which can cause localized heating to occur much faster than chip-wide heating [63]. This can result in an uneven temperature distribution on the chip, creating on-chip thermal gradients. In addition, high-activity modules can create thermal hot-spots whose temperatures could be significantly higher than the chip-wide temperature. With wide-spread adoption of low-power design techniques such as clock and power gating, the occurrence of such on-chip thermal
gradients tends to become more common [66]. Typical package-level temperature management solutions such as heat-sinks, fans etc., are designed to address average chip-wide temperatures, and are not suited for mitigating power-densities in localized thermal hot-spots.

Low power techniques that minimize overall power dissipation [45] may not directly alleviate localized thermal-hotspots. Our experiments indicate that minimizing total switching power does not necessarily minimize peak chip temperature, mainly because minimizing total power dissipation does not prevent the occurrence of localized areas of high power densities, creating thermal-hotspots. Figure 5.3 illustrates this, by plotting the peak chip temperature of twenty-five different datapath designs for the COSINE2 benchmark, synthesized using a low-power design methodology that minimizes total switching power. All datapath designs used identical number and types of functional units (such as ALUs, multipliers, registers, etc.), but different schedules, bindings, and floorplans. In Figure 5.3, each data point represents the temperature of the hottest module in a datapath. To determine the peak temperature of each design, we performed a detailed thermal analysis of the design after datapath synthesis, using the floorplan information and module power traces, to obtain a chip-level thermal distribution and the peak temperature.

Figure 5.3  Plot of Peak Chip Temperature of Twenty-Five Datapath Designs for the COSINE2 HLS Benchmark Synthesized using a Low-Power Design Methodology that Minimizes Average Power
From Figure 5.3 we can conclude that the synthesis steps of scheduling, binding, and floorplanning significantly affect the peak module temperature in a chip. In addition, the total switching power is weakly correlated with peak chip temperature, since minimizing power does not necessarily lower peak chip temperature. For this example, the peak temperatures among the different designs differed by as much as 18.4% even though the total power dissipation among the designs differed by only 3.4%.

Figure 5.4 shows the IC floorplans of two of the designs examined while generating data for Figure 5.3. In Figure 5.4, the rectangles are datapath functional units e.g., ALUs, multipliers, registers, or multiplexers, color coded to indicate each module's average temperature. Both designs in Figure 5.4 have the same power dissipation of 0.799 Watts. However, the on-chip thermal distribution and peak chip temperature of both designs are significantly different, due to the difference in their HLS schedules, bindings, and floorplans. The scheduling, binding, and floorplan used in Design 1 of Figure 5.4 results in an average temperature of 87.7° C, with a peak...
temperature of 88.7°C. However, the schedule, binding, and floorplan used in Design 2 results in a much lower average chip temperature, with a peak temperature of only 75.2°C. This example shows that even designs with the same power dissipation could have dramatically different thermal distributions and peak temperatures, depending on the HLS and floorplanning decisions made during synthesis.

Figure 5.5 illustrates the impact of the sub-tasks of high-level synthesis on power and temperature of resulting datapath designs. During high-level synthesis, decisions regarding assignment of tasks across functional units as well as relative scheduling on individual tasks are made. Decisions made during scheduling and binding determine the static and dynamic power of datapath units. Datapath units with higher switching activity have proportionately higher power,
and correspondingly higher temperatures. Resource allocation decisions made during high-level synthesis also affect the resulting power dissipation and temperature. Clearly, high-level synthesis has a significant impact on thermal stresses in the resulting datapath.

The floorplan assigned to a datapath also determines the conditions that could result in thermal hot-spots. Thermal distribution is spatial in nature, determined by the floorplan. The presence of other hot modules in the vicinity of a module with high power dissipation could exacerbate the thermal conditions locally around the module. For example, in Figure 5.6, the temperature of module A is determined not only by its own power dissipation, but also by the amount of lateral heat diffusion from its neighboring modules H, I, J, K, and L. Consequently, an effective thermal management technique must incorporate some form of floorplan information during high-level synthesis decisions.

In this chapter, we propose an integrated approach to power and thermal management during high-level synthesis. Our synthesis method combines power-aware scheduling, binding, and floorplanning, together with temperature-aware binding and floorplanning based on feedback from thermal simulation.
In our approach, during high-level synthesis, forward-looking estimates of the floorplan are used to guide the HLS decisions of scheduling, allocation, and binding. To perform a multi-objective optimization of delay, chip area, power dissipation, and peak temperature, we use a multi-stage simulated annealing (SA) algorithm [68], where different objectives are minimized in different stages of annealing. In particular, we optimize chip area, power, and delay in the first stage of annealing. The best solution from stage-I is then further optimized for peak temperature during the second stage of annealing. To estimate the peak temperatures of datapath modules and the on-chip thermal distribution, a thermal analysis of the floorplan is performed. Use of a multi-stage annealing approach significantly reduces run-times by avoiding expensive thermal analysis of floorplans in earlier stages of search. We use ISAC [69], an accurate architectural-level thermal modeling tool to estimate the thermal distribution of a floorplan, and the peak power dissipation of all the modules in the floorplan, during the synthesis of the datapath. ISAC takes the power dissipation estimates of the modules and their floorplan locations, to compute module temperatures. The use of thermal modeling during design space exploration allows weeding out potential datapath designs that have a high probability of encountering thermal problems.

5.3 Overview of the Proposed Approach

This section gives an overview of TABS, a unified incremental physical-level and high-level synthesis (HLS) algorithm that considers the impact of task scheduling, resource binding, and floorplanning, on module power and temperature distribution in a floorplan. Our synthesis system uses a multi-stage simulated annealing algorithm (SA) to concurrently perform the synthesis tasks of scheduling, resource binding, floorplanning, and thermal analysis, to yield solutions that have lower peak module temperatures than traditional high-level synthesis techniques that do not account for module temperatures during synthesis.
Figure 5.7 illustrates the main steps in our approach. First, the dataflow graph (DFG) is simulated with typical input traces to profile each operation and data transfer. These profiles are then used to create input switching power tables for each resource type in the target datapath circuit, through simulations of netlists extracted from their layouts. Task schedules, resource bindings, power models, an RTL design library, floorplanner, and a thermal model of the IC package are then used to evaluate the IC temperature profile, power, area, and performance of datapaths created by TABS.

The main algorithm comprises of an HLS synthesis system tightly integrated with an incremental floorplanner and a thermal analysis tool. The inputs to the synthesis algorithm are (i) a behavioral description of a design in the form of a dataflow graph, (ii) an RTL module library, and (iii) profile-based input switching tables for the RTL modules modeling their switching power.
for different task and variable bindings. A simulated annealing based search algorithm is then used to concurrently perform task scheduling, resource binding, floorplanning, and module power minimization. To keep run-times reasonable, we use the FastSA algorithm proposed by Chen and Chang [82]. Module temperature profiles are then determined through a thermal analysis of the solutions. Due to its accuracy and fast run-times, we use ISAC [69] to perform a static thermal analysis of solutions examined by TABS. ISAC takes a floorplan and module power traces as inputs, and accurately computes the module temperatures. The goal of our synthesis algorithm is to concurrently minimize delay, power, area, and peak module temperature.

5.3.1 Multi-Stage Simulated Annealing

The search space addressed by TABS is multi-objective (delay, area, power, and temperature), with some of these objectives conflicting in nature. For example, floorplan and thermal objectives are generally conflicting. Floorplanning tends to pull modules together to minimize floorplan area. On the other hand, temperature minimization objectives tend to separate modules apart. Similarly, task scheduling and thermal objectives are also conflicting. Tighter scheduling constraints can result in higher total and peak module power, making the thermal minimization objective more difficult.

To address the multi-objective nature of the problem, we use a multi-stage simulated annealing algorithm similar to that in [68], where different optimization constraints and objectives, neighborhood moves, and cooling rates are used in different stages of annealing. Note that our multi-stage SA differs significantly from that of [68]. The work proposed in [68] addresses the problem of wirelength minimization in floorplanning, while our work addresses the more complex problem of floorplan-aware high-level synthesis. The objectives optimized in each annealing stage are:
• **Stage-I**: *Layout-aware low-power high-level synthesis*, where the optimization objectives are total power (functional unit and interconnect power), latency (*i.e.*, schedule length), and chip area.

• **Stage-II**: *Temperature-aware layout-driven synthesis*, where the optimization objectives are peak module temperature, total power, and chip area, while using the best schedule length found during stage-I as a constraint.

Figure 5.8 illustrates the annealing schedule used in our algorithm. In first stage of annealing, we perform a floorplan-driven high-level synthesis, with the optimization objectives of latency, module and interconnect power, and chip area, given the resource constraints for the datapath. The second stage is a low-temperature annealing stage, that takes the best low-power solution returned by the first stage, and uses a set of thermally-aware SA moves to minimize peak module temperature and create an even thermal distribution across the chip. Since low module temperatures are directly correlated with lower module power values, the low-power solution returned by the first stage provides a good starting point for peak temperature minimization of the second annealing stage of the multi-stage annealing process.

There are two main reasons for adopting a multi-stage annealing approach in our work. First, our experiments indicate that the use of a single weighted-sum of all objectives during the search process often yields solutions far from the Pareto-optimal front. However, a multi-stage annealing algorithm is able to sample the multi-objective solution space more efficiently, consistently yielding nearly optimal solutions. Secondly, use of a multi-stage annealing approach significantly reduces run-times by avoiding expensive thermal analysis of solutions in earlier stages of search, when they are far from optimal. As shown in Figure 5.9, the sub-space of thermally optimal solutions (region B in Figure 5.9) lies within the sub-space of power-optimal solutions (region A), which itself forms a subset of the entire solution space. We use stage-I of our multi-step SA to sample region A of the search space, and stage-II to sample sub-space B.
Figure 5.8 Multi-Stage Annealing Schedule

Figure 5.9 Solution Sub-Spaces Sampled at Different Stages of Search by Multi-Stage Simulated Annealing
To implement the multi-stage SA, we use an FSM-based controller to transition the SA search through the two stages of annealing. The FSM controller is used to dynamically adapt the search moves and the cost function used by the SA, to the annealing stage. Figure 5.10 illustrates the interface between the FSM controller and the multi-stage SA. Section 5.3.3 provides details on the SA moves used in the two stages of annealing.

5.3.2 Solution Encoding

The solution encoding used by our multi-stage simulated annealing algorithm consists of two parts – (i) a set of DFG node-lists (or variable-lists) specifying the tasks (or variables) bound to each datapath resource (datapath unit or register), and (ii) a sequence pair (S1,S2) representing the relative location of datapath resources on a chip floorplan. SA neighborhood search moves that explore the HLS search space of task schedules and resource bindings perturb the node-list and variable-list portions of a solution encoding. A node-list associated with an RTL module represents node priorities for DFG tasks bound to the module. Node priorities are used by a modified list
scheduling algorithm to schedule tasks in the dataflow graph. The search space of chip floorplans is explored via SA moves defined on the sequence pair. Solution cost functions are evaluated by using algorithms described in Section 5.5.4, that decode these solution encodings.

5.3.3 Simulated Annealing Moves

Simulated Annealing neighborhood moves are defined in both, the HLS search space (task schedules, module and register bindings), and the layout search space (chip floorplans).

5.3.3.1 Floorplan Moves

Five SA moves are defined on the sequence pair representing a solution's floorplan:

- Rotate datapath module,
- Shift datapath module in S1 string,
- Shift datapath module in S2 string,
- Swap two datapath modules in S1 string,
- Swap two datapath modules in S1 string.

These moves are used in both stages of annealing in the multi-stage SA used in TABS.

5.3.3.2 HLS Moves Used in Stage-I of Simulated Annealing

Four SA moves are defined to operate on the node-lists associated respectively with computational units in the solution datapath. These moves are designed to explore different task schedules and bindings encoded by the node-lists. These moves are termed as schedule-length variant moves, because they could result in task schedules with differing schedule lengths. The main objective of these moves is to find a solution that minimizes the total switching power among datapath functional units and interconnects, while at the same time meeting the user-specified delay constraint on the schedule. The SA moves used are:
• Change a DFG node location in a module's node-list,
• Swap two DFG nodes in a module's node-list,
• Change a DFG node's module binding,
• Swap the module bindings of two DFG nodes.

The first two move operations change a task's priority in the node-list used by the list scheduler, resulting in different schedules of possibly different schedule-lengths. The remaining two moves randomly change a DFG node's or variable's resource binding, with goal of minimizing interconnect and multiplexer costs.

5.3.3.3 HLS Moves Used in Stage-II of Simulated Annealing

The schedule-length of the best solution found by the multi-stage SA is used as a constraint in stage-II of annealing. The HLS moves defined in stage-II are designed to explore different schedules and bindings, without changing the length of these schedules (schedule length invariant moves). Five neighborhood search moves are defined:

• Shift a task within its mobility range,
• Migrate a task to another compatible module,
• Swap module bindings of two compatible tasks,
• Migrate a variable to another compatible register,
• Swap register bindings of two compatible variables.

All these three moves are designed to preserve the schedule-length, while exploring a variety of schedules and bindings. The first of these moves changes the time-step when a task is executed, without changing its resource binding. The next two moves change the resource bindings of tasks, and also possibly the time-step when they are executed. The remaining two moves change the bindings of variables to registers.
The main objective of stage-II is to minimize peak module temperature. Starting with a power-minimal datapath found by stage-I, different schedules, resource bindings, and floorplans are explored, with the goal of keeping total switching power within bounds, and at the same time optimizing peak temperatures among datapath functional units.

In addition to these moves, a set of module temperature-aware floorplan moves were designed to create a more uniform on-chip temperature distribution and reduce peak module temperatures. These moves are described in the next section.

5.3.3.4 Temperature-Aware Floorplan Moves

To support thermal-aware, unified high-level and layout-level optimization, the incremental floorplanner used in TABS also incorporates floorplan-level thermal optimization moves. In datapaths where functional units with high power densities are physically close to each other, thermal hot-spots can occur. To mitigate this situation, a thermal-aware swap operation is defined on the sequence pair of a floorplan. Under this operation, the functional units are sorted in order of increasing temperature, and the positions of a randomly chosen high-temperature functional unit in either the S1 or S2 sequence pair is exchanged with that of a low-temperature functional unit. This SA move allows more even thermal distribution over the chip, and prevents high-temperature modules from clustering in one area, leading to thermal hot-spots. These moves are used during stage-II of annealing.

5.3.3.5 Simulated Annealing Cost Function

Figure 5.11 illustrates the main steps used to decode a solution encoding and evaluate the cost function. The node-list associated with each functional unit is used by a list scheduler to schedule tasks in the DFG. Once a feasible schedule is determined, the lifetimes of all edges in the DFG are known, and we use the classical left-edge algorithm to allocate and bind registers in the datapath.
The sequence-pair maintained by each solution encoding is used to pack the datapath units into a floorplan. The task schedule is used to determine the sequence of tasks executed by each functional unit, which together with the module switching capacitance table derived through DFG profiling, is used to compute module switching power. Please note that switching capacitance for each operation-pair and variable-pair is pre-computed and stored in the table, before the start of the SA. The resource binding, together with the switching capacitance lookup table avoids the need for running a power simulation for estimating switching power during each cost function evaluation. The total power dissipation in each datapath module, together with the chip floorplan, is then used to perform a static thermal analysis to module temperatures.

The node-list associated with each functional unit is used differently in stages I and II of the SA. In stage-I, the node-list is used by the list scheduler to determine task schedules, whereas in stage-II, the node-list merely serves the purpose of specifying the list of tasks bound to a given resource.
The cost functions used in stages I and II differ in their objective functions. In stage-I, we use the following cost function:

\[ \alpha \cdot D_{\text{Norm}} + \beta \cdot P_{\text{Norm}} + \gamma \cdot A_{\text{Norm}} \]

where, \( D_{\text{Norm}} \) is the normalized value of latency, \( P_{\text{Norm}} \) is the normalized value of total power, and \( A_{\text{Norm}} \) is the normalized chip area. The terms are normalized with respect to the best latency, power, and chip-area values seen during search, which are dynamically updated during search.

In stage-II, where module temperatures of a low-power datapath are minimized, we use the following objective function:

\[ \theta \cdot T_{\text{Norm}} + \lambda \cdot P_{\text{Norm}} + \phi \cdot A_{\text{Norm}} \]

Here, \( T_{\text{Norm}} \) refers to normalized peak module temperature in the datapath, while \( P_{\text{Norm}} \) and \( A_{\text{Norm}} \) have the same meaning as in stage I.

Using a representative HLS benchmark, we performed an extensive set of experiments to determine a suitable combination of values for the weights used in our cost function. The values for each of these weights were changed in increments of 10, and the combination of weights that yielded the best results over an average of five runs was used for the rest of our experiments. In our experiments, we set \( \alpha = 250, \beta = 100, \gamma = 250, \theta = 300, \lambda = 250, \) and \( \phi = 100. \)

### 5.3.3.6 Chip-Package Thermal Model

As mentioned at the beginning of section 2.5, we use thermal modeling within the optimization flow to provide direct guidance for thermal optimization. To enable static thermal analysis of the placed modules in the floorplan, we use ISAC, a fast and accurate temperature analysis tool [69]. ISAC has been validated against FEMLAB [83], an accurate but slow commercial finite-element based simulator, with less than 3.5% estimation error [69]. For our thermal analysis, we assume
that each chip is attached to a copper heat sink using forced air cooling. Heat dissipates from the silicon die, through the cooling package to the ambient environment, and through the package to the printed circuit board. We assume an ambient temperature of 45°C and a silicon thickness of 200μm, similar to the assumptions made in [80, 81].

5.4 Experimental Results

The proposed algorithm was tested on a Linux-based workstation using a 1.86GHz Intel CoreDuo processor with 2GB memory. The overall flow used in our experiments is shown in Figure 5.7. Our experiments were performed on a comprehensive set of benchmarks drawn from real-life applications in the MediaBench suite [87]. Each of these benchmarks was specified as a DFG [86] capturing the behavioral description of the architecture to be synthesized. The RTL resource set used in our experiments comprised of multipliers, ALUs, registers, and multiplexers. These resources were synthesized using Cadence BuildGates and Encounter tools, and mapped to a 180nm technology library, and the capacitance values of the functional modules were then extracted. The areas, delays, and profiled power dissipation values from actual layouts of these RTL resources served as the RTL module library data in our experiments. We then estimated the switching activity between pairs of DFG operations that can potentially be executed on the same resource consecutively, and these were used to create the switching activity tables used for computing the switching power of resources.

We tested our synthesis technique on a comprehensive set of twenty HLS benchmarks. Each of these benchmarks was specified as a dataflow graph capturing the behavioral description of the architecture to be synthesized. These benchmarks are drawn from two sources:

- popular high-level synthesis benchmarks used in previous literature,
- real-life examples generated from the MediaBench suite [86, 87]
Among the set of popular benchmarks, we selected seven examples widely used in HLS studies. These examples focus on frequently used numeric calculations performed by various DSP applications. They are as follows:

- **ARF**: an implementation of an *auto-regression filter*,
- **EWF**: an implementation of an *elliptic wave filter*,
- **FIR1 and FIR2**: two versions of a *finite impulse response filter*,
- **COSINE1**: an implementation for a *1-D eight-point fast discrete cosine transform filter*, assuming constant coefficients,
- **COSINE2**: an implementation for a *1-D eight-point fast discrete cosine transform filter*, where the coefficients are given as inputs,
- **HAL**: an iterative solution of a *second-order differential equation solver*.

