Technology Mapping and Packing for Coarse-grained, Anti-fuse Based FPGAs

Chang Woo Kang, Ali Iranli, and Massoud Pedram

University of Southern California
Department of Electrical Engineering
Los Angeles CA, USA

Outline

- Introduction
- Cell Library Construction
- Technology Mapping and Cell Packing
- Experimental Results
- Conclusion
Introduction

- Coarse-grained logic block architecture is widely used in the FPGA industry
  - Xilinx Virtex series has 8 LUTs in a configurable logic block
  - QuickLogic pASIC series has 26 inputs in a logic cell

Example of Fine vs. Coarse-grained Antifuse-based FPGAs

Fine Grain: Actel ACT 2 Logic Module
Coarse Grain: QuickLogic pASIC Logic Cell
pASIC Logic Cell

- Large fanin
  - 26 inputs, 4 outputs
- Components
  - Two 6-input AND gates
  - Four 2-input AND gates
  - Six 2:1 MUXs
- High logic capacity
  - More than 5000 library cells could be generated
- A single level of logic delay can realize many complex user functions
- Up to four logic functions can be realized with the same logic cell

Problem Formulation

- Minimize the number of required pASIC logic cells needed to implement a given target circuit
Cell Library Construction

Split the logic cell

Cell personalization

14 primitive cells

Base-gate A

30 primitive cells

Base-gate B

Select 196 useful
primitive cells

Base-gate C

Cell Type Determination

- Note that the same primitive cell may be realized by more than one base gate
- A total of 205 unique primitive cells have been generated and inserted into a cell library
- A total of seven unique cell types have been defined as shown in the Venn’s diagram
Cost Assignment for Primitive Cells

\[
\text{cell \_ cost} = \frac{s}{f \cdot c}
\]

- Space usage, \( s \): the amount of space in a pASIC cell that is used up by the primitive cell
- Freedom, \( f \): the total number of places in a pASIC cell that the primitive cell can fit
- Coverage, \( c \): complexity of the logic that the primitive cell can realize
- Notice that the inverter primitive cell does not have the minimum cost any more

Two-step Optimization Flow

- Perform minimum-area technology mapping by using the generated cell library
  - Use standard technology mapping algorithms (e.g., the SIS mapper)

- Pack primitive cells into pASIC logic cells in order to have the minimum number of logic cells
  - Formulate and solve as a multi-dimensional coin change problem
The Coin-Change Problem

● Problem statement
  – Let $c_1, c_2, \ldots, c_m$ be the coin types of a currency. Let $C_i$ denote the value of coin $c_i$ in cents and $K$ be some integer. We assume $C_1 = 1$. The problem is to produce $K$ cents of change by using a minimum number of coins

$$\text{Minimize } \sum_{i=1}^{m} n_i \quad \text{s.t. } K = \sum_{i=1}^{m} n_i C_i$$

where $n_i$ denotes the number of coins of type $i$

● Solution

$$\text{count}[K] = \begin{cases} 0 & \text{if } K = 0 \\ \min_{i : C_i \leq K} \{ \text{count}[K - C_i] + 1 \} & \text{if } K > 0 \end{cases}$$

Different Ways of Packing Cells

● 37 different cases of completely utilizing a logic cell

- $C_{i,S_j}$: the number of primitive cells of type $S_j$ in the $i_{th}$ combination

- The packer must find optimal packing combinations in a bottom-up manner

<table>
<thead>
<tr>
<th>Combinations of primitive cells</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>35</th>
<th>36</th>
<th>37</th>
</tr>
</thead>
<tbody>
<tr>
<td>2S_1 + 2S_2</td>
<td>2S_4 + 2S_3</td>
<td>2S_5 + 2S_4</td>
<td>...</td>
<td>...</td>
<td>S_1 + 2S_3</td>
<td>S_4 + 2S_3</td>
<td>S_3 + S_4 + S_5</td>
</tr>
</tbody>
</table>
Exact Problem Statement

- Given the different ways of packing a pASIC3 logic cell as described in the previous table and a logic netlist generated by the min-cost technology mapper, find the minimum number of pASIC3 logic cells needed to cover all primitive cells in the logic netlist, i.e.:

\[
\text{Minimize} \quad \sum_{i=1}^{37} n_i \quad \text{s.t.} \quad \forall j: \sum_{i=1}^{37} n_i C_{i,j} \geq |S_j|
\]

where \( n_i \) denotes the number of packings of type \( i \) and \( |S_j| \) denotes the number of primitive cells of type \( j \) in the initial logic netlist.

