A Novel Wirelength-Driven Packing Algorithm for FPGAs with Adaptive Logic Modules

Speaker: Po-Yi Hsu
Sheng-Kai Wu, Po-Yi Hsu, Wai-Kei Mak
Dept. of CS, National Tsing Hua University
Outline

• Introduction
• Preliminaries
• Algorithm
• Experimental results
• Conclusion
Fracturability of ALM

- ALM can serve as a 6-input LUT or two smaller LUTs if the total distinct inputs is less or equal to 8 under certain constraints.

(From http://www.altera.com)
Recent Research of ALM

- [Hutton FPL04] reported that the ALM architecture led to a 15% timing improvement and 12% area reduction on average versus a standard BLE4 architecture.

- How to merge two LUTs into one ALM so that it will not adversely affect the quality for later placement and routing is an important issue.

- In this work, we propose a novel packing algorithm for ALM-based FPGA targeting directly at wirelength minimization.
Outline

• Introduction

• Preliminaries
  • The idea of safe clustering
  • Problem formulation

• Algorithm

• Experimental results

• Conclusion
The idea of safe clustering

• The concept of safe clustering for effectively reducing the problem size in ASIC placement was introduced in [Yan TCAD12].

• Safe clustering guarantees that the clustering will not adversely affect the final total wirelength.
Gradient functions

• Given a hypergraph $G(V, E)$, where $V$ is the set of nodes and $E$ is the set of hyperedges corresponding to the nets.
• $P$: the set of all possible valid placements
• $E_v$: the set of hyperedges incident to $v$

$p \in P, \ e \in E, \ \text{a pair of nodes } \{a, b\} \ \text{with } a \ \text{on the left of } b$
if $a$ is the rightmost vertex of $e$

$$\Delta_a(p, e) = w_e$$

if $a$ is the only leftmost vertex of $e$

$$\Delta_a(p, e) = -w_e$$

otherwise

$$\Delta_a(p, e) = 0$$
Total wirelength gradient function

• From the above gradient function, the total wirelength gradient function is defined as follows:

\[ F_{ab}(p) = \min \left( \sum_{e \in E_a} \Delta_a(p,e), \sum_{e \in E_b} \Delta_b(p,e) \right) \]

• If all possible placements \( F_{ab} \) is not greater than zero. It is safe to cluster \( a \) and \( b \).

\[ \max(F_{ab}(p_1), F_{ab}(p_2), \ldots, F_{ab}(p_i)) \leq 0, \quad p_i \in P \]

• However, it is not practical to generate all possible placements when considering clustering \( a \) and \( b \).
Selective enumeration

• A selective placement enumeration approach is proposed in [Yan TCAD12].

• It only enumerates the placements that might generate worse wirelength if we merge $a$ and $b$. 
All placement \( |P| = \infty \)

For each node \( v \in V_{ab} \)

1. \( v \) is on the left of \( a \)
2. \( v \) is between \( a \) and \( b \)
3. \( v \) is on the right of \( b \)

Only consider \( V_{ab} \) nodes connected with at least one of \( a \) or \( b \)

\( 3|V_{ab}| \)

[Yan TCAD12] has proven that case(2) will never be worse than case(1) or case(3).

Only two possible locations for \( v \)

\( 2|V_{ab}| \)

[Yan TCAD12] identified a subset of nodes in \( V_{ab} \) for which it is unnecessary to consider both possible positions for them.

Final placements that need to be enumerated to check the safeness

\( 2|V_{ab}| - \alpha \)
Problem formulation

• Given:
  • A mapped netlist of 6-input LUTs

• Objective:
  • Merge the LUTs into ALMs under the ALM architecture constraint and cluster ALMs into CLBs so as to optimize the expected wirelength after place and route.
Outline

• Introduction
• Preliminaries
• Algorithm
  - Merge LUTs into ALMs
  - CLB clustering
• Experimental results
• Conclusion
Overall flow

Gate-Level Netlist

Technology mapping

LUT Netlist

ALMPack

Merge LUTs into ALMs

Pack ALMs into CLBs

CLB Netlist

VPR: Placement & Route
Merge LUTs into ALMs

• We model the wirelength-driven ALM formation problem as a *minimum weighted maximum matching* problem.