The dataflow graphs for these examples range in size from 11 nodes to 82 nodes. Table 5.1 provides details of the first set of benchmarks used in our experiments. In Table 5.1, column 1 lists the benchmark name. Columns 2 and 3 specify the characteristics of the corresponding dataflow graph, where column 2 represents the number of nodes and column 3 the number of edges.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Number of DFG Nodes</th>
<th>Number of DFG Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>ARF</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>EWF</td>
<td>34</td>
<td>47</td>
</tr>
<tr>
<td>FIR 2</td>
<td>40</td>
<td>39</td>
</tr>
<tr>
<td>FIR 1</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>66</td>
<td>76</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>82</td>
<td>91</td>
</tr>
</tbody>
</table>
Eleven examples extracted from the MediaBench benchmark suite were used to test our algorithm. The MediaBench suite [86] contains a wide variety of complete applications drawn from image processing, communications, and DSP domains. The dataflow graphs used in experiments range in size from 51 nodes to 333 nodes, and were obtained from [87]. The dataflow graphs were derived from four MediaBench applications:

- JPEG: a lossy compression technique for digital images,
- MPEG2: a digital video compression standard used for high-quality video compression,
- EPIC: an efficient pyramid image coder, and is an image compression utility,
- MESA: a software 3-D graphics package.

Table 5.2 provides details of the dataflow graphs from the MediaBench benchmark set used in our experiments. In Table 5.2, the first column states the names of the various functions where the basic blocks (DFGs) originated, and the second column specifies the MediaBench application to

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Application Domain</th>
<th>Num. of DFG Nodes</th>
<th>Num. of DFG Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2v2_smooth_downsample</td>
<td>JPEG</td>
<td>51</td>
<td>52</td>
</tr>
<tr>
<td>feedback_points</td>
<td>MESA</td>
<td>53</td>
<td>50</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>EPIC</td>
<td>56</td>
<td>73</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>JPEG</td>
<td>106</td>
<td>88</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>MESA</td>
<td>108</td>
<td>104</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>MESA</td>
<td>109</td>
<td>116</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>MPEG 2</td>
<td>114</td>
<td>164</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>JPEG</td>
<td>122</td>
<td>162</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>JPEG</td>
<td>134</td>
<td>169</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>MESA</td>
<td>197</td>
<td>196</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>MESA</td>
<td>333</td>
<td>354</td>
</tr>
</tbody>
</table>
which these functions belong. The remaining two columns specify the the number of nodes and edges in the dataflow graph, respectively.

In our experiments, the objective was to minimize the peak temperature among the functional units in a datapath during high-level synthesis. The temperature-aware synthesis method was compared with three other temperature-unaware methods:

- **Method-A**: A traditional floorplan-aware but power unaware synthesis methodology that minimizes chip area,
- **Method-B**: A low-power floorplan-aware synthesis methodology that minimizes the total power consumption,
- **Method-C**: A low-power floorplan-aware synthesis methodology that minimizes peak module power.

Method-A is an SA-based layout-driven high-level synthesis that tightly integrates a floorplanner within the HLS synthesis loop. The SA cost function used in Method-A minimizes the schedule length and the traditional floorplanning objectives of chip area and total wirelength. Method-B augments the cost function used in Method-A with a power minimization objective of minimizing total power, while Method-C augments Method-A's cost function with a power minimization objective of minimizing the peak power consumption of the datapath functional units. Since chip temperatures are correlated with power, comparing TABS with low-power synthesis techniques allows us to highlight the advantages of a temperature-driven synthesis technique over a low-power design methodology. A thermal analysis is performed on the datapaths produced by these methods, and the peak module temperatures are compared with the datapaths created by TABS.

Method-A is used as a baseline synthesis technique to study the contribution of a low-power design strategy towards minimizing on-chip temperatures, and contrast it against a power-unaware design methodology. The intuition for using Method-B is that lowering the total power
dissipation in a circuit would hopefully lower overall on-chip power density and hence the on-chip temperatures. The intuition for Method-C is that constraining peak module power, could help mitigate the formation of on-chip thermal-hotspots, and hopefully create a more even thermal distribution.

5.4.1 On-Chip Thermal Profiles for Power-Aware Synthesis

Tables 5.3, 5.4, and 5.5 compare results from the power-unaware synthesis technique (Method-A) with the two low-power synthesis methods (B and C). All results in these tables are averaged over 10 independent SA runs, for each of the three methods compared.

Table 5.3 compares the peak temperatures of designs synthesized using the Method-A, B, and C. In the table, column 1 lists benchmark names, while columns 2, 3, and 4 show the peak chip temperatures. The results indicate that a low-power design methodology does indeed result in an overall reduction in peak chip temperatures, when compared to power-unaware synthesis. This trend is true for all the benchmarks tested in our experiments. In addition, among the two low-power methods tested, it is observed that Method-B, on average, leads to lower chip temperatures, when compared to Method-C. Our experiments indicate that minimizing peak module power does not result in minimizing peak module temperatures in a design. As discussed earlier, a module's temperature depends not only on its own power dissipation but also on those of other modules that are physically located in its vicinity on the floorplan. Unless a synthesis methodology accounts for the thermal distribution among modules on a floorplan, minimizing peak module power alone is inadequate in minimizing module temperatures.

Table 5.4 compares the average estimated power of the circuits synthesized using the three methods. As expected, the power savings from using Methods B and C are significant compared to Method-A. It is also noted that, for the benchmarks tested in our experiments, minimizing peak
Table 5.3  Comparison of Peak Chip Temperature using Power-Unaware (Method-A) and Power-Aware (Methods B and C) Techniques

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Method-A (°C)</th>
<th>Method-B (°C)</th>
<th>Method-C (°C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>81.4</td>
<td>77.3</td>
<td>79.3</td>
</tr>
<tr>
<td>ARF</td>
<td>95.9</td>
<td>88.3</td>
<td>90.5</td>
</tr>
<tr>
<td>EWF</td>
<td>104.8</td>
<td>79.4</td>
<td>86.3</td>
</tr>
<tr>
<td>FIR2</td>
<td>95.1</td>
<td>84.6</td>
<td>84.9</td>
</tr>
<tr>
<td>FIR1</td>
<td>101.5</td>
<td>85.1</td>
<td>94.6</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>103.8</td>
<td>91.3</td>
<td>91.8</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>95.7</td>
<td>84.6</td>
<td>88.3</td>
</tr>
<tr>
<td>h2v2_smooth_downsample</td>
<td>80.3</td>
<td>79.9</td>
<td>79.3</td>
</tr>
<tr>
<td>feedback_points</td>
<td>96.4</td>
<td>89.5</td>
<td>91.4</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>101.3</td>
<td>86.4</td>
<td>97.1</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>85.3</td>
<td>78.9</td>
<td>82.6</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>105.9</td>
<td>86.7</td>
<td>93.2</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>102.5</td>
<td>89.8</td>
<td>94.7</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>99.6</td>
<td>87.2</td>
<td>89.3</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>98.4</td>
<td>91.3</td>
<td>92.6</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>107.4</td>
<td>95.8</td>
<td>97.1</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>97.3</td>
<td>88.6</td>
<td>91.9</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>105.2</td>
<td>102.6</td>
<td>103.7</td>
</tr>
<tr>
<td>Block Name</td>
<td>Method-A (W)</td>
<td>Method-B (W)</td>
<td>Method-C (W)</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------</td>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td>HAL</td>
<td>0.583</td>
<td>0.524</td>
<td>0.577</td>
</tr>
<tr>
<td>ARF</td>
<td>1.234</td>
<td>1.043</td>
<td>1.097</td>
</tr>
<tr>
<td>EWF</td>
<td>1.238</td>
<td>0.802</td>
<td>0.897</td>
</tr>
<tr>
<td>FIR2</td>
<td>0.855</td>
<td>0.713</td>
<td>0.786</td>
</tr>
<tr>
<td>FIR1</td>
<td>1.181</td>
<td>0.830</td>
<td>0.892</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>1.479</td>
<td>1.127</td>
<td>1.311</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>1.803</td>
<td>1.397</td>
<td>1.612</td>
</tr>
<tr>
<td>h2v2_smooth_downsample</td>
<td>0.391</td>
<td>0.364</td>
<td>0.386</td>
</tr>
<tr>
<td>feedback_points</td>
<td>1.081</td>
<td>0.921</td>
<td>1.002</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>1.049</td>
<td>0.847</td>
<td>0.919</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>0.654</td>
<td>0.541</td>
<td>0.585</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>3.096</td>
<td>2.135</td>
<td>2.469</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>2.908</td>
<td>2.263</td>
<td>2.355</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>2.181</td>
<td>1.655</td>
<td>1.732</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>1.694</td>
<td>1.371</td>
<td>1.459</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>1.734</td>
<td>1.418</td>
<td>1.544</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>1.342</td>
<td>1.019</td>
<td>1.114</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>4.410</td>
<td>3.671</td>
<td>3.715</td>
</tr>
</tbody>
</table>
Table 5.5  Comparison of Peak Module Power using Power-Unaware (Method-A) and Power-Aware (Methods B and C) Techniques

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Method-A (mW)</th>
<th>Method-B (mW)</th>
<th>Method-C (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>255.8</td>
<td>227.0</td>
<td>224.0</td>
</tr>
<tr>
<td>ARF</td>
<td>279.4</td>
<td>250.4</td>
<td>243.0</td>
</tr>
<tr>
<td>EWF</td>
<td>365.0</td>
<td>202.0</td>
<td>173.0</td>
</tr>
<tr>
<td>FIR2</td>
<td>267.0</td>
<td>242.0</td>
<td>242.0</td>
</tr>
<tr>
<td>FIR1</td>
<td>315.4</td>
<td>255.4</td>
<td>243.0</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>285.2</td>
<td>238.6</td>
<td>224.1</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>277.8</td>
<td>212.8</td>
<td>203.8</td>
</tr>
<tr>
<td>h2v2_smooth_downsample</td>
<td>151.6</td>
<td>150.0</td>
<td>144.7</td>
</tr>
<tr>
<td>feedback_points</td>
<td>299.3</td>
<td>258.2</td>
<td>241.2</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>277.4</td>
<td>223.8</td>
<td>198.4</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>168.0</td>
<td>168.0</td>
<td>153.7</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>369.8</td>
<td>251.2</td>
<td>224.6</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>303.4</td>
<td>257.6</td>
<td>227.3</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>301.6</td>
<td>274.2</td>
<td>231.8</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>316.2</td>
<td>267.1</td>
<td>242.4</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>335.2</td>
<td>274.8</td>
<td>243.5</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>272.6</td>
<td>197.3</td>
<td>184.1</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>335.8</td>
<td>293.4</td>
<td>271.2</td>
</tr>
</tbody>
</table>
Figure 5.12  Floorplan-Level Thermal Distribution for the *interpolate-aux* MediaBench Benchmark Synthesized using the Power-Unaware Method-A. (The part of the floorplan highlighted by a circle shows the presence of a thermal-hotspot in the design)

module power results in a small but significant increase in the total power dissipation, when compared to Method-B. This partly explains the higher peak temperatures observed in circuits synthesized using Method-C, when compared to Method-B.

Table 5.5 compares the peak module power among circuits synthesized using Methods A, B, and C. Since Method-C explicitly minimizes the peak module power instead of total power, it is no surprise that circuits synthesized using Method-C tend to have the lowest peak module power among the three methods compared.

A floorplan-level thermal distribution of the synthesized designs provides a better picture of the impact of power minimization techniques on on-chip temperatures. Figure 5.12 illustrates the chip-level temperature distribution for the MediaBench benchmark *interpolate-aux*, synthesized using Method-A. Figure 5.12 shows that for this design there exists a thermal-hotspot shown by a
Figure 5.13  Floorplan-Level Thermal Distributions for the *interpolate-aux* MediaBench Benchmark Synthesized using the Low-Power Methods B and C. (The portion of Method-C’s floorplan highlighted by a circle shows a localized region of high power density and high temperature)

dotted circle in the lower left corner of the floorplan. Though Method-A is able to find a solution that with a small floorplan, the binding and floorplanning results in significant on-chip thermal gradients, and a thermal-hotspot that could cause reliability concerns, and could even result in timing violations, as explained in Chapter 2.

Figure 5.13 illustrates the floorplan-level thermal distribution plots for the *interpolate-aux* benchmark using the power-aware methods B and C. From the figure, it is clear that a power-aware synthesis methodology helps lower on-chip temperature when compared to a power-
unaware method. It is seen that module temperatures of the low-power solutions in Figure 5.13 are significantly lower than those in Figure 5.12. However, for this example, the thermal gradients on the floorplan created with Method-C are significantly larger than those created using Method-B. This is because, Method-B tries to minimize the total chip power, which appears to create a more uniform temperature distribution across the floorplan, when compared to Method-C. Method-C minimizes peak module power at the expense of overall power, which may result in localized areas of high power-density, as highlighted by the red dotted circle on the floorplan created by Method-C.

The results very similar for the other benchmarks that were tested in our experiments. From these results we can conclude that reducing power dissipation can significantly reduce average chip temperatures and the occurrence of thermal-hotspot formation. However, a low-power design strategy alone cannot eliminate the possibility of localized regions of high power densities and elevated temperatures. As illustrated in Figure 5.3 in Section 5.2, a low-power synthesis technique can result in designs with high peak chip temperatures. Typical low-power design methodologies that use either the total power or peak module power as the optimization criteria fail to capture the spatial power density differences that exist across a chip floorplan, that potentially lead to non-uniform thermal distributions and localized regions of high power density, as seen in the Method-C example of Figure 5.13.

5.4.2 Temperature Reduction through Temperature-Aware HLS

As explained in Section 5.2, the thermal distribution and peak chip temperature depends on three factors:

- the HLS subtasks of scheduling, allocation, and binding.
- the chip-level floorplan.
- the power density distribution among the functional units in the datapath.
To successfully address thermal management at design time, a high-level synthesis technique must tightly integrate the high-level tasks of scheduling, allocation, and binding, with the floorplanning stage. In addition, it must be able to directly control all three factors listed above, to alleviate regions of high power densities (and hence temperature), and ensure a uniform on-chip thermal distribution.

Our technique addresses thermal management during architectural synthesis by directly controlling all three of these factors. To minimize power density across the chip we use a two-stage power minimization strategy. In the first stage, which is similar to other low-power HLS methods, we minimize total power dissipation in the circuit. The best solution found by the first stage then forms the starting point for temperature-driven power density optimization in the second stage, where feedback from accurate floorplan-aware thermal analysis is used to guide HLS binding and floorplanning decisions in an attempt to evenly distribute power over the chip and mitigate the occurrence of localized regions of high power density.

We also use thermally-aware HLS re-scheduling and re-binding moves during the second stage of the algorithm, to explore the architectural design space and search for solutions that have uniform chip-wide thermal distributions and low peak temperatures. These re-scheduling and re-binding moves are guided by feedback obtained from accurate floorplan-level thermal analysis, to find solutions with low peak temperatures.

In our work, we also search for floorplans that minimize chip temperatures. Thermal hot spots may occur because a number of functional units with high power density are physically close to each other. It is common for high-activity, high-power functional units to frequently communicate with each other. This causes the floor planner to position the functional units near each other in order to reduce wirelength. During the second stage of our algorithm, our technique uses feedback from thermal analysis to identify regions of the floorplan containing clusters of high temperature datapath units, and makes an attempt to separate high temperature modules,
without significantly increasing the chip area or wirelength. This is done through a combination of changing the positions of these modules, as well as an effective use of white-space in the floorplan to alleviate power density in high temperature regions.

Figure 5.14 compares the floorplan-level thermal distributions of the best solutions found by the low-power synthesis Method-B and our proposed temperature-aware synthesis method TABS, for the interpolate-aux benchmark. It can be observed that the floorplan of the best solution found by TABS is substantially different from that of Method-B. The on-chip thermal gradient of the solution from TABS is significantly smaller than that produced by Method-B.
Whereas, more than 50% of the chip area in Method-B's solution has a temperature higher than 85°C, the highest temperature on the floorplan of the solution found by TABS was only 77.6°C, with a significant portion of the chip with temperatures less than 70°C. Similar trends were observed for all other benchmarks tested.

TABS achieves these improvements by using a combination of the following: (i) selectively re-binding DFG operations from high-temperature and high power density functional units, to functional units with lower temperatures and power densities, (ii) modifying the floorplan by selectively changing the locations of high temperature functional units, so as to even out the thermal gradients on the chip and minimize the occurrence of thermal-hotspots, and (iii) selective use of white-space on the chip floorplan to mitigate localized regions of high-temperature.