Analogy to the Coin-Change Problem

- After technology mapping, we count the number of primitive cells of each type, \( |S_1|, \ldots, |S_7| \). These are analogous to seven different target change counts.
- The 37 different entries in the pASIC packing table are analogous to different currencies.
- This is a multi-dimensional coin change problem, which can be solved optimally and efficiently by using a dynamic programming technique.
Dynamic Programming Solution

\[
\text{count}(|S_1|,\ldots,|S_7|) = \begin{cases} 
0 & \text{if } \forall j, |S_j| \leq 0 \\
\min_{i:j \in \mathcal{C}_i} \left( \text{count}(|S_i| - C_{i,S_j}, \ldots, |S_7| - C_{i,S_7}) + 1 \right) & \text{otherwise}
\end{cases}
\]

- Note that we must track remaining unpacked primitive cells for all seven cell types at the same time.
- Computational complexity \( O\left(\prod_{i=1}^{7} |S_i|\right) \)

### Experimental Results

<table>
<thead>
<tr>
<th>Circuits</th>
<th>Primitive cell count</th>
<th>Number of logic cells</th>
<th>Cell utilization (%)</th>
<th>CPU time (sec)</th>
<th>Number of logic cells</th>
<th>Cell utilization (%)</th>
<th>CPU time (sec)</th>
<th>Number of logic cells</th>
<th>Cell utilization (%)</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alu2</td>
<td>193</td>
<td>26</td>
<td>65.87</td>
<td>0.08</td>
<td>51</td>
<td>100</td>
<td>50.27</td>
<td>32.9</td>
<td>34.1</td>
<td></td>
</tr>
<tr>
<td>Alu4</td>
<td>377</td>
<td>150</td>
<td>65.11</td>
<td>0.34</td>
<td>101</td>
<td>98.43</td>
<td>960.59</td>
<td>32.7</td>
<td>31.9</td>
<td></td>
</tr>
<tr>
<td>Apex6</td>
<td>249</td>
<td>134</td>
<td>59.86</td>
<td>0.37</td>
<td>117</td>
<td>80.23</td>
<td>106.24</td>
<td>24</td>
<td>25.4</td>
<td></td>
</tr>
<tr>
<td>Dali</td>
<td>471</td>
<td>194</td>
<td>63.39</td>
<td>0.56</td>
<td>138</td>
<td>90.75</td>
<td>143.81</td>
<td>28.9</td>
<td>30.1</td>
<td></td>
</tr>
<tr>
<td>C1355</td>
<td>210</td>
<td>83</td>
<td>64.81</td>
<td>0.11</td>
<td>58</td>
<td>93.75</td>
<td>1.17</td>
<td>30.1</td>
<td>30.9</td>
<td></td>
</tr>
<tr>
<td>C1908</td>
<td>213</td>
<td>96</td>
<td>55.84</td>
<td>0.13</td>
<td>74</td>
<td>77.74</td>
<td>20.54</td>
<td>22.9</td>
<td>21.3</td>
<td></td>
</tr>
<tr>
<td>C432</td>
<td>108</td>
<td>59</td>
<td>65.85</td>
<td>0.03</td>
<td>51</td>
<td>100</td>
<td>13.36</td>
<td>51.1</td>
<td>24.2</td>
<td></td>
</tr>
<tr>
<td>C499</td>
<td>230</td>
<td>32</td>
<td>64.81</td>
<td>0.11</td>
<td>58</td>
<td>93.75</td>
<td>1.97</td>
<td>30.1</td>
<td>30.9</td>
<td></td>
</tr>
<tr>
<td>C7349</td>
<td>257</td>
<td>268</td>
<td>64.62</td>
<td>0.2</td>
<td>191</td>
<td>91.89</td>
<td>56.21</td>
<td>26.7</td>
<td>26.1</td>
<td></td>
</tr>
<tr>
<td>C1730</td>
<td>214</td>
<td>92</td>
<td>62.76</td>
<td>0.12</td>
<td>64</td>
<td>93.45</td>
<td>65.95</td>
<td>30.4</td>
<td>32.8</td>
<td></td>
</tr>
<tr>
<td>C5155*</td>
<td>764</td>
<td>133</td>
<td>59.97</td>
<td>1.94</td>
<td>256</td>
<td>79.09</td>
<td>42.97</td>
<td>23.1</td>
<td>24.2</td>
<td></td>
</tr>
<tr>
<td>C6288*</td>
<td>1457</td>
<td>664</td>
<td>55.19</td>
<td>13.11</td>
<td>593</td>
<td>61.84</td>
<td>19.14</td>
<td>10.7</td>
<td>10.8</td>
<td></td>
</tr>
<tr>
<td>C7552*</td>
<td>1052</td>
<td>443</td>
<td>64.94</td>
<td>1.7</td>
<td>312</td>
<td>86.31</td>
<td>86.06</td>
<td>24.5</td>
<td>21.9</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*The packing algorithm used segmented lists so as not to exceed the amount of available memory in the computer.*
Conclusions

- Proposed a minimum-area packing algorithm for coarse-grained, anti-fuse based FPGAs, comprising of library generation, technology mapping and cell packing.
- Solution of a multi-dimensional coin-change problem resulted in a polynomial time optimal solution to the cell packing problem.
- Our algorithm resulted in an average of 27% fewer logic cells compared to a greedy algorithm.