• We construct a weighted undirected graph where each node corresponding to a LUT, and there is an edge between two nodes if and only if the corresponding LUTs can be merged into an ALM.
Merge LUTs into ALMs flow

1. LUT netlist
2. Identify LUT pairs which can be merged into an ALM
3. Compute edge weight of each LUT pairs
4. call minimum weighted maximum matching solver
5. ALM netlist
Edge weight

• We define the edge weight between two nodes $a$ and $b$ as follows:

$$weight(a, b) = \begin{cases} 
0, & \text{if } 2|V_{ab}| - \alpha_{ab} \leq \text{enum\_bound} \\
\frac{\left(\sum_{i=1 \text{s.t.} F_{ab}(p_i) > 0} F_{ab}(p_i))}{L_{ab}}, & \text{if } 2|V_{ab}| - \alpha_{ab} \leq \text{enum\_bound} \\
B - \beta \frac{|\text{NET}(a) \cap \text{NET}(b)|}{|\text{NET}(a) \cup \text{NET}(b)|}, & \text{if } 2|V_{ab}| - \alpha_{ab} > \text{enum\_bound}
\end{cases}$$
Pack ALMs into CLBs

1. ALM netlist
   - $N = 2^k$
   - $N \neq 2^k$

2. Identify ALM pairs which can be merged into a CLB
3. Compute edge weight of each ALM pairs
4. Call minimum weighted maximum matching solver
5. Pack feasible ALM with smallest edge weight
6. Choose one ALM put into a new CLB
7. Compute edge weight of other unpacking ALM
8. If CLB is not full
   - Pack feasible ALM with smallest edge weight
9. If exists unpacking ALM
   - Pack feasible ALM with smallest edge weight

CLB netlist

$k$ loops
Outline

• Introduction
• Preliminaries
• Algorithm
• Experimental results
  • ALM-based FPGA (compare to AAPack)
  • Traditional BLE4 FPGA (compare to T-VPack)
• Conclusion
Environment Setup

- We implemented our packing algorithm, ALMPack, using C++ on an Ubuntu workstation with 8 GB memory and 2.13 GHz CPU.
- 20 largest MCNC benchmarks for the experiments.

<table>
<thead>
<tr>
<th></th>
<th>Exp. 1</th>
<th>Exp. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic element</td>
<td>ALM</td>
<td>4-LUT</td>
</tr>
<tr>
<td>Cluster size (N)</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td># inputs of CLB(I)</td>
<td>34</td>
<td>22</td>
</tr>
<tr>
<td>Placement Algorithm</td>
<td>Timing-driven</td>
<td>Timing-driven</td>
</tr>
<tr>
<td>Routing Algorithm</td>
<td>Timing-driven</td>
<td>Timing-driven</td>
</tr>
<tr>
<td>Segment length</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Switch type</td>
<td>Buffered</td>
<td>Buffered</td>
</tr>
<tr>
<td>$F_c$ (pad,input,output)</td>
<td>(1,0.5,0.25)</td>
<td>(1,0.5,0.25)</td>
</tr>
<tr>
<td>$F_s$</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