Table 5.6 compares the peak chip temperatures using our temperature-aware technique with the low-power Methods B and C. The temperature values are averaged over ten independent simulated annealing runs using different random number seeds. For all the benchmarks tested, our technique found solutions with lower peak temperature values when compared to a low-power synthesis technique, highlighting the advantages of a temperature-aware technique over low-power methods. The average peak chip temperature for the entire benchmark set using our method was 76.1°C. Using Method-B, the average peak temperature was 87.1°C, while using Method-C resulted in an average peak temperature of 90.5°C. Table 5.7 shows the percentage improvements in peak temperatures using our method, when compared to the power-minimizing Methods B and C. Our temperature-aware synthesis methodology is able to achieve significant reductions in peak temperature over the power minimizing Methods B and C. Improvements over the total-power minimizing technique (Method-B) averaged 12.4%, with an improvement up to of 20.7%. The average improvement over Method-C was 15.74%, with a maximum figure of 22.5%.
Table 5.6  Comparison of Peak Chip Temperature using the Temperature-Aware Synthesis Methodology (TABS) and Power-Aware Techniques (Methods B and C)

<table>
<thead>
<tr>
<th>Block Name</th>
<th>TABS (°C)</th>
<th>Method-B (°C)</th>
<th>Method-C (°C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>69.7</td>
<td>77.3</td>
<td>79.3</td>
</tr>
<tr>
<td>ARF</td>
<td>78.2</td>
<td>88.3</td>
<td>90.5</td>
</tr>
<tr>
<td>EWF</td>
<td>74.3</td>
<td>79.4</td>
<td>86.3</td>
</tr>
<tr>
<td>FIR2</td>
<td>78.2</td>
<td>84.6</td>
<td>84.9</td>
</tr>
<tr>
<td>FIR1</td>
<td>75.3</td>
<td>85.1</td>
<td>94.6</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>75.9</td>
<td>91.3</td>
<td>91.8</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>75.7</td>
<td>84.6</td>
<td>88.3</td>
</tr>
<tr>
<td>h2v2_smooth_downsample</td>
<td>69.4</td>
<td>79.9</td>
<td>79.3</td>
</tr>
<tr>
<td>feedback_points</td>
<td>76.1</td>
<td>89.5</td>
<td>91.4</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>76.6</td>
<td>86.4</td>
<td>97.1</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>75.6</td>
<td>78.9</td>
<td>82.6</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>78.4</td>
<td>86.7</td>
<td>93.2</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>80.7</td>
<td>89.8</td>
<td>94.7</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>74.3</td>
<td>87.2</td>
<td>89.3</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>72.4</td>
<td>91.3</td>
<td>92.6</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>79.8</td>
<td>95.8</td>
<td>97.1</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>71.2</td>
<td>88.6</td>
<td>91.9</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>88.4</td>
<td>102.6</td>
<td>103.7</td>
</tr>
</tbody>
</table>

Average 76.1 87.1 90.5
Table 5.7  Percentage Improvements in Peak Chip Temperature using the Temperature-Aware Synthesis Methodology (TABS) over Power-Aware Techniques (Methods B and C)

<table>
<thead>
<tr>
<th>Block Name</th>
<th>% improvement over Method-B</th>
<th>% improvement over Method-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>9.83</td>
<td>12.1</td>
</tr>
<tr>
<td>ARF</td>
<td>11.44</td>
<td>13.59</td>
</tr>
<tr>
<td>EWF</td>
<td>6.42</td>
<td>13.91</td>
</tr>
<tr>
<td>FIR2</td>
<td>7.57</td>
<td>8.56</td>
</tr>
<tr>
<td>FIR1</td>
<td>11.52</td>
<td>20.4</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>16.87</td>
<td>17.32</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>10.52</td>
<td>14.27</td>
</tr>
<tr>
<td>h2v2_smooth_downsample</td>
<td>13.14</td>
<td>12.48</td>
</tr>
<tr>
<td>feedback_points</td>
<td>14.97</td>
<td>16.74</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>11.34</td>
<td>21.11</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>4.18</td>
<td>8.47</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>9.57</td>
<td>15.88</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>10.13</td>
<td>14.78</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>14.79</td>
<td>16.8</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>20.7</td>
<td>21.81</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>16.7</td>
<td>17.82</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>19.64</td>
<td>22.52</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>13.84</td>
<td>14.75</td>
</tr>
</tbody>
</table>

Average: + 12.40% + 15.74%
<table>
<thead>
<tr>
<th>Block Name</th>
<th>% Area Overhead over Method-B</th>
<th>% Area Overhead over Method-C</th>
</tr>
</thead>
<tbody>
<tr>
<td>HAL</td>
<td>9.08</td>
<td>10.28</td>
</tr>
<tr>
<td>ARF</td>
<td>11.24</td>
<td>9.92</td>
</tr>
<tr>
<td>EWF</td>
<td>6.35</td>
<td>4.82</td>
</tr>
<tr>
<td>FIR2</td>
<td>11.02</td>
<td>10.77</td>
</tr>
<tr>
<td>FIR1</td>
<td>8.57</td>
<td>7.71</td>
</tr>
<tr>
<td>COSINE 1</td>
<td>9.27</td>
<td>8.84</td>
</tr>
<tr>
<td>COSINE 2</td>
<td>10.68</td>
<td>7.21</td>
</tr>
<tr>
<td>h2v2_smooth_downsample</td>
<td>1.02</td>
<td>1.47</td>
</tr>
<tr>
<td>feedback_points</td>
<td>6.85</td>
<td>7.42</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>7.21</td>
<td>9.33</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>6.28</td>
<td>4.51</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>9.41</td>
<td>11.36</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>8.14</td>
<td>10.22</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>5.92</td>
<td>6.88</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>11.22</td>
<td>12.39</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>9.48</td>
<td>8.57</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>9.37</td>
<td>7.56</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>14.6</td>
<td>10.74</td>
</tr>
</tbody>
</table>

Average                  + 8.61%                        + 8.38%
These improvements in thermal distribution are achieved through re-binding and floorplanning moves performed by TABS. In an attempt to mitigate localized areas of high power density, some of these floorplanning moves involve introducing some white-space in the floorplan, leading to chip area overhead. Figure 5.15 compares the floorplan areas of the Mediabench benchmarks using the three approaches compared. Table 5.8 lists the area overhead of our technique when compared to solutions found by Methods A and B.
From Table 5.8 and Figure 5.15, we can see that the significant improvements in peak chip temperatures using our approach only has a modest area overhead that averages to less than 10% for the tested benchmarks, with a peak area overhead of 14.6% for the largest MediaBench benchmark *invert_matrix_general*.

### 5.4.3 Comparison with Related Work

This section compares our work with other temperature-aware behavioral synthesis works reported in the literature.

The temperature-aware binding algorithm of Mukherjee, Memik, and Memik [77] uses an analytical model that relates the switching activity of datapath functional units to their expected heat dissipation. Their analytical model does not utilize geometry or floorplan information to guide temperature estimation of functional units, making their approach inaccurate because it fails to capture lateral heat dissipation among the modules through the silicon substrate, which can lead to significant differences [104].

The temperature-aware binding technique proposed by Mukherjee, Memik, and Memik in [78] applies an iterative re-binding algorithm to a traditional low-power HLS solution, to create a datapath with uniform switching activity distribution among the functional units. Their temperature-aware re-binding algorithm redistributes tasks among the functional units with the goal of creating an even power dissipation among the resources. The main drawback of their approach is that they base all their re-binding decisions on functional unit switchings instead of module temperatures, which could lead to significant inaccuracies. In addition, their algorithm does not use any floorplan or module geometry information to guide their temperature minimization moves, which could lead to significantly inaccurate results.

The work proposed by Mukherjee and Memik in [80] integrates task scheduling, resource allocation and assignment, and post-floorplan thermal simulation into a thermal-aware high-level
synthesis framework. This work addresses some of the deficiencies of [77] and [78] by using floorplanning information during temperature minimization. This work starts with an initial low-power solution, which is then used to create a thermal-aware floorplan using HotFloorplan [102]. A series of re-scheduling and re-binding moves are then applied to this initial solution, in an attempt to redistribute the switching activity among the functional units and mitigate thermal-hotspots. After each move, on-chip temperature profiles are obtained using the HotSpot simulator [70]. Though the authors of [80] use accurate estimates of module temperatures, they restrict their search to the HLS sub-space of task schedules and resource bindings, without modifying the initial floorplan. Since on-chip thermal profiles can change significantly with changes in the floorplan, by not exploring different floorplans, the algorithm proposed in [80] could miss many opportunities for temperature minimization.

Gu et al. [81] propose a thermal-aware floorplanning high-level synthesis system that makes use of integrated high-level and physical-level thermal optimization techniques. They use multiple voltages and voltage partitioning of the tasks in order to reduce a design's power consumption and peak temperature. However, much of the peak temperature minimizations in their technique were obtained through the use of multiple voltages for power minimization.

In contrast to these works, our approach tightly integrates high-level and physical-level synthesis steps at all stages of synthesis, to provide accurate estimates of thermal distribution. We concurrently explore the sub-spaces of operation scheduling, resource binding, and floorplanning, in an attempt to minimize module temperatures during behavioral synthesis. During search, we also minimize peak and average switching power of modules, to mitigate on-chip thermal hotspots. Unlike the works in [77], [78] and [80], in addition to minimizing module switching power, we also explore the impact of changes to the floorplan, in an effort to minimize chip temperatures. This combined search in the sub-spaces of behavioral synthesis and physical
synthesis leads to significant improvements over other techniques proposed in the literature, that restrict their search to only behavioral synthesis.

To accurately determine on-chip thermal distribution, a detailed information of the chip floorplan, module power profiles, and chip packaging (size and thermal characteristics of the silicon chip, heat spreader, heat sink, etc.) is required. The module power profiles are used to determine on-chip heat sources, and the floorplan information is used to determine the geometrical location of these heat sources, and the lateral thermal conductivity between the datapath modules on chip.

Layout-driven temperature-aware behavioral synthesis requires a detailed knowledge of the following information, in order to accurately estimate the on-chip thermal profile of a synthesized design:

- number and type of datapath functional units allocated,
- operation schedule and resource binding information, to determine the switching power,
- layout-level floorplan information,
- characterized RTL library data (module areas, delay).

To enable a fair comparison, a detailed information of the RTL module library used, together with task schedules, the number and types of functional-units used, datapath resource bindings, the chip floorplan, and the functional-unit power profiles, is needed for each benchmark tested in the experiments. Since the RTL module library and power profile data used in [77, 78, 80, 81] are not publicly available, a direct comparison with their work is not possible. However, to put the quality of our results in perspective with those proposed in the literature, we list the peak temperature reductions obtained by TABS with those reported in other temperature-aware high-level synthesis methods, in Table 5.9. TABS enables higher temperature reductions due to the tight coupling between its incremental floorplanner and high-level synthesis engine, which enables more accurate thermal analysis during synthesis.
Table 5.9 Comparison with Related Work

<table>
<thead>
<tr>
<th>Work Compared</th>
<th>Average Peak Temperature Reduction (°C)</th>
<th>[77]</th>
<th>[78]</th>
<th>[80]</th>
<th>[81]</th>
<th>Ours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg T (°C)</td>
<td></td>
<td>3.6</td>
<td>12.21</td>
<td>7.34</td>
<td>12.5</td>
<td>14.4</td>
</tr>
</tbody>
</table>

The CPU run times of our technique compare favorably with those of other temperature-aware high-level synthesis methods reported in the literature. While the authors of [77, 78] do not report the run times of their algorithms, the authors of [80] state that run time for creating their initial thermally-aware floorplan (using HotFloorplan [102]) is in the order of hours. The authors of [81] report that for their largest benchmark consisting of 119 DFG nodes, the CPU run time was 1195 seconds.

Table 5.10 shows the CPU run times for the MediaBench benchmark examples synthesized using our algorithm. In the table, column 1 specifies the benchmarks, while column 2 specifies the sizes of these benchmarks in terms of the number of DFG nodes. Column 3 lists the wall-clock time in seconds, averaged over the ten independent runs for these benchmarks.

<table>
<thead>
<tr>
<th>Block Name</th>
<th>DFG Size</th>
<th>Run times (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>h2v2_smooth_downsample</td>
<td>51</td>
<td>816</td>
</tr>
<tr>
<td>feedback_points</td>
<td>53</td>
<td>730</td>
</tr>
<tr>
<td>collapse_pyr</td>
<td>56</td>
<td>853</td>
</tr>
<tr>
<td>write_bmp_header</td>
<td>106</td>
<td>1006</td>
</tr>
<tr>
<td>interpolate_aux</td>
<td>108</td>
<td>1210</td>
</tr>
<tr>
<td>matrix_multiply</td>
<td>109</td>
<td>1125</td>
</tr>
<tr>
<td>IDCT_col</td>
<td>114</td>
<td>1255</td>
</tr>
<tr>
<td>JPEG_IDCT_ifast</td>
<td>122</td>
<td>1410</td>
</tr>
<tr>
<td>JPEG_FDCT_islow</td>
<td>134</td>
<td>1510</td>
</tr>
<tr>
<td>smooth_color_z_triangle</td>
<td>197</td>
<td>1576</td>
</tr>
<tr>
<td>Invert_matrix_general</td>
<td>333</td>
<td>2705</td>
</tr>
</tbody>
</table>
Figure 5.16 illustrates the evolution of the peak module temperature with run time, using our algorithm, for the IDCT_col MediaBench benchmark, which is a large benchmark consisting of 114 DFG nodes. In the figure, Stage-I is the low-power floorplan-driven synthesis phase (stage-I of the two-stage SA run). The algorithm starts with a random initial solution in stage-I, when the initial floorplan is large, leading to a low initial peak temperature. As the algorithm progresses in stage-I, it improves the floorplan, which leads to an increase in peak temperature since the modules are now packed closer together. This is evidenced by the rise in the peak temperature during the first half of stage-I. However, as the algorithm optimizes module switching power
through re-scheduling and re-binding moves in stage-I, it leads to a lowering of the average switching power of the datapath, leading to a corresponding fall in the peak temperature. During the thermally-aware synthesis phase of stage-II, this trend in peak temperature reduction continues through the use of temperature aware re-scheduling, re-binding, and floorplanning moves used by our algorithm. It is interesting to note that for the example illustrated in Figure 5.16, a significant portion of the temperature improvement is obtained within the first 10 minutes of run time, with additional improvements obtained with more CPU time. A similar trend was also observed for other benchmarks.

Since our algorithm is an iterative improvement technique that always maintains a complete solution during its run, it can be stopped at any time, to obtain a valid solution. This is a significant advantage of using our technique, since we can obtain different solution quality-runtime trade offs depending on how long the algorithm is allowed to run.

5.5 Conclusions

In this chapter, we have presented TABS, an integrated temperature-aware high-level synthesis technique that tightly couples the HLS tasks of scheduling and binding, with physical-level estimates from an incremental floorplanner. A multi-stage synthesis approach is used that concurrently performs power and thermal optimizations during synthesis, with the goal of minimizing peak temperature of the functional units and obtain an even thermal distribution among the resources. We have demonstrated that our approach is indeed effective in preventing thermal-hotspot formation by minimizing peak temperatures. Our experimental results indicate that compared to temperature-unaware power-optimized high-level synthesis approaches, our synthesis technique reduces peak module temperatures by an average of 16%. These improvements in peak temperatures are achieved with an average of less than 9% increase in chip area over power-optimized designs.
CHAPTER 6

STOCHASTIC WIRELENGTH ESTIMATION-BASED HIGH-LEVEL DESIGN
SPACE EXPLORATION

This chapter presents an iterative binding algorithm for high-level synthesis design space exploration, that simultaneously optimizes clock period and wirelength. The algorithm uses a stochastic interconnect distribution model and a top-down partition-based global placement in a novel framework to provide fast and accurate estimates for wire length and wire delays to guide register and module binding decisions during high-level synthesis. Section 6.1 introduces the problem and outlines the motivations for using stochastic wirelength estimation in high-level synthesis. Section 6.2 discusses related work and introduces the stochastic wirelength model used in the this work. Section 6.3 describes a novel technique for computing the stochastic wirelength estimate of a gate-level netlist examined during design-space exploration. Section 6.4 describes an iterative binding algorithm for wirelength and clock period optimization during high-level synthesis. Section 6.5 discusses the experimental results, and Section 6.6 concludes the chapter.

6.1 Introduction and Motivation

Advances in scaling of process technology has increased the importance of interconnect-centric approaches to VLSI design. Interconnect capacitance now forms a significant portion of the total load capacitance of a gate [5], resulting in a corresponding increase in the overall delay and power dissipation due to interconnects [106]. As a result, high-level synthesis (HLS) flows must take into account the effect of their design decisions on the wiring complexity of designs.
High-level (or behavioral) synthesis takes an abstract behavioral specification of a digital
system in a high-level programming language (e.g., C, C++) and a set of design constraints, and
finds a control sequence and a register-transfer level (RTL) structure that realizes the given
behavior [12, 13]. Figure 6.1 illustrates a typical high-level synthesis flow that creates an RTL
structural netlist from a high-level behavioral description. Logic and layout synthesis tools are
then used to assemble the RTL-netlist, thus providing a route from algorithmic behavior to
physical chip layout. In this paradigm, the application described in a high-level programming
language is processed by a compiler, which performs several optimizations such as constant
propagation, loop unrolling, and function in-lining. The compiler generates an intermediate representation of the application, capturing the control flow and data dependency within the application. A commonly used model for this intermediate representation is a control-dataflow graph (CDFG). A high-level synthesis stage follows the compiler stage and takes the CDFG as input and generates a RTL description of the design. Back-end tools perform logic synthesis and physical synthesis on this RTL description of the design, mapping it to a netlist in the target layout fabric (e.g., standard-cell, FPGA).

In a high-level synthesis flow, the RTL netlist forms the interface between HLS and physical synthesis. High-level synthesis systems have traditionally optimized the RTL structure, but the real cost of the synthesized design will ultimately be measured in the physical domain in terms of circuit delay, chip area, and power dissipation. The high-level synthesis system therefore optimizes the final cost of the solution indirectly.

A design may need to be iterated through the HLS and physical synthesis steps several times if design constraints such as timing, area, and power are not met. Due to the separation of physical synthesis from behavioral synthesis, often the impact of design decisions taken during high-level synthesis will only be known after physical synthesis, necessitating numerous design iterations to achieve design closure. To address this, feedback paths (such as A and B in Figure 6.1) between HLS and physical synthesis steps may exist to incorporate physical-level information during HLS. Use of feedback improves the interaction between HLS and physical synthesis and refines the quality of the solution generated at each stage. Feedback measures employed during HLS could use estimates of the layouts (such as floorplan or place and route information) to guide high-level design decisions. Alternatively, the post-synthesis results from physical synthesis, could be used as feedback to guide HLS decisions, if designs constraints are not met.

The primary focus of high-level synthesis in the past was optimizing logic (i.e., functional units such as adders, multipliers, registers, and multiplexers), and wire delays could be ignored.
without loss of accuracy. However, with process scaling, wire delays have become significant, shifting the focus of VLSI design from transistors to interconnects [5]. For example, in [57, 116], the authors show that interconnect delays can contribute an additional 20% to the clock period of a HLS design, and conclude that high-level synthesis tools must take physical effects into consideration if they are to produce high-quality designs. As a result, a number of researchers have worked on interconnect-aware high-level synthesis [15, 16, 17, 34, 35, 36, 37, 38, 57, 58].

The solution space that needs to be explored during high-level synthesis is usually quite large [12, 13]. Therefore, it is inefficient for a high-level synthesis flow to go through a time-consuming physical synthesis step (place and route) every time a candidate solution needs to be evaluated. Most of the previous work on interconnect-aware high-level synthesis use some form of floorplanning in conjunction with high-level synthesis, to estimate wirelengths. In these approaches, the placeable modules used by the floorplanner are RTL components (such as adders,
multipliers, and registers), which have the same level of abstraction as the functional units used to describe the RTL netlist generated in the HLS step. Using the register-transfer level of abstraction for floorplanning and wirelength estimation reduces the size of the problem by one or two orders of magnitude when compared to using a lower level of abstraction such as gate-level [12, 13]. Figure 6.2 illustrates the conventional interconnect-aware HLS flow used in these works. In this flow, during the high-level synthesis step, the operations and variables in the CDFG are mapped to RTL components. To ensure fidelity between wirelength estimates and actual wirelengths obtained after physical synthesis, the logic hierarchy of the functional units in the RTL netlist is identical to the physical hierarchy of the functional units (i.e., macro-cells) used by the floorplanner. In other words, during physical synthesis, the functional units in the RTL netlist are bound to macro-cells, available as RTL components (either pre-characterized or synthesized using layout tools). Hence, these approaches are more suited to HLS flows that use macro-cells at the granularity of RTL components for their designs.

The previous work on interconnect-aware high-level synthesis assumes that the functional units of the RTL netlist from HLS are mapped to floorplan modules at the register-transfer level of abstraction. This imposes a logic hierarchy on the layout, determined by the structure of the RTL netlist (i.e., the layout is a netlist of datapath macro-cells, as opposed to a flattened netlist of gates). However, the author of [5] shows that retaining the logic hierarchy of an RTL netlist during floorplanning and recursive synthesis can lead to results that are inferior to those produced by placement and routing of a flattened gate-level netlist, since a flattened netlist lends itself to better packing of the placeable objects. This suggests that the previous works on interconnect-aware HLS can produce suboptimal results in the final layout by restricting floorplanning or placement algorithms to strictly follow the logic hierarchy boundary imposed by the RTL netlist.

In contrast to previous works, we propose an interconnect-aware HLS flow that can be applied to non-hierarchical design flows that use completely flattened gate-level netlists for their final
layouts. Figure 6.3 illustrates the HLS flow used in our approach. Traditional floorplan-based wirelength estimation methods used in previous interconnect-aware HLS techniques cannot be applied to estimate wirelengths of such layouts during HLS since conventional floorplanning methods have unacceptable run times for rapid design space exploration when applied to flattened gate-level netlists containing tens of thousands of gates. The novelty of our approach is in the use of a Rent-based stochastic wirelength distribution model to estimate wirelengths of gate-level netlists. As shown in Section 6.3.1 of this chapter, our stochastic wirelength estimates are in very good agreement with actual wirelengths obtained from four modern standard cell placement tools. In addition, we show in Section 6.5, that the run times of our wirelength estimation technique is an order of magnitude smaller than using place and route to determine wirelengths, making our technique suitable for design space exploration during HLS.
Stochastic wirelength estimation [18, 107, 108, 109] has been proposed as a suitable technique to allow the estimation of wirelength distributions in logic gate netlists. These models are based on Rent's rule [18], which is an empirical law that relates the number of terminals in a logic block to the number of gates in the block. The models use Rent's rule to compute the wirelengths based on two empirically derived parameters, namely, Rent's exponent \((p)\) and Rent's coefficient \((k)\). During design space exploration in high-level synthesis, a variety of designs with different netlist structures are examined. We use the method proposed in [39] to dynamically extract the Rent parameters of a gate-level netlist and apply it to estimate the wiring complexity of the datapath netlists examined during design space exploration.

Estimating the achievable clock period of a design is an important issue during high-level synthesis. While some of the early work have only accounted for module delays and ignored wire delays, others [15, 16, 36, 57, 58] have used simple RTL floorplan-based models to account for wire delays when estimating the clock period. In this work, we address the problem of estimating the wiring complexity and clock period of standard-cell based designs examined during high-level design space exploration.

The main contributions of this work are:

- Use of a dynamic Rent-parameter extraction technique and stochastic wirelength estimation models to quickly evaluate the wiring complexity of RT-level netlists during design space exploration.
- An iterative HLS design-space exploration engine that uses this information to guide module and register binding decisions, with goal of minimizing the clock period.

To the best of our knowledge, the proposed work is the first to apply stochastic wirelength estimations to drive design space exploration during high-level synthesis.
6.2 Related Work

Sutherland and Oestreicher were the first to derive an upper bound for the interconnect wirelength of a square array of gates [107]. Their models assumed a random placement of the gates in the gate array, leading to overly pessimistic estimates. Donath [40, 108] used Rent's rule [18, 111] to improve on Sutherland's model, and derive a tighter upper bound for interconnect estimates for a hierarchical placement approach. However, Donath's model assumed a random placement of gates at each hierarchical placement level, leading to an over-estimation of wirelengths. Stroobandt [109] improved on Donath's model by introducing the concept of occupation probability to encapsulate the tendency of good placers to position strongly connected cells closer. Both Donath's and Stroobandt's models assume a square array of gates. Dembre [112] extended Stroobandt's model to accommodate rectangular arrays of gates. Later, Davis et al. [41] constructed a wirelength distribution model based on a non-hierarchical placement. This wirelength distribution is more accurate than the earlier models, especially for long wires. In this paper, we use the stochastic wirelength distribution model proposed by Davis et al. [41] to rapidly estimate the wirelengths of RTL netlists synthesized during high-level synthesis. Since deriving stochastic wirelength estimates are faster than creating a layout, these estimates can be used to determine the effects of high-level design decisions and use them to drive the high-level synthesis process.

The stochastic wirelength estimation used in this work is based on an empirical power law called Rent's rule [18, 111]. Rent's rule relates the number of terminals ($T$) to the number of gates ($N$) in a gate-level netlist using a simple empirical formula,

$$T = kN^p$$  \hspace{1cm} 6.1

where, $p$ is known as the Rent's constant, and $k$ is the called Rent's coefficient. The Rent's constant, whose value lies in the range $0 < p < 1$, provides an indication of the wiring complexity of a circuit, while the Rent coefficient ($k$) indicates the average number of terminals per gate. The
Rent's constant is usually determined empirically by hierarchically partitioning a netlist, recording the number of terminals created in each partition, and then performing a linear regression on these data points on a log-log plot. The slope of the log-log plot is the Rent exponent.

Davis et al. [41] proposed an interconnect distribution model for a square array of $N$ uniformly tiled gates with $\sqrt{N}$ rows and $\sqrt{N}$ columns. Wirelengths are expressed in gate pitches, with the horizontal and vertical gate pitches being one unit, so the aspect ratio of the entire gate array is also one. A continuous interconnect density function $i(l)$ is defined as the number of interconnects in this gate array, with $l$ between gate pitches $a$ and $b$ as

$$
\int_a^b i(l) \, dl
$$

The continuous interconnect density function $i(l)$ models the interconnections among the gates in the netlist. This interconnect density function has three components [41] – a gate pair structural distribution function $M(l)$, a net occupancy probability $I_{\text{exp}}(l)$, and a normalization factor $\Gamma$

$$
i(l) = \Gamma M(l) I_{\text{exp}}(l)
$$

The gate pair structural distribution function $M(l)$, which represents the number of gate pair connections with a Manhattan distance $l$, is shown below to be [41]

$$
M(l) = \begin{cases} 
\frac{|l^3|}{3} - 2\sqrt{N} l^2 + 2Nl & \text{if } 1 \leq l \leq \sqrt{N} \\
2\sqrt{N^3}/3 & \text{if } \sqrt{N} < l \leq 2\sqrt{N} \\
0 & \text{otherwise}
\end{cases}
$$
Davis also showed that the occupancy probability (probability that a certain gate pair in an optimal placement on a 2-D gate array is connected), is proportional to \( n^{2p-4} \). The total number of interconnections in a 2-D gate array [11] is \( akN(1-N^{p-1}) \).

The constant factor \( \alpha \) is equal to \( f/f+1 \), where \( f \) is the average fanout of the netlist. The final normalization factor \( \Gamma \) can be found by integrating \( i(l) \) over the entire range of wirelength.

\[
\Gamma = \frac{akN(1-N^{p-1})}{\int_{1}^{2\sqrt{N}} M |l| l^{2p-4} dl}
\]

Then, the average interconnect length in a netlist can be derived as [41],

\[
L_{avg} = \frac{L_{total}}{I_{total}} = \frac{\int_{1}^{2\sqrt{N}} i(l) |l| dl}{\int_{1}^{2\sqrt{N}} i |l| dl}
\]

where \( L_{total} \) represents the total estimated wirelength for the given netlist, and \( I_{total} \) is the total number of nets in the netlist. The total wirelength for the netlist is obtained by integrating the closed-form expression for the cumulative wirelength,

\[
L_{total} = \int_{1}^{2\sqrt{N}} i(l) |l| dl
\]

The expression given by Equation 6.6 is used in our work to estimate the average wire length of a given topological netlist.

### 6.3 Rent Rule-based Stochastic Wirelength Estimation

The complexity of the interconnection topology of a digital circuit is captured well by Rent's rule [18]. This empirical law states that when a circuit is partitioned into a set of more or less
equally sized modules subject to a cut-size minimization objective, there exists a relationship between the average terminal count $T$ and the module size $N$, as illustrated in Equation 6.1. Rent's rule has been the basis of several *a priori* wirelength estimation models [41, 108, 109]. These wirelength estimation techniques use Rent's rule as a simple model for the partitioning behavior of logic circuits.

Figure 6.4 illustrates the relationship between the number of terminals and the number of gates of a digital circuit, obtained by recursively partitioning the circuit in a top-down fashion. In Figure 6.4, the data points obtained from the partitioning process are plotted on a *log-log* plot, where each data point represents the average terminal count for sets of modules with the same gate count.

Rent's rule provides a fairly accurate approximation for a broad range of module sizes, but significant deviations do occur, especially for large as well as very small module sizes. These regions of deviations are called Region II (for large module sizes) [18], and Region III (for very small module sizes) [109]. The Rent plot shown in Figure 6.4, shows that the deviation
Figure 6.5  Stochastic Wirelength Estimation for High-Level Synthesis
corresponding to Region II is present for this circuit. The source of the deviation for Region II is mainly due to I/O pin limitations on a module. The range of module sizes for which Rent's rule is applicable is termed as Region I.

Stochastic wirelength estimation models estimate wirelengths of gate-level netlists, based on (1) the number of gates, (2) the Rent exponent, and (3) the Rent coefficient. These parameters can be directly extracted from a gate-level netlist. In this work, we apply the stochastic wirelength estimation model proposed by Davis et al. [41] to estimate the wiring complexity of high-level synthesis (HLS) benchmarks at the gate-level.

Figure 6.5 illustrates the stochastic wirelength estimation technique used in this work. The RTL wirelength estimation technique first flattens a given RTL netlist to its gate-level equivalent. The gate-level netlist is then recursively bi-partitioned, using hMetis [110], with cut-size minimization and balanced partitions, as the partitioning criteria. The recursive bi-partitioning step results in a partition-tree of the gate-level netlist.

To extract the Rent parameters of a given netlist, we developed a Dynamic Rent Parameter Extraction (DRPE) algorithm, shown as Algorithm-1, which can be used to determine the Rent Exponent \( p \) and Rent coefficient \( k \) of a gate-level netlist obtained by flattening the RTL netlist created in the HLS step. The DRPE starts by recursively bi-partitioning the gate-level netlist. During partitioning, the two sub-circuits resulting from each bi-partition are analyzed to obtain their gate-count and terminal-count. The gate and terminal counts of all partitions obtained through recursive bi-partitioning of the gate netlist are used as data points on a log-log plot, to determine the Rent parameters of the netlist. A linear regression is applied to find the slope of the fitted line on this log-log plot. The slope represents the Rent exponent, while its y-intercept represents \( \log k \), where \( k \) is the Rent coefficient.

A linear regression of the fitted line on a log-log plot of the terminal and gate-counts of recursively partitioned gate-level netlists, tend to fall into two categories [18], termed Region I
Figure 6.6 Dynamic Rent Parameter Extraction

and II. Region II corresponds to netlist partitions closer to the root of a netlist partition-tree, while Region I relates to partitions at lower levels of the tree. Extensive studies [41,108,109] have shown that for many real designs, netlist partitions at the lower levels of a partition-tree tend to closely follow Rent's rule. However, for partitions higher up in the partition-tree (partitions belonging to Region II), the number of terminals decreases due to factors such as limited number of external I/O pins. The data points corresponding to these partitions do not follow Rent's rule. Only those data points that belong to Region I are used to determine the Rent parameters of a netlist. The stochastic wirelength distribution model of Davis et al. is then used to compute the total wirelength of an RTL netlist. As explained in Section 6.2, this interconnect distribution model provides a closed-form expression for the estimated wirelength of a gate-level netlist, in terms of the total number of gates, and the Rent parameters extracted from the netlist.
In our experiments, we validated the accuracy of the stochastic wirelength estimation model on a set of data dominated HLS benchmarks. For our experiments, a set of ten data path designs, chosen from four different data-dominated DSP benchmarks were used. Table 6.1 shows the circuit characteristics of these benchmarks. In the table, column 1 lists the name of the benchmarks. In the column, IIR stands for an 8-point Infinite Impulse Response filter, EWF represents a five-point Elliptic Wave filter, DCT represents a 1-point 8 x 8 Discrete Cosine Transform filter, and FIR represents a 16-point Finite Impulse Response filter. Datapaths with different scheduling and allocation combinations were designed for these benchmarks. Logic synthesis was then performed on these data path designs, to yield their gate-level equivalents. The different data path designs are identified by their design IDs specified in column 2. Columns 3 and 4 list the number of gates and signal nets respectively, in each of the data paths. Columns 5 and 6 respectively list the Rent exponent ($p$) and Rent coefficient ($k$) extracted from these data path netlists.

The stochastic wirelength estimation model assumes that Rent's rule provides a good approximation of a given circuit's partitioning characteristics. Our experiments indicate that most

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>ID</th>
<th>Gates</th>
<th>Nets</th>
<th>$p$</th>
<th>$k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR</td>
<td>1</td>
<td>1372</td>
<td>1342</td>
<td>0.46</td>
<td>2.41</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>1394</td>
<td>1345</td>
<td>0.42</td>
<td>2.93</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1400</td>
<td>1320</td>
<td>0.41</td>
<td>2.52</td>
</tr>
<tr>
<td>EWF</td>
<td>4</td>
<td>1096</td>
<td>1023</td>
<td>0.63</td>
<td>2.28</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1712</td>
<td>1587</td>
<td>0.57</td>
<td>2.21</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2682</td>
<td>2575</td>
<td>0.58</td>
<td>2.64</td>
</tr>
<tr>
<td>DCT</td>
<td>7</td>
<td>2152</td>
<td>2010</td>
<td>0.51</td>
<td>3.03</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3152</td>
<td>3039</td>
<td>0.49</td>
<td>2.96</td>
</tr>
<tr>
<td>FIR</td>
<td>9</td>
<td>2432</td>
<td>2386</td>
<td>0.48</td>
<td>2.52</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2266</td>
<td>2342</td>
<td>0.48</td>
<td>2.86</td>
</tr>
</tbody>
</table>
Figure 6.7 Rent Characteristic Plots for HLS Benchmarks
data-dominated HLS benchmarks closely follow Rent's rule. Figure 6.7 shows the Rent plots for the four HLS benchmarks evaluated in our experiments. These plots empirically demonstrate the validity of Rent's rule for data-dominated HLS benchmarks. From column 5 in Table 6.1, we can observe that the Rent exponent ($p$) for these benchmarks varies from 0.41 to 0.63, depending on the datapath design. From [108], we can state that for this range of Rent exponents, the interconnect complexity grows linearly with circuit size.

### 6.3.1 Experimental Validation of Stochastic Wirelength Estimation

To determine the Rent parameters of an RTL netlist, we first flattened the netlist to their gate-level description of the circuit. All power, ground, and clock nets were then excluded from the circuit, since these nets are usually treated differently from signal nets during physical synthesis. A hypergraph description of the gate-level netlist was then recursively partitioned using hMetis [110], creating a partition tree. This partition tree was then analyzed, as described in Algorithm-1, to determine the Rent parameters of the netlist. These Rent parameters were then used in the wirelength estimation model, to estimate the total wirelengths of each of the datapath designs.

The estimated wirelengths were compared with wirelengths obtained from three state-of-the-art academic standard-cell placement tools, and Cadence Silicon Ensemble (a modern commercial tool). The academic cell placement tools used in our study were DRAGON [113], CAPO [114], and FengShui [115].

Figure 6.8 compares our estimated wirelengths of HLS benchmarks with wirelengths from layouts created using the following standard-cell placement tools: Cadence Silicon Ensemble, DRAGON, CAPO, and FengShui. From the figure, it is evident that our stochastic wirelength estimates agree closely with actual wirelengths reported by these placement tools. Table 6.2 shows the percentage error of our wirelength estimates with respect to the wirelengths reported by
Figure 6.8 Comparison of Wirelength Estimates with Post-Placement Wirelengths of DRAGON, CAPO, FengShui, and Silicon Ensemble

these tools. In the table, column 1 lists the names of HLS benchmarks, while columns 2, 3, and 4 list the wirelength estimation errors. For the HLS benchmarks used in our experiments, the wirelength estimation errors were less than 15%, with a maximum error of 14.2% for the IIR benchmark placed and routed using Silicon Ensemble. The average estimation error with respect to placements with Silicon Ensemble was 11.6%. The average errors with respect to DRAGON,
Table 6.2 Estimation Errors of Stochastic Wirelengths with Respect to Measured Wirelengths of Standard-Cell Placers

<table>
<thead>
<tr>
<th>Design</th>
<th>Cadence Silicon Ensemble</th>
<th>Dragon</th>
<th>CAPO</th>
<th>FengShui</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR-1</td>
<td>11.34</td>
<td>7.43</td>
<td>9.38</td>
<td>5.48</td>
</tr>
<tr>
<td>IIR-2</td>
<td>10.39</td>
<td>6.48</td>
<td>10.39</td>
<td>7.46</td>
</tr>
<tr>
<td>IIR-3</td>
<td>14.16</td>
<td>11.08</td>
<td>12.15</td>
<td>9.02</td>
</tr>
<tr>
<td>EWF-1</td>
<td>8.28</td>
<td>7.39</td>
<td>11.32</td>
<td>9.35</td>
</tr>
<tr>
<td>EWF-2</td>
<td>12.23</td>
<td>10.26</td>
<td>8.29</td>
<td>7.31</td>
</tr>
<tr>
<td>EWF-3</td>
<td>9.08</td>
<td>9.06</td>
<td>10.07</td>
<td>8.05</td>
</tr>
<tr>
<td>DCT-1</td>
<td>13.35</td>
<td>11.34</td>
<td>8.29</td>
<td>9.31</td>
</tr>
<tr>
<td>DCT-2</td>
<td>10.81</td>
<td>8.79</td>
<td>9.81</td>
<td>6.78</td>
</tr>
<tr>
<td>FIR-1</td>
<td>12.60</td>
<td>9.65</td>
<td>10.63</td>
<td>7.66</td>
</tr>
<tr>
<td>FIR-2</td>
<td>13.43</td>
<td>10.37</td>
<td>11.39</td>
<td>8.32</td>
</tr>
<tr>
<td>Average</td>
<td>11.57%</td>
<td>9.18%</td>
<td>10.2%</td>
<td>7.84%</td>
</tr>
</tbody>
</table>

CAPO, and FengShui were respectively 9.2%, 10.2%, and 7.8%. Though these estimation errors are reasonable for HLS design space exploration, from our experiments we observe that for all benchmarks, the stochastic wirelength estimation model tends to underestimate the wirelength, when compared to real wirelengths from these placement tools. This is primarily due to the fact that the stochastic wirelength distribution model of Davis et al. does not account for wiring congestion, whereas these placement tools adjust the placement of gates to mitigate wiring congestion-hotspots [113, 114, 115], resulting in longer wirelengths compared to an idealized placement that ignores wiring congestion.

The good agreement between our wirelength estimates and real wirelengths could be attributed to the fact that all the designs used in our experiments satisfy the general assumptions made in the Davis model. The gate-layout style used in our experiments for wirelength measurements comprise of a homogeneous array of logic gates of similar size. Note that the wirelength measures in our experiments only include signal nets in a design and do not include clock and power nets, which are usually treated differently from signal nets in real designs.
The wirelength estimation model used in our experiments derives the Rentian properties of a gate-level netlist using a recursive partitioning process. This process closely matches most modern cell placement tools, all of which use a top-down recursive partitioning based approach to scale to large problem instances. All four cell placers used in this study use variants of the basic recursive partitioning-based approach to perform the global placement of gates. This underlying similarity between the approaches leads to a good match between estimated and measured wirelengths.

The main advantage of using the estimation model is the reduced computational effort needed to determine the total wirelength, when compared to performing a full place-and-route of a netlist. This speed-up advantage is exploited in the iterative binding algorithm for design space exploration, as proposed in the next section.

### 6.4 Iterative Binding for High-Level Design Space Exploration

We propose an HLS design space exploration algorithm that uses accurate estimates of the clock period and wirelength to guide HLS binding decisions. During design space exploration, hundreds of candidate designs are examined, in an effort to identify solutions that meet all design specifications. Any estimation technique used for design space exploration must be fast, in addition to being reasonably accurate. The use of stochastic wirelength estimates and a multi-level partitioning-based global cell-placer to provide wire delay bounds for clock period estimation, enables our technique to be both fast and accurate.

Figure 6.9 illustrates the overall flow of our iterative binding algorithm. The input to the algorithm is a dataflow graph (DFG) specifying the behavior of the circuit to be synthesized, and a user specified time or resource constraint for the design. The algorithm starts by creating an initial datapath netlist which is then iteratively improved for clock period and wirelength through a series of HLS re-binding moves. To create an initial solution, a force-directed scheduler [30] is
Figure 6.9 Overall Flow of Iterative Binding Algorithm for Wirelength and Clock Period Improvement
Figure 6.10  Cost Function Computation in the Iterative Binding Algorithm
used to schedule the operations of the input DFG. This is followed by a clique partitioning-based
euristic [79] for the initial binding of the DFG operations to the datapath functional units. We
use the left-edge algorithm of [117] to allocate and bind DFG variables to datapath registers in
this initial solution. The RTL netlist of this initial solution is then iteratively improved using a
series of re-binding moves in an effort to improve the total wirelength and the clock period of the
circuit.