ARCHITECTURE AND EXPERIMENTAL SETTINGS
# ALM-based FPGA

## Benchmarks and Performance Metrics

<table>
<thead>
<tr>
<th>Bench.</th>
<th># LUTs</th>
<th># CLBs</th>
<th>min. channel width</th>
<th>wirelength</th>
<th>delay($10^{-8}$ sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>AAPack</td>
<td>ALMPack</td>
<td>AAPack</td>
</tr>
<tr>
<td>ex5p</td>
<td>745</td>
<td>54</td>
<td>57</td>
<td>58</td>
<td>48</td>
</tr>
<tr>
<td>spla</td>
<td>2072</td>
<td>166</td>
<td>187</td>
<td>106</td>
<td>86</td>
</tr>
<tr>
<td>alu4</td>
<td>803</td>
<td>65</td>
<td>70</td>
<td>62</td>
<td>58</td>
</tr>
<tr>
<td>apex2</td>
<td>1058</td>
<td>88</td>
<td>85</td>
<td>74</td>
<td>72</td>
</tr>
<tr>
<td>apex4</td>
<td>787</td>
<td>65</td>
<td>63</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td>des</td>
<td>555</td>
<td>46</td>
<td>46</td>
<td>40</td>
<td>38</td>
</tr>
<tr>
<td>ex1010</td>
<td>2703</td>
<td>224</td>
<td>221</td>
<td>114</td>
<td>94</td>
</tr>
<tr>
<td>misex3</td>
<td>818</td>
<td>65</td>
<td>66</td>
<td>62</td>
<td>60</td>
</tr>
<tr>
<td>pdc</td>
<td>2417</td>
<td>195</td>
<td>206</td>
<td>116</td>
<td>96</td>
</tr>
<tr>
<td>seq</td>
<td>960</td>
<td>77</td>
<td>76</td>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>bigkey</td>
<td>579</td>
<td>50</td>
<td>50</td>
<td>44</td>
<td>40</td>
</tr>
<tr>
<td>clma</td>
<td>3911</td>
<td>330</td>
<td>328</td>
<td>88</td>
<td>74</td>
</tr>
<tr>
<td>dsip</td>
<td>689</td>
<td>43</td>
<td>43</td>
<td>38</td>
<td>34</td>
</tr>
<tr>
<td>diffeq</td>
<td>660</td>
<td>55</td>
<td>54</td>
<td>44</td>
<td>30</td>
</tr>
<tr>
<td>elliptic</td>
<td>1795</td>
<td>140</td>
<td>138</td>
<td>74</td>
<td>50</td>
</tr>
<tr>
<td>frisc</td>
<td>1797</td>
<td>133</td>
<td>132</td>
<td>88</td>
<td>74</td>
</tr>
<tr>
<td>s298</td>
<td>780</td>
<td>62</td>
<td>62</td>
<td>60</td>
<td>58</td>
</tr>
<tr>
<td>s38417</td>
<td>2781</td>
<td>217</td>
<td>215</td>
<td>54</td>
<td>42</td>
</tr>
<tr>
<td>s38584</td>
<td>2504</td>
<td>195</td>
<td>185</td>
<td>64</td>
<td>44</td>
</tr>
<tr>
<td>tseng</td>
<td>660</td>
<td>51</td>
<td>49</td>
<td>40</td>
<td>28</td>
</tr>
<tr>
<td>Avg. Impv.</td>
<td>-0.47%</td>
<td>14.54%</td>
<td>17.97%</td>
<td>2.14%</td>
<td></td>
</tr>
</tbody>
</table>

**Summary:**

The table above compares the performance of ALM-based and ALMPack FPGA designs across various benchmarks, including the number of LUTs, CLBs, minimum channel width, wirelength, and delay (in $10^{-8}$ seconds). The Avg. Impv. column indicates the average improvement in delay across all benchmarks, showing a range from a 2.14% decrease to a 17.97% increase.
ALM-based FPGA

• It shows that the final wirelength is improved in all cases and is 17.97% shorter on average using ALMpack.
• Reduced the minimum channel width in 18 of the 20 benchmarks and never increased the minimum channel width.
• We ran AAPack in default mode which optimizes both area and timing, but we still obtained 2.14% improvement for delay on average with comparable area.
• Our delay improvement will be more significant if we route the designs under the same channel widths as AAPack.
## Traditional BLE4 FPGA