The algorithm assumes the availability of an RTL module library describing the gate-level
netlists of all datapath functional units. Figure 6.10 illustrates the procedure used to evaluate the
solution cost of datapaths found during design space exploration. This cost, in turn, is used to
guide HLS re-binding decisions. The algorithm consists of three main steps: (1) estimating total
wirelength of the gate-level netlist using the Davis stochastic wirelength distribution model,
(2) determining cell locations in the core placement area through global placement, and
(3) determining all register-to-register delays in the datapath to estimate the clock period. To
estimate wirelengths of the gate-level datapath netlists during the re-binding moves, the
algorithm first flattens the input RTL netlist to its gate-level equivalent using the RTL module
library. The total layout area required to place the cells of the netlist is also determined using the
RTL module library data. The gate-level netlist is then recursively bi-partitioned with hMetis
[110] using cut-size minimization and area-balancing as the partitioning criteria. At the same
time, the layout area is recursively bi-partitioned into placement regions, each of which is
assigned a corresponding sub-circuit during partitioning. We then apply Algorithm-1 to the
partitioned sub-circuits to estimate total wirelength. Wire delays in the between datapath modules
in the RTL netlist, and the clock period are estimated using the technique described in Section
6.4.3. The following sub-sections elaborate on each of these steps.
6.4.1 Estimating Placement of RTL Modules

The partitioning step to determine Rent parameters works on a topological representation of the netlist. However, to estimate the wire delays associated with data transfers between RTL modules in a datapath (and hence estimate the clock period), we need geometrical information on the relative locations of these modules on the chip floorplan. We obtain this information by a top-down recursive global placement of the gates in the netlist, while bi-partitioning the netlist for Rent parameter extraction.

To estimate the placement of RTL modules, we start with a core layout area whose dimensions are estimated based on the gate-count of the flattened RTL netlist. Figure 6.11 illustrates the recursive partitioning-based global placement method used in our algorithm to estimate the location of individual gates in the input netlist. In this method, when a netlist is recursively partitioning to extract its Rent parameters, the layout area is also recursively partitioned into global placement bins, each of which is assigned to a partitioned netlist. For example, in Figure 6.11, after the first partitioning step, the two sub-circuits B and C are assigned to the two bins B and C in the layout. Similarly, after the second partitioning step, the resulting sub-circuits D, E, F, and G are assigned the bins D, E, F, and G in the layout. This partitioning of the gate-level netlist and the layout area results in a partition-tree of the netlist, and a global placement of the gates into bins in the placement area. We assume that all gates assigned to a global placement bin, are located at the center of the bin. As a trade-off between speed and accuracy, our global placement procedure can stop the partitioning process when the bin-size is small enough to allow estimating the approximate location each gate placement with reasonable accuracy. In our algorithm, the accuracy of the wire delay computation is primarily determined by the granularity of the global placement bins (and therefore, the number of levels to which a given netlist is partitioned).
Figure 6.11  Layout-Area Partitioning for Estimating Gate Placements

Figure 6.12  Determining Coordinates and Dimensions of RTL Modules
Once the global placement bins for all gates are determined, the next step involves estimating the dimensions and relative locations of the datapath RTL modules on the chip layout. We make the reasonable assumption that an efficient cell placement tool will place all gates belonging to an RTL module in close proximity. With this assumption, we can approximately estimate the dimensions and relative locations of the RTL modules by determining the smallest rectangular bounding box enclosing all the logic gates belonging to each RTL module. Figure 6.12 gives an example of this for two RTL modules. After determining the locations and dimensions of the datapath modules, we can estimate the inter-module wire lengths and wire delays used for clock period estimation, as explained in the next sub-section.

6.4.2 Clock Period Estimation

In our work, we assume a point-to-point multiplexer-based architecture for interconnects between datapath modules. To estimate the clock period, the scheduled DFG is analyzed to determine all data transfers present in a given schedule. The HLS binding sub-task determines the binding of RTL resources to these DFG data transfers. Using this binding information, and the placement information of RTL modules, we can determine the flow of data among the modules on the chip floorplan, and the corresponding wire delays. Algorithm-2 outlines the procedure for computing data transfer delays, given a scheduled DFG and the relative locations of RTL modules.

In the scheduled DFG, each operation is analyzed to identify the module, and source and sink registers used to perform the operation. This provides information on the flow of data during this operation's execution. As shown in Figure 6.13, in a multiplexer-based datapath model, for each DFG operation, data flows from a source register, through an input multiplexer, to a datapath functional unit (such as an ALU, multiplier etc.) The output from the functional unit is then latched into a destination register, through its input multiplexer. In this datapath model, the clock
Figure 6.13 Datapath Model for Clock Period Estimation

The clock period $T_{clk}$ is calculated as

$$T_{clk} = \text{MAX} \left( T_{FU} + T_{reg} + T_{mux1} + T_{mux2} + T_{wires} \right)$$

where $T_{FU}$ represents the maximum critical path delay among all functional units in the design, $T_{reg}$ represents the register set-up and hold times, and $T_{mux1}$ ($T_{mux2}$) represent the multiplexer delays. Mux1 and Mux2 are multiplexers that arise due to register sharing and functional unit sharing in the design respectively.
Algorithm-2  Determine clock period of datapath

Inputs:  
1. Scheduled dataflow graph \( G(V,E) \)
2. HLS Binding of \( G(V,E) \)
3. Coordinates of datapath RTL modules

Output:  Clock period \( CP \) of RTL datapath

for each DFG operation \( V_k \) in \( G(V,E) \)
   1. Determine the RTL module \( M_k \) bound to \( V_k \)
   2. Determine the Register bindings \( R1_k \) and \( R2_k \) of the two inputs of \( V_k \)
   3. Determine the Register binding \( R3_k \) of the output of \( V_k \)
   4. Identify all input Multiplexers \( MX_k \) to these modules.
   5. Look up the coordinates of the RTL modules \( M_k, R1_k, R2_k, R3_k, \) and \( MX_k \).
   6. Compute path delay \( d1 \) from \( R1_k \) to \( R3_k \)
   7. Compute path delay \( d2 \) from \( R2_k \) to \( R3_k \)

Set \( CP = \) maximum path delay found in \( G \)
return \( CP \)

Figure 6.14  Clock Period Estimation

Figure 6.15  Estimating Module-to-Module Wire Delays
The process of determining the clock period for an RTL datapath is illustrated in Algorithm-2. In this algorithm we model all nets between RTL modules as 2-point nets. The algorithm begins by identifying each RTL module pair between which data flows when executing a DFG operation. To estimate the wire delays between a pair of communicating modules, we first determine the relative locations and dimensions of these modules, using the method described in Section 6.4.1. The half-perimeter of the smallest rectangular bounding box that completely encloses the module-pair is then used as an estimate of the wirelength between the pair of module. The Elmore delay model is used to compute wire-delays between communicating module pairs. To model module delays, we use a pre-characterized RTL module library implemented in the target cell library. The clock period is then computed by examining valid path delays on a directed graph built using the delay models of the RTL module instances present in the datapath and edges representing the physical connectivity between them.

Figure 6.15 shows a simple example of estimating wire lengths of nets between communicating RTL modules. The figure shows a scheduled DFG consisting of four operations. Assume that operation \textit{op2} is bound to multiplier \textit{Mult-1}, and the output of \textit{Mult-1} is stored in register \textit{Reg-1}. To compute the wire delay between the \textit{Mult-1} and \textit{Reg-1}, we first estimate the relative locations of these modules on the chip layout by using the technique explained in Section 6.4.1. In Figure 6.15, these are shown as shaded boxes in the chip floorplan on right. The estimated wire length of the net connecting these two modules is half perimeter of the smallest bounding box (HPBBOX) enclosing these two modules, as shown in Figure 6.15.

6.4.3 Iterative Binding Algorithm

The design space exploration framework described in this paper is based on an iterative binding algorithm that uses the wirelength and clock period estimation techniques detailed in Section 6.4.2, to evaluate binding solutions examined by the algorithm. The algorithm starts with
an initial binding and datapath, created using a constructive technique, and improves this solution through a greedy acceptance criterion. Using a greedy algorithm is acceptable for early design space exploration, since the main goal at this stage is to quickly explore a large design space, weeding out inferior solutions, and providing a reasonably good starting point for a more detailed implementation of the design.

Algorithm-3 (and Figure 6.9) shows the approach used in our iterative binding framework. The algorithm accepts a scheduled dataflow graph as input, and returns a datapath optimized for wirelength and clock period. An initial solution for the scheduled dataflow graph is first found. In our case, this was done by performing scheduling using either ASAP or Force-directed scheduling, and module allocation/binding using Clique Partitioning. Note that the purpose of this phase is only to get a feasible solution that satisfies user constraints. Having obtained an initial solution that meets user constraints, the following iterative improvement phase improves the architecture by reducing the wirelength and clock period while still satisfying user constraints. To evaluate the quality of solutions, we use the following weighted cost function of the clock period and total wirelength:

\[ C_{\text{Soln}} = w_1 \cdot T_{CP} + w_2 \cdot L \]

where \( C_{\text{Soln}} \) is the cost, \( T_{CP} \) is the clock period, and \( L \) is the total wirelength. Here, \( T_{CP} \) is normalized with respect to the clock period of the initial solution. Similarly, \( L \) is normalized with respect to the total wirelength of the initial solution. In our experiments, we set \( w_1 = 0.60 \) and \( w_2 = 0.40 \). At each stage, the best solution seen thus far is stored. If the result of the current iterative improvement phase actually improves over the best solution seen previously, then the best solution is set to the current solution.

The iterative improvement stage is repeated over multiple passes, where a sequence of HLS binding moves is applied to the current solution in each pass, in an attempt to find a binding that
**Algorithm 3**  Iterative Binding for Clock Period Optimization

**Input:** Scheduled Dataflow Graph $G(V,E)$  
**Output:** Optimized RTL Netlist of Datapath

1. Read scheduled DFG  
2. Use clique partitioning to bind RTL resources to $G$ to create initial solution  
3. Evaluate its cost function  

while (cost improvement = TRUE  
or number of binding moves < MAX_MOVES)  
{
    sort DFG operations $V_k$ of $G$ by their delay  
    for each $V_k$ in $G$ do  
    \[ i_1 = \text{input operand 1 of } V_k \]  
    \[ i_2 = \text{input operand 2 of } V_k \]  
    \[ o_k = \text{output produced by } V_k \]  

    \[ list(R) = \text{compatible RTL resources for } V_k \]  
    for each RTL resource $r_k \in list(R)$ do  
    \[ (1) \text{tentatively bind } V_k \text{ to } r_k \]  
    \[ (2) \text{evaluate cost of new solution } S_k \]  
    \[ (3) \text{if } cost(S_k) < \text{cost of current solution} \]  
    \[ \text{then accept new binding for } V_k \]

    \[ list(R) = \text{compatible RTL resources for } i_1 \]  
    for each RTL resource $r_k \in list(R)$ do  
    \[ (1) \text{tentatively bind } i_1 \text{ to } r_k \]  
    \[ (2) \text{evaluate cost of new solution } S_k \]  
    \[ (3) \text{if } cost(S_k) < \text{cost of current solution} \]  
    \[ \text{then accept new binding for } i_1 \]

    \[ list(R) = \text{compatible RTL resources for } i_2 \]  
    for each RTL resource $r_k \in list(R)$ do  
    \[ (1) \text{tentatively bind } i_2 \text{ to } r_k \]  
    \[ (2) \text{evaluate cost of new solution } S_k \]  
    \[ (3) \text{if } cost(S_k) < \text{cost of current solution} \]  
    \[ \text{then accept new binding for } i_2 \]

    \[ list(R) = \text{compatible RTL resources for } o_k \]  
    for each RTL resource $r_k \in list(R)$ do  
    \[ (1) \text{tentatively bind } o_k \text{ to } r_k \]  
    \[ (2) \text{evaluate cost of new solution } S_k \]  
    \[ (3) \text{if } cost(S_k) < \text{cost of current solution} \]  
    \[ \text{then accept new binding for } o_k \]
\}

return best solution in terms of clock period and total wirelength

---

Figure 6.16 Iterative Binding Algorithm
improves the clock period and wiring complexity. Since the datapaths explored in this work are based on multiplexer-based point-to-point interconnects, the interconnect complexity is strongly dependent on resource binding. Each pass of the iterative binding algorithm starts by sorting all DFG operations of the scheduled dataflow graph in decreasing order of their register-to-register delay (computed by Algorithm-2) on the datapath of the current solution. Then, starting with the DFG operation with the maximum delay, a series of HLS re-bindings are tried as tentative moves. If the result of a re-binding move improves over the best solution seen previously, then the best solution is set to the current solution, and the tentative re-binding is made permanent. An iterative improvement pass is complete when either a MAX_MOVES number of re-binding moves are tried, or all DFG operations in the sorted list have been processed in the current pass. The algorithm terminates when no improvement over the current best solution is found in one complete pass of the iterative improvement phase.

6.4.4 Runtime-vs-Accuracy Trade-off of Estimation Algorithm

The run-time of the dynamic Rent-parameter extraction technique is determined by number of levels to which the gate-level netlist is partitioned. This is in turn, dictated by the minimum number of data points needed for computing the Rent parameters, and the granularity of the placement bins used in estimating the positions of gates. To accurately determine the Rent parameters, a certain minimum number of partitions is required, to obtain an adequate number of data points in Region I of the Rent's curve [18]. In addition, to be able to uniquely identify all RTL modules, the smallest bin-size on the partitioned layout must be no larger than the size of the smallest RTL module in the datapath. For runtime efficiency, the partitioning process in our algorithm is stopped when an adequate number of data points needed to compute Rent parameters and RTL module locations, are obtained.
Through experiments, we found that obtaining five or more data points in Regions-I of a net-list's Rent's curve was adequate to accurately determine the Rent's parameters. The number of bi-partitions needed to uniquely identify all RTL modules in the datapath, can be computed from the ratio of the gate-count of the smallest datapath RTL module, to the total gate-count, as shown in the following equation:

\[
\text{Number of partition-levels} = \log(N_s) - \log(N_T)
\]

where \( N_s \) is the gate-count of the smallest RTL module, and \( N_T \) is the total number of gates in the datapath (assuming uniformly sized gates).

Figure 6.17 illustrates the percentage error in wirelength estimation for the DCT-1 benchmark, as a function of the levels of partitioning to which the gate-level netlist is partitioned during wirelength estimation. A similar trend was observed for all the other benchmarks. From the figure it is evident that the accuracy of wirelength estimation improves with the number of levels to which the partitioning process is carried out on the gate-level netlist. For our experiments, we used a partitioning depth of 10 for all the benchmarks.

![Figure 6.17](image.png)
6.5 Experimental Results

The methods described in this paper were implemented and tested on a Linux workstation running on a 1.86 GHz Intel Core2 Duo CPU with 2GB RAM.

In this section, we present results from our experiments on clock period optimization by the proposed iterative binding algorithm. The algorithm was tested on four data-intensive HLS benchmarks – 8-point IIR filter, 16-point FIR filter, 5-point elliptic wave filter (EWF), and an 8 x 8 Discrete Cosine Transform (DCT) filter. Our algorithm accepts inputs in the form of dataflow graphs. For our experiments, we used ASAP and force-directed scheduling to schedule the input dataflow graphs, for which the initial resource allocation and binding was done using a clique-partitioning heuristic. The clock period of this initial solution was then iteratively improved by our iterative binding algorithm.

Table 6.3 illustrates the clock periods of the initial and best bindings for the benchmarks tested. In the table, column 1 lists the name of the benchmark. Column 2 shows the estimated clock period of the initial solution provided to the iterative binding algorithm, while column 3 shows the clock period of the best binding solution found. Column 4 shows the percentage improvement in the clock period. The percentage improvements in the clock period vary from 6.65% for the DCT-1 benchmark design, to 14.42% for the DCT-2 benchmark design, with an average improvement of 9.55%.

Figures 6.18 and 6.19 show the convergence plots for our iterative binding algorithm, illustrating the trend in the clock period and total wirelength of the best solution found by the iterative binding algorithm, during design space exploration. The x-axis in these figures represent the number of binding moves attempted during the iterative improvement phase, and the y-axis represents the clock period and total wirelength for the best binding solution found. For these experiments, we set the maximum number of binding moves attempted to 200. These figures
Table 6.3  Clock Period Improvement by the Iterative Binding Algorithm

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Initial CP (ps)</th>
<th>Best CP (ps)</th>
<th>% improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR-1</td>
<td>1261</td>
<td>1171</td>
<td>7.14</td>
</tr>
<tr>
<td>IIR-2</td>
<td>1227</td>
<td>1115</td>
<td>9.13</td>
</tr>
<tr>
<td>IIR-3</td>
<td>1133</td>
<td>1009</td>
<td>10.94</td>
</tr>
<tr>
<td>EWF-1</td>
<td>2110</td>
<td>1884</td>
<td>10.8</td>
</tr>
<tr>
<td>EWF-2</td>
<td>1586</td>
<td>1397</td>
<td>11.92</td>
</tr>
<tr>
<td>EWF-3</td>
<td>1347</td>
<td>1243</td>
<td>7.72</td>
</tr>
<tr>
<td>DCT-1</td>
<td>2796</td>
<td>2610</td>
<td>6.65</td>
</tr>
<tr>
<td>DC T-2</td>
<td>2168</td>
<td>1855</td>
<td>14.42</td>
</tr>
<tr>
<td>FIR-1</td>
<td>1332</td>
<td>1233</td>
<td>7.38</td>
</tr>
<tr>
<td>FIR-2</td>
<td>1127</td>
<td>1021</td>
<td>9.41</td>
</tr>
</tbody>
</table>

Avg: 9.55%

illustrate that the clock period of an initial binding solution can be improved significantly through a layout-aware binding. We also observe that for all the benchmarks the total wirelength increases, albeit by a small amount, with the increase being typically less than 15%. Clearly, the improvements in the clock period were achieved without any significant sacrifice in the total wirelength.

The improvements in the clock period through binding could be attributed to a better wire distribution of the final layout, due to a more balanced binding of DFG operations and variables among the data resources. The binding moves attempted by the algorithm try to identify natural clusters of connected modules in the datapath that could potentially lead to smaller wire delays for data transfers and lower wire congestion.

Table 6.4 shows the runtime for design space exploration performed by our algorithm for the tested benchmarks. Column 2 indicates the total number of datapath designs examined by the algorithm, and column 3 states the CPU time in minutes and seconds. Each binding move attempted by the algorithm involves creating a new datapath architecture, which is recursively partitioned and global placed by the algorithm, to estimate the wirelength and clock period. In a
Figure 6.18  Convergence Plots of the Iterative Binding Algorithm for IIR and EWF Benchmarks
Figure 6.19  Convergence Plots of the Iterative Binding Algorithm for DCT and FIR Benchmarks
Table 6.4  CPU Runtimes for the Iterative Binding Algorithm

<table>
<thead>
<tr>
<th>HLS benchmark</th>
<th>Number of bindings moves</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIR-1</td>
<td>100</td>
<td>6m:24s</td>
</tr>
<tr>
<td>EWF-1</td>
<td>200</td>
<td>11m:04s</td>
</tr>
<tr>
<td>FIR-1</td>
<td>200</td>
<td>22m:49s</td>
</tr>
<tr>
<td>DCT-1</td>
<td>200</td>
<td>34m:07s</td>
</tr>
</tbody>
</table>

typical physical synthesis step, every cell-placement and routing step would take several minutes. Hence, for a traditional synthesis flow, evaluating each binding move, by itself would take several minutes for synthesis and timing analysis. Figure 6.20 compares the run times for a traditional HLS design space exploration using standard-cell place & route, and the proposed iterative binding using stochastic wirelength estimation. The iterative binding algorithm proposed in this paper performs the same task almost an order-of-magnitude faster than the traditional synthesis flow.

Figure 6.20  Runtime Comparison Between HLS Design Space Exploration with Traditional Place & Route and the Proposed Stochastic Wirelength Estimation Method
6.6 Conclusions

In this chapter, we presented an iterative binding algorithm for clock period optimization, that uses stochastic wirelength models to estimate the total wirelength of cell-based designs, and a top-down partitioned based RTL placement to estimate the clock period. Use of these estimates to guide HLS binding decisions enables our approach to achieve an order-of-magnitude improvement in the search time for HLS design space exploration. Our wirelength estimates are within 15% of Dragon, Capo, FengShui, and Cadence Silicon Ensemble. Experiments on dataflow intensive HLS benchmarks show that our iterative binding algorithm can improve the clock period of a datapath by an average of 9.6%, with minimal impact on the wirelength.
CHAPTER 7
A GENETIC ALGORITHM FOR HIGH-LEVEL DESIGN SPACE EXPLORATION

High level synthesis comprises of interdependent tasks such as scheduling, allocation, and module selection. In this chapter, we present a framework for efficient design space exploration during high-level synthesis of datapaths for data-dominated applications. The framework uses a Genetic Algorithm (GA) to concurrently perform scheduling, allocation, and binding, with the aim of finding solutions that lead to superior designs while considering user-specified latency and area constraints. The GA uses a multi-chromosome representation to encode datapath schedules and module allocations, and efficient heuristics to minimize functional and storage area costs, while minimizing circuit latencies. The framework provides the flexibility to perform resource-constrained scheduling, time-constrained scheduling or a combination of the two, using a simple and fast list scheduling technique. A graded penalty function is used as an objective function in evaluating the quality of designs, to enable the GA to quickly reach areas of the search space where designs meeting user specified criteria are most likely to be found. Since GAs are population-based search heuristics, a unique feature of our framework is its ability to offer a large number of alternative datapath designs, all of which meet design specifications but differ in module, register, and interconnect configurations. Many experiments on well-known benchmarks show the effectiveness of our approach.

The chapter is organized as follows. Section 7.1 discusses the motivations for this work and Section 7.2 outlines related work on GA-based approaches to high-level synthesis. Section 7.3, describes our GA-based approach to solve this problem; we discuss the encoding strategies,
various crossover and mutation operators, initialization and selection strategy, the heuristics used to interpret the encodings, and the fitness function used to evaluate solutions. In Section 7.4 describes the experimental setup, the benchmark problems, and experimental results. Section 7.5 discusses the experimental results and compares them with those of other high-level synthesis techniques from the literature, and finally in Section 7.6 summarizes the paper and draw conclusions based on the experimental results.

7.1 Motivations for this Work

The domain of VLSI design is multi-objective in nature, often with the need to trade-off several conflicting design objectives such as chip area, circuit speed, power dissipation, and manufacturing cost. For many VLSI designs, speed and cost requirements vary over different market segments of the target application. High-end segments, such as server CPUs and hardware router ASICs demand high performance, while low-end but high volume markets such as ASICs for cell-phones demand low cost and power consumption (but not necessarily high performance). Hence, depending on the application, there is a strong motivation to explore the design space of alternative circuit implementations before finalizing one that meets all design trade-offs. Design space exploration has a higher payoff when done during high-level synthesis than at lower levels of abstractions such as logic or transistor levels. Design space exploration in high-level synthesis involves evaluating different operator schedules, module allocations, and module binding alternatives to a given design specification.

Datapath synthesis can be modeled as the process of searching a complex multi-dimensional space represented by the set of possible schedules, allocations, and bindings that can realize a given behavioral specification. Under latency and resource constraints, the sub-tasks of scheduling, allocation, and binding are known to be NP-hard [12, 144]. There is a strong inter-dependence between these synthesis subtasks, and it is not clear in which order they should be
performed [12]. For example, the allocated number of hardware units constrain the maximum number of concurrent operations that can be scheduled, but the minimum number of hardware units that need to be allocated to meet design timing constraints cannot be known until the scheduling subtask is completed. In addition, decisions taken in any of the subtasks significantly impact decisions taken in subsequent subtasks, and often determining the quality of the solutions found. To simplify the synthesis task, many high-level synthesis systems start with some assumptions (such as an upper bound on the number of functional units available), and perform the synthesis sub-tasks sequentially. Many systems typically start with scheduling, followed by allocation, in that order. However, performing these subtasks independently may miss the optimal design altogether, and the best strategy is to perform these synthesis subtasks concurrently [12, 144].

As modern VLSI designs become more complex, a major problem is the extremely large number of possible schedule and allocation combinations that must be examined in order to select a design that meets constraints and is optimal. This process, called design space exploration, is further compounded by the need for shortening design times due to time-to-market pressures. Since an exhaustive search could be prohibitive and an ad hoc design exploration could be inefficient, designers often select a conservative architecture after some experimentation, which often results in a sub-optimal design. Given this scenario, there is an acute need for techniques that automate the efficient exploration the large space in a reasonable time, during high-level synthesis of datapaths.

In this chapter, we present a genetic algorithm based technique for performing the combined subtasks of scheduling and allocation in high-level behavioral synthesis. Our approach is motivated by the fact that the search space for high-level synthesis is large and discrete, and genetic algorithms are known to work well on such problems [167, 168, 174]. In addition, the inherent parallelism of genetic algorithms allows efficient design space exploration during a
single optimization iteration. Our approach uses a genetic algorithm to concurrently search the space of datapath schedules and module/storage allocations. It combines an indirect chromosome representation with efficient heuristics to derive a solution from the chromosome encoding. A multi-chromosome representation is used to encode a topologically ordered list of scheduled tasks, and a resource list containing the number of functional units allocated. An efficient list scheduling heuristic [144] is used to derive a schedule from the task list, and a left-edge heuristic [29] is used to determine the storage requirements for the datapath from the derived schedule.

7.2 Related Work

The search space of the combined scheduling/allocation problem in high-level synthesis is highly multi-modal, with many widely separated local minima. Searching the complex solution space of this problem requires a trade-off between two apparently conflicting objectives: exploiting the current best solutions in the search space and efficiently exploring the search space.

Genetic algorithms offer several advantages over other techniques in the area of design space exploration for high-level synthesis. Genetic algorithms are effective in intelligently handling the trade-off between the exploration and exploitation of solutions in the search space [167], which is an essential ingredient for efficient design space exploration in high-level synthesis. In addition, since genetic algorithms are population based search techniques, one advantage of using GA-based synthesis is their ability to produce multiple designs that satisfy user specified design constraints. These multiple designs may vary in circuit speed or chip area, and have the potential to offer greater flexibility in the subsequent stages of the design process.

Several high-level synthesis systems have used genetic algorithms to perform some or all of the synthesis sub-tasks. More recently, genetic algorithms have also been applied to design space exploration at the System-on-a-chip level [177], and at the microcode level of instruction set synthesis.
processors [178]. In all these GA-based approaches, a population of solutions to the synthesis problem is iteratively improved through the application of genetic operators to selected individuals in the population. These systems mainly differ in their chromosome representations and the genetic operators used to search the solution space.

In ADaPaS [164], a genetic algorithm is applied to the scheduling and allocation sub-tasks of datapath synthesis. In their work, they use a genetic algorithm to search for a trade-off between resource allocation and schedule length by re-weighing a cost function. The method is based upon assigning a displacement to each of the operations to be scheduled and construct a topologically ordered schedule, using a modified ASAP algorithm. Resource constraints are used to defer operations when all resources in the current cycle considered for scheduling are occupied. In their chromosome representation, each operation is represented as its absolute displacement from its ASAP time-frame. They use simple crossover and mutation operations, and do not need any specialized genetic sequencing operators. The main disadvantage of their chromosome representation is that displacement of operations in the critical path of a data-flow graph has a significant impact on the quality of schedules generated by the genetic algorithm. Special initialization methods are presented to seed the initial population with schedules within their time constraint, by distributing displacements over critical paths. Though an improvement in the quality of solutions is reported with this seeding technique, the simple crossover operators used in their genetic algorithm do not preserve the displacements of critical path operations during the run of the genetic algorithm, producing sub-optimal schedules.

In PSGA [163], a problem-space genetic algorithm is applied to the design space exploration of datapaths. Their approach is based on the use of a problem-specific chromosome representation where each operation is assigned a priority (called work remaining) based on the length of the longest path from the node to the output. They use the concept of a heuristic/problem pair to map each chromosome to a valid schedule. A problem-specific
heuristic, called the *work remaining heuristic*, is used to decode each chromosome into a schedule and a module allocation. Their chromosome representation enables them to use simple crossover and mutation operators to generate valid solutions. However, with their chromosome representation, the number of unique chromosomes that map to the same solution is large, making the search space large, even for problems of modest size.

In [165], Heijlingers *et al.* propose a genetic algorithm based technique for time-constrained scheduling. The authors use a permutation of operations to represent a chromosome, and use a list scheduler to decode each of the chromosomes into a valid schedule. Since many operations in an input dataflow graph typically have precedence constraints, and hence can be scheduled only after all their predecessor operations are completed, the use of random permutations to represent schedules results in an exponentially large number of permutations mapping to the same schedule. This increases the size of the search space proportionately, thus slowing the GA search.

In GABIND [166], a genetic algorithm is applied to the allocation and binding phases of a high-level synthesis. Their approach uses an unconventional crossover mechanism relying on a force directed datapath binding completion algorithm. A bus-based interconnection scheme, and the use of multi-port memories are two of the key features of their system. Their system however does not handle scheduling of operations, and assumes a scheduled data-flow graph as its input.

Finally, in [156, 174], the authors describe a GA-based high-level synthesis system that uses a binary encoding for the chromosomes, as opposed to problem-specific representations such as in [163, 164, and 166]. A binary encoding is used to store information on the the control step assigned to each operation in the data-flow graph, and the functional module assigned to the operation. An inherent disadvantage of using a binary encoding is that the size of the chromosome increases exponentially with the size of the problem, with a corresponding increase in the size of the search space, leading to large run-times for realistic problem sizes.
In this paper, we propose a genetic algorithm based technique that addresses some of the deficiencies of earlier GA-based approaches to high-level synthesis. Our GA is designed to perform the combined sub-tasks of scheduling and allocation in high-level behavioral synthesis, and concurrently search the space of data path schedules and module/storage allocations. Our approach is motivated by the fact that the search space for high-level synthesis is large and discrete, and genetic algorithms are known to work well on such problems [167]. In addition, the inherent parallelism of genetic algorithms allows efficient design space exploration during a single optimization iteration. Our GA incorporates three features that enable it to efficiently perform design space exploration during high-level synthesis.

Firstly, a multi-chromosome encoding is used to concurrently handle search in the space of schedules and allocations for an input behavioral specification. A novel encoding is used to specify datapath schedules. This encoding, designed to improve the efficiency of the GA search for design space exploration, uses a chromosome representation that encodes the precedence relationships among the tasks in the input behavioral specification with a topological order-based representation to specify schedule priorities. This alleviates the many-to-one relationship that exists between the genotype and phenotype encoding in other GA-based high-level synthesis systems. This chromosome representation enables the GA to prune the search space, increasing search efficiency.

Secondly, our GA uses an efficient list scheduling heuristic to decode chromosomes into valid schedules. The list scheduling heuristic is not limited to using a single fixed priority function as in other list scheduling approaches, but instead uses adaptive task priorities that are dynamically decided by the GA. This enhances the robustness of our GA, enabling the scheduler avoid local optima.

Thirdly, our GA concurrently performs register minimization in the datapath, in addition to functional allocation. A variant of the *left-edge algorithm* [29] is used to determine the register
storage requirements for each schedule. Since the cost of registers, in terms of chip area can be significant, our GA-based framework aims to minimize the number of registers needed in a synthesized datapath.

7.3 Design of the Genetic Algorithm

The proposed high-level synthesis system employs the robust search capabilities of a genetic algorithm to concurrently solve the datapath synthesis sub-tasks of scheduling and allocation of resources (functional units such as ALUs and storage units such as registers), with the aim of finding schedules and module/storage combinations that lead to superior designs, while considering user-specified latency and area constraints. The genetic algorithm uses a multi-chromosome representation to encode datapath schedules and functional unit allocations, and efficient heuristics to minimize register and interconnect costs. The proposed synthesis framework provides the flexibility to perform resource-constrained scheduling, time-constrained scheduling, or a combination of the two.

The various design choices used in our GA are described in this section, with the assumption that the reader has knowledge of the basic ideas and general structure of genetic algorithms.

7.3.1 Overall Structure

Figure 7.1 gives an overview of the proposed framework used for high-level synthesis. The input to the GA includes a description of the data-flow graph (DFG) representation of the behavioral description of the datapath, a set of design constraints, user control parameters, and specifications (area/speed) of the functional and storage modules of a design library. The user constraints could include resource and/or latency constraints. User control parameters could include cost function related parameters such as performance-area trade-off weights, and genetic algorithm related parameters such population size, maximum number of evaluations, and
probabilities of application of genetic operators (crossover and mutation). User control parameters could be optionally preset once and stored in a file that the genetic algorithm would read each time it is run, or it could be tuned by the user for each problem instance.

The overall structure of the GA used in our high-level synthesis framework, is shown in Figure 7.2. The GA used is a steady-state GA based on the classification in [162]. The iterative loop of the algorithm starts with an initial population of individuals created randomly, to ensure a uniform sampling of points in the search space, as a starting point. In our version of the steady-state GA, in each generation, two parent individuals are selected from the population, with a probability proportional to their fitness, and two new individuals (i.e., offspring) are created from
Create initial population and evaluate fitness of population members

Select parent members from population based on their fitness criteria

With high probability perform crossover on the node-priority and resource allocation fields of some of the selected parent members in the population

With low probability perform mutation on the node-priority and resource allocation fields of some of the selected parent members in the population

For all offspring population members do: (1) schedule and bind DFG nodes to resources, (2) allocate registers and bind DFG edges, (3) optimize datapath interconnections, (4) evaluate cost function to determine fitness.

Replace population members with offspring based on fitness criteria

STOP?

NO

YES

Return best solutions found

Figure 7.2 Structure of Genetic Algorithm for High-Level Synthesis
these through the application of genetic operations, which are then immediately added to the population. *Binary tournament selection* [161] was seen to perform well during our preliminary experiments, and was used as the selection operator for the two parent individuals. To ensure sufficient diversity in the population at all times, duplicate individuals (*i.e.*, individuals representing identical solutions) are not allowed in the population. Our experiments indicated that duplicate avoidance results in significant improvements in performance over those that allow duplicates. Hence, before every offspring is added to the population, a *duplicate check* is performed. In a duplicate check operation, the *genome* representing the offspring is compared with that of every member in the GA population. If the genome of the offspring is identical to that of an existing population member, the offspring is discarded, else it is added to the population. An offspring that survives the duplicate check is introduced in the population only if it is better than the current worst member in the population, in which case the offspring replaces the worst member. This makes our GA “elitist” in nature, and ensures that search by the GA monotonically progresses towards optimal regions of the search space. The duplicate avoidance policy adopted in our GA ensures that every population member is unique in the genotype space. Our experiments have shown that doing a duplicate check on a newly created offspring before it is introduced in the population enables the GA to maintain population diversity throughout the GA run, and reach highly-fit areas of the solution space efficiently.

Each population member comprises of two strings; one of them is a topologically sorted permutation of tasks, and the other a list of the number of hardware functional units allocated to implement the schedule. A topologically sorted permutation is used to ensure that all precedence constraints among tasks in the input data-flow graph are maintained in each chromosome representation. Our GA applies genetic operators to both the strings independently, enabling the GA to concurrently search the spaces of schedules and resource allocations. Problem-specific heuristics are used as decoders to process the two strings representing an individual, and
determine the schedule-lengths, and the amount of functional and storage units needed to implement the schedule. The initial population consists of random permutations of tasks that are topologically correct in preserving the precedences in the input behavioral specification, and a random number of allocated functional resources.

### 7.3.2 Solution Encoding

A suitable problem encoding must be chosen to ensure that both scheduling and allocation information could be represented, and suitable genetic operators could be designed to enable the genetic algorithm to reach optimal or near-optimal solutions. Our GA uses a multi-chromosome representation, comprising of two independent sub-strings, to encode each individual in the GA population. One of the sub-strings, called the node priority field is used to encode a topologically sorted permuted list of tasks to be scheduled. The node priority field is used as an indirect representation of the final schedule, which is obtained by applying a list scheduling heuristic to
the node priority field. Using this combination of a permuted list of tasks and a list scheduler heuristic ensures that every chromosome in the population results in a feasible schedule. The other sub-string, called the resource allocation field, consists of a list of integers specifying the maximum number of functional units of each type available for scheduling in every time-step of the schedule. This encoding scheme always gives an active schedule and ensures that for each active schedule, there exists an appropriate corresponding permuted sequence of tasks.

Figure 7.3(b) illustrates an example of the encoding used for an IIR filter dataflow graph shown in Figure 7.3(a). A schedule builder is used to decode the node priority field to derive a valid schedule, subject to the resource constraints specified by the resource allocation field of the chromosome representation. The schedule builder used in our system is a list scheduling heuristic which binds ready operations to available functional units in the order they appear in the node priority field of the chromosome. An operation in the node priority field is said to be ready in a given time step if all its predecessor nodes have completed execution before the time step, and a functional unit which supports the operation is available at the time step. Since every permutation of operations could be decoded into a valid schedule, this representation ensures that the genetic algorithm can reach every part of the solution space, including the optimal solution. The resource allocation field ensures that every possible combination of functional resources can be represented.

The individuals in the initial population contain random permutations of operations that are topologically correct in preserving the precedences in the input behavioral specification, together with random combinations of allocated functional units, subject to an upper bound on the number of functional units. The upper bound can be either user specified or computed from the ASAP schedule [12] of the input algorithmic specification. Since the GA searches for priority lists that maximize the cost function, by traversing a search space induced by permutations of operations, it has the ability to find optimal or near-optimal schedules. It can be similarly argued that by traversing a search
space induced by integers representing all possible combinations of functional units allocations, the genetic algorithm has the ability to find optimal or near-optimal allocations of functional units.

### 7.3.3 Crossover Operators

The two sub-strings of the chromosome (node-priority field and resource-allocation field) have their own crossover operators. In our genetic algorithm, the crossover operators for the two sub-strings are invoked independent of each other (i.e., the crossovers acting on the two sub-strings are not necessarily performed together during each crossover event).

Since the node priority field encodes topologically sorted permutations of tasks with precedence constraints, a crossover operator for this sub-string that preserves precedence relationships has been developed; i.e., the permutation of tasks in the offspring also preserve the precedence constraints imposed by the dataflow graph. This ensures that every offspring created during crossover results in a valid schedule. This crossover operator is similar to a one-point crossover, and can be described as follows.