<table>
<thead>
<tr>
<th>Bench.</th>
<th># LUTs</th>
<th># CLBs</th>
<th>min. channel width</th>
<th>wirelength</th>
<th>delay($10^{-8}$ sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>T-VPack</td>
<td>ALMPack</td>
<td>T-VPack</td>
<td>ALMPack</td>
</tr>
<tr>
<td>ex5p</td>
<td>892</td>
<td>116</td>
<td>112</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>spla</td>
<td>3016</td>
<td>386</td>
<td>377</td>
<td>54</td>
<td>52</td>
</tr>
<tr>
<td>alu4</td>
<td>1205</td>
<td>155</td>
<td>151</td>
<td>42</td>
<td>38</td>
</tr>
<tr>
<td>apex2</td>
<td>1441</td>
<td>187</td>
<td>181</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>apex4</td>
<td>1061</td>
<td>138</td>
<td>135</td>
<td>50</td>
<td>44</td>
</tr>
<tr>
<td>des</td>
<td>1238</td>
<td>156</td>
<td>155</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>ex1010</td>
<td>3854</td>
<td>513</td>
<td>496</td>
<td>72</td>
<td>58</td>
</tr>
<tr>
<td>misex3</td>
<td>1173</td>
<td>151</td>
<td>150</td>
<td>42</td>
<td>36</td>
</tr>
<tr>
<td>pdc</td>
<td>3435</td>
<td>442</td>
<td>430</td>
<td>62</td>
<td>58</td>
</tr>
<tr>
<td>seq</td>
<td>1361</td>
<td>177</td>
<td>172</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>bigkey</td>
<td>1146</td>
<td>144</td>
<td>144</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>clma</td>
<td>5621</td>
<td>706</td>
<td>703</td>
<td>60</td>
<td>50</td>
</tr>
<tr>
<td>dsip</td>
<td>1368</td>
<td>171</td>
<td>171</td>
<td>18</td>
<td>12.00</td>
</tr>
<tr>
<td>diffeq</td>
<td>981</td>
<td>123</td>
<td>123</td>
<td>22</td>
<td>18.00</td>
</tr>
<tr>
<td>elliptic</td>
<td>2050</td>
<td>261</td>
<td>257</td>
<td>34</td>
<td>28</td>
</tr>
<tr>
<td>frisc</td>
<td>2282</td>
<td>288</td>
<td>286</td>
<td>50</td>
<td>40.00</td>
</tr>
<tr>
<td>s298</td>
<td>1053</td>
<td>134</td>
<td>133</td>
<td>38</td>
<td>36.00</td>
</tr>
<tr>
<td>s38417</td>
<td>4978</td>
<td>623</td>
<td>623</td>
<td>32</td>
<td>20.00</td>
</tr>
<tr>
<td>s38584</td>
<td>4497</td>
<td>563</td>
<td>559</td>
<td>30</td>
<td>20.00</td>
</tr>
<tr>
<td>tseng</td>
<td>779</td>
<td>98</td>
<td>98</td>
<td>16</td>
<td>12.00</td>
</tr>
<tr>
<td>Avg. Impv.</td>
<td>1.4</td>
<td>16.59</td>
<td>17.57</td>
<td>3.62</td>
<td></td>
</tr>
</tbody>
</table>
Traditional BLE4 FPGA

• Use WireMap implemented in ABC [10] to generate the netlists of 4-LUTs and compare our algorithm with T-VPack [9].
• We reduced the wirelength for all 20 benchmarks with 17.57% improvement on average.
• We achieved better minimum channel width in 18 of the 20 benchmarks with 16.59% improvement on average.
• Although ALMpack does not directly target to minimize delay, it still obtained 3.71% delay improvement on average compared to the timing-driven packing algorithm, T-VPack, while reducing the channel width by 16.59%.
Outline

• Introduction
• Preliminaries
• Algorithm
• Experimental results
• Conclusion
Conclusion

• We proposed a novel wirelength-driven algorithm to merge the LUTs and pack the ALMs to ensure that it will not adversely affect the final wirelength.

• The experimental results show that our packing algorithm consistently outperforms AAPack for ALM-based FPGA and T-VPack for traditional FPGA by a large margin.
Thank you for listening.