#### 7.3.3.1 One-Point Topological Crossover

A crossover operator that preserves the topological constraints among the tasks in the node-priority field is described in Figure 7.4. In this description, parent-1 and parent-2 refer to the two parents, while offspring-1 and offspring-2 refer to the two offspring created by the crossover operator. The crossover will be illustrated with reference to the example data-flow graph shown in Figure 7.3.

In applying this operator, the parent chromosomes are divided randomly into two sections. For the first section, the offspring inherits both the positions and the order of the tasks from one parent. For the second section, it inherits the order of the rest of the tasks from the other parent.
Since the precedence relationships are maintained in both parents, it is also maintained in the first section of the offspring. Since the tasks in the second section of the offspring all follow the tasks of the first section, and the order of the tasks inherited from the second parent also maintains all precedence relationships among the tasks specified in the input data-flow graph.

Using the example dataflow graph from Figure 7.3, consider the following

$$parent - 1 = \{ 1 \ 2 \ 5 \ 6 \ 7 \ 4 \ | \ 10 \ 3 \ 8 \ 9 \ 11 \}$$

$$parent - 2 = \{ 4 \ 3 \ 1 \ 2 \ 5 \ 8 \ | \ 6 \ 10 \ 7 \ 9 \ 11 \}$$

then,

$$offspring - 1 = \{ 1 \ 2 \ 5 \ 6 \ 7 \ 4 \ | \ 3 \ 8 \ 10 \ 9 \ 11 \}$$

$$offspring - 2 = \{ 4 \ 3 \ 1 \ 2 \ 5 \ 8 \ | \ 6 \ 7 \ 10 \ 9 \ 11 \}$$
7.3.3.2 Crossover Operation for Resource Allocation Sub-string

The resource-allocation sub-string of the chromosome encodes the number of hardware functional units of each type available for scheduling operations in each time-step. Since the number of allocated functional units of each type is independent of each other, standard one-point crossover can be applied to this sub-string.

Consider for example two parent substrings parent-1 and parent-2, where parent-1 represents a solution containing four adders, two multipliers, and three subtracters, while parent-2 represents a solution containing two adders, three multipliers, and one subtracter.

\[
parent - 1 = \{ 4 \ | \ 2 \ 3 \} \\
parent - 2 = \{ 2 \ | \ 3 \ 1 \}
\]

Applying one-point crossover to these at a randomly selected cut-point shown above creates the two offspring solutions offspring -1 and offspring -2 as shown below.

\[
offspring - 1 = \{ 4 \ 3 \ 1 \} \\
offspring - 2 = \{ 2 \ 2 \ 3 \}
\]

Here, the crossover creates one solution containing four adders, three multipliers, and one subtracter, and another with two adders, two multipliers, and three subtracters.

7.3.4 Mutation Operators

As in the case of crossover operators, the node-priority and resource-allocation sub-strings have their own independent mutation operators, and just as for the crossover, the mutation operators are invoked independent of each other (i.e., the mutation operators acting on the two sub-strings are not necessarily performed together during each mutation event).
7.3.4.1 Mutation Operation for Node Priority Sub-string

The one-point topological crossover operator described above produces valid offspring from valid parent chromosomes. However, since they preserve the topological order of tasks, they preserve many of the subsequences that are common to both parents. This could lead to a loss of diversity in the population, leading to premature convergence. An effective mutation operator is needed to continually reintroduce diversity in the population, and also enable the GA to escape from local minima. In our GA, we used a variation of the shift mutation for permutation representations, which is modified to preserve all the precedence relationships in a string.

Consider a string $p$ containing a topologically sorted permuted list of tasks, and let $i$ be the position of some task $T$ in the string. Since precedence relations specified in the input dataflow graph are preserved in the string, all tasks appearing in positions before $i$ correspond to tasks that either precede task $T$ in the input dataflow graph, or have no precedence constraints with $T$. Likewise, all tasks appearing in positions located after $i$ correspond to tasks that either succeed $T$ in the input dataflow graph, or can be scheduled independent of $T$. Note that each of the tasks in the input dataflow graph have a maximum of two predecessor tasks (since all dataflow graph operations considered have a cardinality of two), but may possibly have many successor tasks. Let $P_{\text{MAX}}$ denote the maximum position of a predecessor task of $T$ in the string $p$, and let $S_{\text{MIN}}$ denote the minimum position of a successor task of $T$ in the string $p$, then the task $T$ can be moved anywhere in string $p$ between positions $P_{\text{MAX}}$ and $S_{\text{MIN}}$ without violating any precedence constraints. The mutation operator, called precedence preserving shift mutation, is designed to preserve the precedence constraints specified in the node priority field of a chromosome while perturbing the priority list to create a new solution. This is done by randomly shifting the position of a task in the node priority field of the chromosome between the task's corresponding $P_{\text{MAX}}$ and $S_{\text{MIN}}$ positions in the node priority field. The precedence preserving shift mutation operation is illustrated in Figure 7.5.
Using the example dataflow graph from Figure 7.3, consider the following

\[ p = \{ 4, 3, 1, 2, 5, 8, 6, 10, 7, 9, 11 \} \]

Let the task \( T \) picked for shifting be 8. For this task, the predecessor task in the DFG is 3 and successor task is 9, and therefore, \( P\text{MAX} = 2 \) and \( S\text{MIN} = 10 \). We can see that task \( T \) can be shifted to any position from 3 to 9, without violating any precedence constraints in the dataflow graph. If for example, \( T \) is randomly shifted to position 3, the resulting string becomes

\[ p = \{ 4, 3, 8, 1, 2, 5, 6, 10, 7, 9, 11 \} \]

### 7.3.4.2 Mutation Operation for Resource Allocation Sub-string

This mutation operator provides a means to perturb the number of functional units allocated in a solution. Perturbing the allocated amounts of many of different resource types in the same mutation operation might result in high disruption of fit schemata present in a substring. To minimize this, only one randomly chosen resource is altered in any mutation event. The mutation operation for the resource allocation field is implemented as shown in Figure 7.6. In the figure, \( p \) represents the resource allocation substring of the chromosome, where for each functional resource of type \( k \), \( p[k] \) represents the number of units of the resource available.
7.3.5 Solution Cost

The objective of our GA is to find solutions to a high-level synthesis problem that minimizes the schedule-length (i.e., latency) and the number of functional and storage units required to implement the schedule, while meeting user specified constraints (in terms of latency and overall area). An indirect representation (topologically sorted permuted list of tasks) is used in the chromosomes for the schedules, to avoid producing infeasible schedules during crossover and mutation. A list scheduling heuristic is used to decode the permuted list of tasks into valid schedules. To search the space of functional module allocations, the resource allocation field of the chromosome is used by the list scheduler while decoding. The amount of registers that need to be allocated can be computed after all the tasks in the input DFG have been scheduled. The Left Edge algorithm [29] is used as a heuristic to determine the storage requirements from a given schedule. Figure 7.7 gives an overview of the fitness evaluation procedure.

**Modified List Scheduling Algorithm:** The modified list scheduling algorithm used in our GA takes as input both the substrings in the chromosome (i.e., node priority field and resource allocation field) and creates a valid schedule of tasks. The operation of the list scheduler is similar to a resource-constrained list scheduler [12]. However, in our GA the functional unit resource constraints are specified by the resource allocation substring of each chromosome, which could be different among the set of chromosomes in a GA population. The basic idea is to

---

**Algorithm Resource Allocation Sub-string Mutation**

Randomly pick a functional unit \( p \) of type \( k \), to change its resource allocation
Randomly choose to increment or decrement resource allocation
if increment then \( p[k] = p[k] + 1 \)
else if decrement and \( p[k] > 1 \) then \( p[k] = p[k] - 1 \)

Figure 7.6 Resource Allocation Sub-string Mutation

---
use the order of tasks in the node priority substring of the chromosome as a priority list for the list scheduler, and use the order of tasks in the substring to provide a selection criterion if operations compete for resources. Since the GA population contains many unique node priority strings, it enables the GA to search the space of priority functions for a list scheduler.

Figure 7.8 illustrates the modified list scheduling algorithm used on a node priority substring $p$ containing $N$ tasks to be scheduled in a given dataflow graph.

Using the example dataflow graph from Figure 7.3, consider the following chromosome with the node priority and resource allocation fields shown below

node priority field = { 4 3 1 2 5 8 6 10 7 9 11 }

resource allocation field = { 2 2 }
**Algorithm**  Modified List Scheduling Algorithm

**INPUT:**  DFG, Chromosome

control_step = 1

while not all tasks scheduled

    for $i = 1$ to $N$ do

        if $p[i]$ is unscheduled and a corresponding functional unit is available then

            assign task $p[i]$ to the functional unit in the current control step

        endif

    endfor

    control_step = control_step + 1

end while

schedule_length = control_step

return schedule_length

---

**Figure 7.8** Modified List-Scheduling Heuristic

**Table 7.1** Example Schedule by Modified List-Scheduling Heuristic

<table>
<thead>
<tr>
<th>Control step</th>
<th>Scheduled tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4, 3</td>
</tr>
<tr>
<td>2</td>
<td>1, 2, 8, 10</td>
</tr>
<tr>
<td>3</td>
<td>5, 6</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>

In this example, the resource allocation field indicates that the datapath has 2 multipliers and 2 adders. For this node priority list, the dataflow graph operations are scheduled as shown in Table 7.1. From the table, it is seen that for this schedule example, the schedule length is 6 controls steps. When the clock cycle time for each control step is 10 nanoseconds, the latency represented by this schedule is 60 nanoseconds.
Left edge algorithm: In this work, as in most common in high-level synthesis, all input data are assumed to be latched in registers. Similarly, all data transfers between control steps (which represent intermediate variables in the input algorithmic description) are stored in registers. Finally, all output data are also saved in registers. The number of registers required in a datapath implementation is determined by the maximum number of concurrent data transfers between any two control steps, which in turn is decided by the operation schedule [12]. Hence, the number of registers needed in a datapath can be determined only after operation scheduling is done.

The birth time of a data transfer is the control step in a schedule when it is created by a producer. Likewise, the end time of a data transfer is the control step when it is consumed by the last consumer. The left-edge algorithm can be described as shown in Figure 7.9, where $N$ is the total number of data transfers (including input and output) in the scheduled DFG and $k$ represents the number of registers allocated.
**Objective function:** The objective function used in our system is divided into two sub-functions. Each sub-function evaluates the success of the solution in the minimization of one of the two objectives of the high-level synthesis problem. The individual fitness scores for each evaluation function are combined in a weighted function that reflects the required objectives of the optimization process. The two sub-functions used in our system correspond to the schedule length, and the respective area contributions of the functional units and registers. The objective function can be written as

\[
COST = W_1 \cdot \frac{L_s}{L_{MAX}} + W_2 \cdot \frac{\sum A_i}{A_{MAX}}
\]

and

\[
\sum A_i = \{ A_{FUs} + A_{Regs} \}
\]

where,

- \(COST\) is the cost function of the solution represented by a chromosome,
- \(W_1, W_2\) represent the weights assigned to the contributions of latency and resource constraints, respectively,
- \(L_s\) is the schedule length of the solution represented by a chromosome,
- \(L_{MAX}\) is the longest schedule among all chromosomes in the current generation,
- \(A_{MAX}\) is the maximum area of datapath among all chromosomes in the current generation,
- \(A_{FUs}\) is the sum of the areas of FUs in the datapath represented by a chromosome,
- \(A_{Regs}\) is the sum of the areas of registers in the datapath represented by a chromosome.

Normalizing \(L_s\) and \(\sum A_i\) as shown in the Cost equation keeps their values within the range \([0...1]\).

The GA allows a user to specify lower and upper bounds on the resource and time constraints that a datapath design must meet. Our system uses a *graded penalty function* to reduce the search
space and to quickly guide the genetic algorithm to those areas of the search space that are most likely to contain solutions meeting the user specified design constraints. This graded penalty function reduces the fitness of a solution in proportion to the amount by which a solution fails to meet a constraint. This feature is particularly useful in design space exploration. With the graded penalty function, the new cost function can be written as

\[
COST_{gp} = W_1 \cdot \frac{|L_s - Time\ Constraint|}{L_{MAX}} + W_2 \cdot \frac{|\sum A_i - Area\ Constraint|}{A_{MAX}}
\]

where \(COST_{gp}\) represents the graded penalty cost and the meanings of other quantities remain the same as before. The weights \(W_1\) and \(W_2\) are user-specified and reflect user preferences about the trade-off between area and latency in the desired solution.

**Population size and stopping criteria:** Much experimentation was carried out to determine an appropriate population size and stopping criteria. Our study showed that for the high-level synthesis problem, a constant population size and a constant number of generations give good average results. However, this takes no account of the size of the problem instance. Making both the population size and the number of generations proportional to the problem size leads to better quality results, but means either poor results for the smaller problem instances, if the constants of proportionality are made small, or inordinately large processing times for the larger ones otherwise. A good compromise proved to be setting the population size proportional to the size of the dataflow graph and stopping after a maximum of 100 generations. These choices lead, on average, to the best results and the highest degree of robustness.
7.4 Experimental Results

The proposed GA-based high-level synthesis system was implemented and tested on a SUN UltraSPARC 2 workstation running under SunOS. To perform a qualitative assessment of our GA, it was tested on a number of DSP benchmarks drawn from high-level synthesis literature. We used two metrics to compare the quality of solutions found by our GA with those of other high-level synthesis systems: (i) the latency of a synthesized datapath, which represents the time required by a synthesized datapath to complete all operations specified in an input dataflow graph, and (ii) the cost in terms of chip area needed to realize the synthesized datapath. We compared our results with those obtained by several non-heuristic techniques (ASAP [12], ALAP [12], FDS [30], and SAM [176]), as well as several GA and non-GA based heuristic techniques such as PSGA [163], Ly [155], Torbey [174], Heijligers [165], SALSA [153, 154], and OASIC [151]. This section presents results from the design-space exploration of these benchmark examples using our GA-based high-level synthesis technique, and a comparison of the solutions with those of other techniques.

For all the benchmarks tested, the synthesized designs were assumed to operate with a clock period of 20ns. We used a 0.35um CMOS module library, where ALUs, multipliers, registers and multiplexers are implemented as hard macro-cells (cells having fixed aspect ratio and pin locations). The ALUs have a propagation delay of 6.5ns, and multipliers have a propagation delay of 15ns. We assume that the area cost and delay of a pipelined multiplier are the same as those of a non-pipelined multiplier, respectively.

In all the experiments, the size of the GA population was set to 100, the crossover probability was 0.90, and the mutation probability set to 0.20. Each of the GA runs was stopped after 10,000 fitness evaluations. Since GAs are stochastic algorithms, ten independent runs with different random number seeds were performed for each of the benchmark problem instances, and the best solution found by the GA in each of the 10 runs were recorded.
For each of the benchmark problems, a design-space exploration was performed by setting different values to the weights $W_1$ and $W_2$ corresponding to the delay and area terms of the fitness function. Assigning different values to these weights allows the fitness function to assign different priorities to the area and delay values of a design, enabling an exploration of the space of possible designs that satisfy specified design constraints. In these experiments, we assumed that $W_1 + W_2 = 1.0$, and the value of $W_1$ was changed from 0.0 to 1.0, in increments of 0.05. The best solutions found by the GA (with 10 independent runs for each benchmark), for different settings of $W_1$ and $W_2$ are presented in Table 7.2. Solutions with different settings for $W_1$ and $W_2$ differ in the number of ALUs, multipliers, and storage registers used, and the number of clock cycles (i.e. latency) required by the datapath to complete all computations with these allocated functional and storage units. In all our experiments, the CPU times required by the proposed GA for finding these solutions ranged from under 10 seconds (for the smaller benchmark examples), to under 2 minutes for the larger benchmarks.

Our GA-based synthesis system was tested on seven benchmark examples drawn from literature [30, 31, 146, 151, 153, 155]. These benchmarks represent varying problem sizes and solution complexities, and provide a representative sample of test problems used to study the effectiveness of our high-level synthesis system. In these experiments, the synthesized designs are assumed to work with a clock period of 20ns, and each of the functional units (ALUs/Multipliers) in the synthesized designs take less than 1 clock cycle to complete an operation. Table 7.2 summarizes the synthesis results for these benchmarks, for a number of circuit latencies and chip areas, representing the best solutions found by the GA for different settings of $W_1$ and $W_2$. To assess the effectiveness of our GA-based technique in its ability to converge to near optimal solutions, ten independent runs of the GA was performed with different random seeds, and in each of these runs the best solution found was recorded. Table 7.2
Table 7.2 GA Results for DSP Benchmarks (Note: ALU Delay = 6.5ns, Multiplier Delay = 15ns, Clock Period = 20ns)

<table>
<thead>
<tr>
<th>#</th>
<th>HLS Benchmark</th>
<th>Latency Constraint</th>
<th>Chip Area (mm²)</th>
<th>(Mean – Best) % Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Overall best solution in 10 runs</td>
<td>Mean of best solutions over 10 runs</td>
</tr>
<tr>
<td>1</td>
<td>IIR</td>
<td>80ns</td>
<td>6.741</td>
<td>6.741</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100ns</td>
<td>5.012</td>
<td>5.012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140ns</td>
<td>3.465</td>
<td>3.465</td>
</tr>
<tr>
<td>2</td>
<td>FIR</td>
<td>100ns</td>
<td>5.012</td>
<td>5.012</td>
</tr>
<tr>
<td></td>
<td></td>
<td>120ns</td>
<td>3.465</td>
<td>3.465</td>
</tr>
<tr>
<td>3</td>
<td>FFT</td>
<td>80ns</td>
<td>5.038</td>
<td>5.038</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100ns</td>
<td>2.944</td>
<td>2.944</td>
</tr>
<tr>
<td>4</td>
<td>EWF</td>
<td>280ns</td>
<td>5.724</td>
<td>5.724</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300ns</td>
<td>4.004</td>
<td>4.004</td>
</tr>
<tr>
<td></td>
<td></td>
<td>320ns</td>
<td>3.822</td>
<td>3.822</td>
</tr>
<tr>
<td></td>
<td></td>
<td>540ns</td>
<td>3.639</td>
<td>3.639</td>
</tr>
<tr>
<td>5</td>
<td>ARF</td>
<td>160ns</td>
<td>11.066</td>
<td>11.066</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200ns</td>
<td>7.973</td>
<td>7.973</td>
</tr>
<tr>
<td></td>
<td></td>
<td>360ns</td>
<td>6.244</td>
<td>6.244</td>
</tr>
<tr>
<td>6</td>
<td>DCT</td>
<td>120ns</td>
<td>11.813</td>
<td>11.952</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140ns</td>
<td>11.439</td>
<td>11.446</td>
</tr>
<tr>
<td></td>
<td></td>
<td>160ns</td>
<td>9.884</td>
<td>9.887</td>
</tr>
<tr>
<td></td>
<td></td>
<td>180ns</td>
<td>9.537</td>
<td>9.537</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200ns</td>
<td>7.990</td>
<td>8.094</td>
</tr>
<tr>
<td></td>
<td></td>
<td>220ns</td>
<td>7.981</td>
<td>8.054</td>
</tr>
<tr>
<td></td>
<td></td>
<td>320ns</td>
<td>7.625</td>
<td>7.799</td>
</tr>
<tr>
<td></td>
<td></td>
<td>360ns</td>
<td>6.426</td>
<td>6.426</td>
</tr>
<tr>
<td>7</td>
<td>FDCT</td>
<td>120ns</td>
<td>17.625</td>
<td>17.625</td>
</tr>
<tr>
<td></td>
<td></td>
<td>140ns</td>
<td>11.439</td>
<td>11.439</td>
</tr>
<tr>
<td></td>
<td></td>
<td>160ns</td>
<td>9.537</td>
<td>9.822</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200ns</td>
<td>7.990</td>
<td>8.204</td>
</tr>
<tr>
<td></td>
<td></td>
<td>220ns</td>
<td>7.981</td>
<td>7.981</td>
</tr>
</tbody>
</table>
shows the overall area of the best solution found by our GA-based synthesis system in these ten runs, for various performance (latency) constraints. The average area of the best solutions found by the GA over these ten runs is also shown in the table. Ideally, an effective GA-based search strategy should converge to the optimal or near-optimal solution in every GA run. Computing the average of the best solutions found over the ten independent runs provides an indication of the robustness of a GA-based search strategy in its ability to converge to the high-fitness areas of the search space. In five of the seven benchmarks tested, the GA found the optimal solution in all ten runs. For the remaining two largest benchmarks, the difference in the fitness among the best solutions found by our GA over the 10 runs is less than 3%. This illustrates that our GA is able to consistently converge to high-quality solutions in every GA run, for all the benchmarks tested.

Assigning suitable values to $W_1$ and $W_2$ allows the GA to explore different areas of the area-delay trade-off curves and find solutions that meet a variety of design constraints. Thus, for example, setting $W_1$ to a high value such as 0.90 causes the GA to assign a higher fitness measure to solutions with smaller delays (and a correspondingly larger overall area). Conversely, a low value for $W_1$ (such as 0.10 for example) results in a higher fitness measure to be assigned to solutions that have smaller overall area (and slower speeds). These results clearly demonstrate that faster implementations of a design have larger areas since they require more functional/storage modules, and that minimizing overall area results in designs that are slower.

Table 7.2 illustrates the typical trend observed in the area-delay trade-offs of most VLSI designs. High throughput datapath designs normally require more functional/storage modules, and consequently result in circuits with larger areas. In contrast, designs resulting in smaller circuit areas are usually have larger latencies. While this trend is broadly consistent for all the benchmarks, the actual shape of the area-delay curves are quite different for different examples.
In some benchmarks, a small reduction in the area could result in substantial reduction in the circuit delay, whereas in others this is much smaller. This illustrates that an effective design-space exploration could payoff significantly in terms of final implementation cost.

To further assess the performance of our GA-based high-level synthesis system, we compared our results with those obtained from four different scheduling techniques commonly used in high-level synthesis, namely, ASAP scheduling [16], ALAP scheduling [16], Force-directed (FDS) scheduling [30], and Simultaneous Scheduling-Allocation-Mapping (SAM) [176] heuristic. These scheduling algorithms were tested in a traditional high-level synthesis framework that performs the three synthesis subtasks of scheduling, allocation, and binding independently. The goal of this comparison was two-fold: (i) to verify the performance gains from concurrently performing scheduling and allocation, over a traditional synthesis flow that carries out these subtasks independently, and (ii) to use the performance of these scheduling techniques as a baseline to compare our results. We used the same benchmarks for comparing the results. Table 7.3 tabulates of the results from our experiments comparing the chip areas of the datapaths found by our GA with those synthesized using the other four scheduling algorithms. From the table, it can be seen that our GA finds better solutions than those of the other four scheduling techniques, for all the benchmarks tested.
Table 7.4 Comparison of Proposed GA-based Method with Other Heuristic HLS Techniques (Note: *m = Multi-Cycled Multipliers Used, Clock Period = 20ns)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EWF (*m)</td>
<td>360ns</td>
<td>5.021</td>
<td>5.377</td>
<td>5.194</td>
<td>5.194</td>
<td>5.194</td>
<td>5.377</td>
<td>5.021</td>
</tr>
<tr>
<td>DCT (*m)</td>
<td>360ns</td>
<td>6.063</td>
<td>7.262</td>
<td>n.a</td>
<td>n.a</td>
<td>n.a</td>
<td>6.419</td>
<td>n.a</td>
</tr>
<tr>
<td></td>
<td>380ns</td>
<td>4.169</td>
<td>n.a</td>
<td>n.a</td>
<td>n.a</td>
<td>n.a</td>
<td>4.525</td>
<td>n.a</td>
</tr>
</tbody>
</table>

Table 7.4 compares our GA-based technique with those reported by PSGA [163], Ly et al. [155], Torbey et al. [174], Heijlingers et al. [165], SALSA [153, 154], and OASIC [151]. Of these [163], [165], and [174] are techniques that apply GA-based methods to the high-level synthesis problem. [153] and [154] use simulated annealing based approach for high-level synthesis, while [155] uses a simulated evolution based approach for high-level synthesis. Finally, [151] addresses the high-level synthesis problem using an Integer Linear Programming (ILP) based method. In all these approaches, the synthesized datapaths assume that ALUs take one clock cycle to complete an operation, while multipliers take two clock cycles to complete an operation. In all the experiments reported in Table 7.4, a clock period of 10ns is assumed, so that with the ALUs and multipliers chosen from our module library, all ALU operations can be done in one clock cycle, and all multiplications can be done in 2 clock cycles. As can be seen from Table 7.4, for the instances tested, our technique finds solutions that are better than those found by the other techniques, for the specified design latencies for these benchmarks.

7.5 Analysis and Discussion

The search space of the high-level synthesis problem is large, complex, and highly multimodal, even for reasonably sized problems. There exists a many-to-one mapping between the high-level synthesis subtasks of operation scheduling and resource allocation. For example, every
module allocation maps to many possible operation schedules. Similarly, for every set of allocated modules, there exists a many-to-one mapping between the scheduled operations and the modules to which they can be bound. This is further compounded by the interdependence that exists between these two subtasks of high-level synthesis.

Due to the nature of its search space, greedy search heuristics do not perform well in high-level synthesis. Scheduling algorithms such as ASAP and ALAP often result in sub-optimal solutions, as seen from the synthesis results in Table 7.2. Other heuristics such as FDS and SAM, that schedule one operation at a time, have a tendency to be trapped in local optima. Techniques that use a more global approach, such as simulated annealing and genetic algorithms, tend to perform better than FDS and SAM, on the high-level synthesis problem.

The interdependence of the high-level synthesis subtasks makes it imperative to perform operation scheduling and resource allocation simultaneously, to enable finding high quality solutions. Several high-level synthesis systems such as SAM [176], OASIC [151], SALSA [153], Ly [155], PSGA [163], and Torbey et al. [174] perform the synthesis subtasks simultaneously. Some of the earlier work on high-level synthesis [30, 147, 148, 150] performed these subtasks independently, which may result in sub-optimal solutions.

SAM schedules and allocates dataflow graph operation one at a time. Due to this lack of a global approach to scheduling and allocation, the algorithm can be trapped by local optima. On the benchmarks tested in ours experiments, other approaches found better solutions than SAM.

OASIC is an ILP-based approach to high-level synthesis. It finds optimal solutions, but its worst case run times are exponential in the size of the problem. Hence, this technique is not suited to problems of practical size.

Our technique was compared with three other GA-based high-level synthesis approaches, PSGA [163], that of Torbey et al. [174], and that of Heijligers et al. [165]. The quality of solutions found by our technique was better than the other three due to a better chromosome
representation and GA search operators. In PSGA, the chromosomes use a problem-specific representation where each of the dataflow graph operations is assigned a priority (called work remaining) based on the length of the longest path from the dataflow graph node to an output. Integers are used to encode these priorities. The weakness of this chromosome representation is that in the worst case, an exponentially large number of genotype representations map to the same solution. This increases the size of the search space, slowing down the GA search. The GA-based high-level synthesis technique proposed in [165] also has a similar weakness. In their work, the authors represent chromosomes as random permutations of dataflow graph operations, and use standard permutation based crossover operators to search for good schedules. However, when there are dependencies among operations in a dataflow graph, many permutations map to the same schedule, leading to slower GA search. The technique of [174] uses a binary encoding for the chromosomes. However, with this encoding the size of the chromosome increases exponentially with the size of the problem, with a corresponding increase in the size of the search space.

To reduce the size of the search space, we use a permutation representation for the dataflow graph operations, where the operations in the chromosome are topologically sorted based on the operator dependencies present in the input dataflow graph. All the chromosomes in the population maintain this property at all times, and the GA search operators are designed to preserve operator precedences when creating new individuals. This aims to alleviate the many-to-one relationship that exists between the genotype and phenotype encoding when using a naive permutation representation. Our experiments confirm that this representation leads to fast convergence of the GA to good solutions. In addition, to maintain population diversity throughout the search process, and avoid premature convergence, we do not allow duplicate chromosomes in the population. Results from the benchmarks tested indicate this to be an effective strategy for improving GA performance.
7.6 Summary and Conclusions

Efficient design space exploration during high-level synthesis is critical to the design of cost-effective VLSI circuits. In this chapter, we have presented a genetic algorithm based approach for combined scheduling and allocation during high-level synthesis of datapaths during design space exploration. Our technique offers several advantages over traditional high-level synthesis methods.

Our approach integrates the interdependent subtasks of scheduling and allocation, as opposed to treating these subtasks independently, as is done in many traditional high-level synthesis systems. We have shown that concurrently performing scheduling and allocation is effective in obtaining high-performance datapath architectures as compared with conventional approaches which perform these subtasks sequentially.

Our approach combines the global search technique of genetic algorithms with fast local search heuristics for scheduling and register allocation. This combination allows our synthesis system to search a large, highly complex, and multi-modal design space, in an efficient manner in order to find the best possible solution within reasonable CPU times. Our GA incorporates three key features that enable it to efficiently perform design space exploration in high-level synthesis.

First, a multi-chromosome encoding is used to concurrently handle search in the space of schedules and allocations for an input behavioral specification. A novel encoding is used to specify datapath schedules. This encoding, designed to improve the efficiency of the GA search for design space exploration, uses a chromosome representation that encodes the precedence relationships among the tasks in the input behavioral specification with a topological order-based representation to specify schedule priorities. This alleviates the many-to-one relationship that exists between the genotype and phenotype encoding in other GA-based high-level synthesis systems. This chromosome representation enables the GA to prune the search space, increasing
search efficiency, leading to significantly better solutions than those obtained by other GA-based high-level synthesis systems.

Second, our GA uses an efficient list scheduling heuristic to decode chromosomes into valid schedules. The list scheduling heuristic is not limited to using a single fixed priority function as in other list scheduling approaches, but instead uses adaptive task priorities that are dynamically decided by the GA. This enhances the robustness of our GA, enabling the scheduler to avoid local optima.

Third, our GA concurrently performs register minimization in the datapath, in addition to functional allocation. A variant of the left-edge algorithm is used to determine the register storage requirements for each schedule. Since the cost of registers, in terms of chip area, can be significant, our GA-based framework minimizes the number of registers needed in a synthesized datapath.
CHAPTER 8

CONCLUSIONS AND FUTURE WORK

The continuing scaling of CMOS technology poses several design challenges for future high-performance architectures. Some of these challenges include poor scaling of interconnect delays, increased power consumption, rising thermal concerns, and manufacturing challenges. The work presented in this dissertation aims to address some of these challenges during behavioral synthesis through the use of unified physical-level and high-level synthesis techniques. In particular, we propose novel synthesis methods to mitigate the growing impact of interconnect delays and on-chip thermal gradients by accounting for them during high-level synthesis.

With submicron device dimensions and millions of transistors integrated on a chip, interconnects have begun to dominate performance, power, and size of integrated circuits. Three-dimensional (3-D) integrated circuit technology has the potential to alleviate many of the performance and power related issues raised by interconnects in conventional planar (2-D) integrated circuits. 3-D integration provides increased device density, reduced latency, and lower power. The relative benefits of 3-D integration technology will increase in future technology generations, making it a very attractive option for future designs. Physical synthesis for 3-D integrated circuits is substantially different from traditional planar 2-D integrated circuits due the presence of additional constraints posed by the need to place circuit blocks in multiple silicon die. To realize the full potential offered by 3-D integrated circuits, high-level synthesis of these circuits must take layout-related issues unique to 3-D technology into account during synthesis. In Chapter 3, we proposed a 3-D layout aware binding algorithm for high-level synthesis, that
tightly integrates the synthesis tasks of resource binding, assignment of datapath functional units to multiple silicon layers, 3-D floorplanning, and through-silicon via minimization. Since floorplanning and resource binding are inter-dependent, we demonstrate that the proposed algorithm significantly outperforms a conventional synthesis flow that separates the binding and floorplanning steps. Compared to a 3-D layout-unaware approach, our experiments show an improvement in the total wirelength of 29% on average, while the longest wirelength is reduced by 21%. In addition, the number of through-silicon vias are reduced by 27%. These optimizations were achieved with no penalty in chip area.

In Chapter 4, we proposed techniques to minimize interconnect delays during high-level synthesis through the use of a net-topology aware binding algorithm. The algorithm uses accurate net topologies and distributed wire-delay models to guide resource allocation and binding decisions during design-space exploration. The proposed approach tightly integrates a floorplanner with a high-level synthesis binding algorithm. The location of data path modules in the floorplan is used to determine the minimal length Rectilinear Steiner Minimum Tree (RSMT) of every net, to which the delay model is applied to accurately estimate delays of multi-terminal nets. Our results show that, when compared to previous approaches, our method reduces wire delays by as much as 48.9% in 70nm technology, with an average improvement of 38.6%, and an overhead of only 3.6% in chip area.

With increasing device counts and operating frequencies, total power and power density is becoming a serious problem in high-performance VLSI circuits. In these circuits, power can be unevenly distributed, leading to thermal-hotspots with significantly greater temperatures than surrounding regions. Elevated chip temperatures have an adverse impact on performance, reliability, power consumption, and cooling costs. Thermal-hotspots lead to serious reliability issues such as thermal-runaways. To ensure adequate thermal management, all phases of the design flow must account for thermal effects on their design decisions. In Chapter 5, we showed
that on-chip thermal distributions are strongly influenced by high-level synthesis as well floorplanning. We proposed an integrated approach to power and thermal management during high-level synthesis, through the use of a two-stage simulated annealing-based synthesis technique that combines power minimization with temperature-aware scheduling, binding, and floorplanning. We showed through extensive experiments that minimizing average power alone does not guarantee minimal peak temperatures. However, our approach consistently found solutions that have lower on-chip peak temperatures and uniform on-chip temperature distributions, compared to a traditional low-power synthesis methodology that minimizes average power. Our method reduces peak temperatures by an average of 15% and up to 23%, compared to traditional low-power synthesis that minimizes average power. These improvements in chip-level temperature distributions are achieved with a modest increase in chip area of under 9% on average.

In Chapter 6, we showed that stochastic wirelength estimation is a viable technique for evaluating the interconnect complexity of designs explored during high-level synthesis. Stochastic wirelength estimation uses analytical models to estimate wirelength distributions of logic gate netlists, based on a well-established empirical law called Rent's rule. We propose an efficient technique to dynamically extract the Rent parameters of a gate-level netlist, and use it to estimate the wiring complexity of the datapath netlists examined during design space exploration. We also develop an iterative HLS design-space exploration engine that uses this information to guide module and register binding decisions during high-level synthesis, with goal of synthesizing a design with the minimal achievable clock period. To the best of our knowledge, this is the first work to apply stochastic wirelength estimation to interconnect-aware high-level synthesis. The key advantage to using our approach is that it can be applied to non-hierarchical (i.e. flattened) gate-level layouts of standard-cells or gate arrays. In contrast, most
existing approaches to interconnect-aware HLS derive their wirelength estimates from a floorplan of RTL macro-cells, making their approach only applicable to macro-cell based layouts.

In Chapter 7, we presented a Genetic Algorithm-based approach for combined scheduling, allocation, and binding during high-level synthesis of datapaths during design space exploration [54]. The Genetic Algorithm uses a multi-chromosome representation to encode datapath schedules and module allocations and efficient heuristics to minimize functional and storage area costs, while minimizing circuit latencies. Our approach combines the global search capability of Genetic Algorithms with fast local search heuristics for scheduling and register allocation. This combination allows our synthesis system to search a large, multi-modal design space, in an efficient manner, in order to find the best possible solution within reasonable CPU times. Since Genetic Algorithms are population-based search heuristics, a unique feature of our framework is its ability to offer a large number of alternative datapath designs, all of which meet design specifications but differ in module, register, and interconnect configurations.

The publications resulting from this dissertation are given in the references [39, 42, 43, 44, 50, 51, 52, 54]. The work presented in this dissertation could be extended in several directions.

The higher power densities in 3-D integrated circuits makes thermal management a critical issue. The temperature-aware synthesis methods of Chapter 5 could be incorporated in the 3-D high-level synthesis algorithm proposed in Chapter 3.

The net-topology aware binding algorithm presented in Chapter 4 could be extended to incorporate interconnect planning and estimation techniques for wire-sizing, wire-spacing, and buffer planning using the interconnect estimation models of Cong and Pan in [124].

The temperature-aware high-level synthesis technique presented in Chapter 5 currently estimates on-chip thermal distributions based on the switching power dissipation of the datapath functional units. Future work could extend the current technique to also account for the effects of leakage power when estimating the temperatures of datapath functional units.
The stochastic wirelength estimation based clock period optimization algorithm presented in Chapter 6 could be extended to minimize interconnect power consumption.

A natural extension of the Genetic Algorithm proposed in Chapter 7 is to integrate it into an evolvable hardware-based HLS framework. In this framework, the evolved circuit is implemented in an FPGA hardware. The population of solutions is maintained outside the circuit, and the evolutionary mechanisms of selection, crossover, mutation, and fitness computation, are executed outside the resulting circuit, as well. However, the fitness of the solutions are evaluated by programming them in the FPGA hardware, where the performance characteristics of the implemented circuit are determined in terms of measurable metrics such as silicon area, operation speed, and dissipated power. The datapath circuits are synthesized off-line using higher level functions (adders, multipliers, multiplexers, registers). Use of higher-level functional modules reduces the search space by orders of magnitude, when compared to using a gate-level (i.e., logic cell-level) representation. In addition, it has the benefit of reducing the size of the genome used to represent solutions. During fitness evaluation, each chromosome encoding in the GA population is decoded into a circuit netlist comprising of an interconnection of such higher-level functional modules. Since these functional modules are drawn from a pre-designed module library, the decoded circuit is mapped to a netlist of logic cells implementing the desired circuit, and loaded onto the FPGA circuit board for evaluation.
REFERENCES


[90] Predictive Technology Model (PTM), [online]: http://www.eas.asu.edu/~ptm


ABOUT THE AUTHOR

Vyas Krishnan received the Bachelor of Engineering degree in Electrical Engineering from the National Institute of Technology – Surathkal, India. He received the Master of Science degree in Computer Science from the University of South Florida, and a Master of Science degree in Physics, also from the University of South Florida. He has taught several courses as an instructor, both, at the Department of Computer Science and Engineering, and at the Department of Physics, in the University of South Florida. He has published several research papers in peer-reviewed journals and international conferences, in the areas of VLSI design automation and semiconductor physics. His research interests are in the areas of semiconductor device physics, high-level synthesis for low-power high-performance designs, physical synthesis for nanometer-CMOS technologies, VLSI design automation, computer architecture, and multimedia processors.