Recent Advances in Cut-based FPGA Technology Mapping

Kevin Chung
April 3, 2009
Logic synthesis and verification research is alive and vibrant.

FPGAs are growing fast – scalability in runtime and memory paramount.
1. Review of Cut-based Mapping
2. More Efficient Cut Computation
3. Lossless Synthesis
4. Priority Cuts
5. Area Recovery
6. WireMap
Cut-based Mapping Algorithm

Input: And-Inverter Graph

1. Compute all $K$-feasible cuts
2. Compute best arrival time at each node
   • In topological order (from PI to PO)
   • Assuming that each cut maps to a $K$-LUT
   • Assuming that each $K$-LUT has unit delay
3. Chose the best cover
   • In reverse topological order (from PO to PI)

Output: Mapped Netlist
Advantages

- Cuts have direct correspondence to LUTs
  - Easy to create LUT-based cost functions
    - different LUT input delays
    - output switching activity
- Cut computation is fast and simple
- Dynamic programming mapping solution
  - guarantees optimal delay
  - efficient search of LUT design space
Cut-based Mapping Challenges

- Feasible cuts grow quickly wrt LUT size
- Results depend upon AIG netlist structure
  - many possible equivalent AIG structures
  - logic restructuring optimizations that works well for one part of the design may not give good mapping for another

<table>
<thead>
<tr>
<th>K</th>
<th>Avg # of cuts per node</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>38</td>
</tr>
<tr>
<td>7</td>
<td>95</td>
</tr>
<tr>
<td>8</td>
<td>240</td>
</tr>
</tbody>
</table>
Outline

1. Review of Cut-based Mapping
2. More Efficient Cut Computation
   • Cut Dropping
   • Cut Dominance
3. Lossless Synthesis
4. Priority Cuts
5. Area Recovery
6. WireMap
Cut Dropping

During bottom up computation of cuts, the set of cuts of a node can be freed once all its fan-outs have been processed

\[
\{ \{r\}, \{p, q\}, \{p, b, c\}, \{a, b, q\}, \{a, b, c\} \}
\]

Bottom-up computation

- Once the cuts of node \(r\) are computed, the cuts of \(q\) are no longer needed
- But can’t discard the cuts of node \(p\) since not all fan-outs of \(p\) have been processed
- Dramatically reduces peak memory consumption on large designs
Bottom-up cut computation in the presence of re-convergence might produce *dominated* cuts

\[ x = \neg a + a \cdot b + \neg b \cdot c \]

- The “good” cut \{a, b, c\} is there: so not a quality issue
- But the “bad” cut \{a, d, b, c\} may be propagated further: so a run-time issue
- Want to discard dominated cuts *quickly*
Signature-based Dominance

Problem: Given two cuts how to quickly determine whether one is a subset of another

Define signature of a cut:

\[ \text{sig} (c) = \sum_{n \in c} 2^{\text{ID}(n) \mod 32} \]  

(\(\Sigma\) means bit-wise OR)

where \(\text{ID}(n)\) is the integer id of node \(n\)

Observation: If cut \(c_1\) dominates cut \(c_2\) then

\[ \text{sig}(c_1) \text{ OR sig}(c_2) = \text{sig}(c_2) \]

Cheap test for the common case that a cut does not dominate another. Only if this fails is an actual comparison made.
Example

- Let the node id’s be $a = 1$, $b = 2$, $c = 3$, $d = 4$
- Let $c_1 = \{a, b, c\}$ and $c_2 = \{a, d, b, c\}$
- $\text{sig}(c_1) = 2^1 \text{ OR } 2^2 \text{ OR } 2^3$
  - $= 0001 \text{ OR } 0010 \text{ OR } 0100$
  - $= 0111$
- $\text{sig}(c_2) = 2^1 \text{ OR } 2^4 \text{ OR } 2^2 \text{ OR } 2^3$
  - $= 0001 \text{ OR } 1000 \text{ OR } 0010 \text{ OR } 0100$
  - $= 1111$
- As $\text{sig}(c_1) \text{ OR } \text{sig}(c_2) \neq \text{sig}(c_1)$, $c_2$ does not dominate $c_1$
- But $\text{sig}(c_1) \text{ OR } \text{sig}(c_2) = \text{sig}(c_2)$, so $c_1$ may dominate $c_2$
## Run-time of $K$-feasible Cut Computation

<table>
<thead>
<tr>
<th>Name</th>
<th>$K = 4$</th>
<th>$K = 5$</th>
<th>$K = 6$</th>
<th>$K = 7$</th>
<th>$K = 8$</th>
<th>$L/N, %$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>C/N</td>
<td>T, s</td>
<td>C/N</td>
<td>T, s</td>
<td>C/N</td>
</tr>
<tr>
<td>alu4</td>
<td>2642</td>
<td>6.7</td>
<td>0.00</td>
<td>12.3</td>
<td>0.01</td>
<td>23.1</td>
</tr>
<tr>
<td>apex2</td>
<td>2940</td>
<td>7.2</td>
<td>0.01</td>
<td>14.2</td>
<td>0.02</td>
<td>29.2</td>
</tr>
<tr>
<td>apex4</td>
<td>2017</td>
<td>8.5</td>
<td>0.00</td>
<td>19.5</td>
<td>0.03</td>
<td>47.0</td>
</tr>
<tr>
<td>bigkey</td>
<td>3080</td>
<td>6.6</td>
<td>0.01</td>
<td>12.1</td>
<td>0.02</td>
<td>24.2</td>
</tr>
<tr>
<td>clma</td>
<td>11869</td>
<td>8.1</td>
<td>0.04</td>
<td>18.2</td>
<td>0.11</td>
<td>44.4</td>
</tr>
<tr>
<td>des</td>
<td>3020</td>
<td>8.0</td>
<td>0.01</td>
<td>17.0</td>
<td>0.03</td>
<td>38.7</td>
</tr>
<tr>
<td>diffeq</td>
<td>2566</td>
<td>6.5</td>
<td>0.01</td>
<td>12.3</td>
<td>0.01</td>
<td>26.6</td>
</tr>
<tr>
<td>dsip</td>
<td>2521</td>
<td>6.2</td>
<td>0.01</td>
<td>10.7</td>
<td>0.01</td>
<td>20.7</td>
</tr>
<tr>
<td>elliptic</td>
<td>5502</td>
<td>6.4</td>
<td>0.01</td>
<td>10.6</td>
<td>0.03</td>
<td>18.5</td>
</tr>
<tr>
<td>ex1010</td>
<td>7652</td>
<td>9.2</td>
<td>0.02</td>
<td>23.3</td>
<td>0.11</td>
<td>61.8</td>
</tr>
<tr>
<td>ex5p</td>
<td>1719</td>
<td>9.4</td>
<td>0.01</td>
<td>24.1</td>
<td>0.02</td>
<td>66.2</td>
</tr>
<tr>
<td>frisc</td>
<td>5905</td>
<td>7.1</td>
<td>0.01</td>
<td>14.4</td>
<td>0.04</td>
<td>32.3</td>
</tr>
<tr>
<td>misex3</td>
<td>2441</td>
<td>7.7</td>
<td>0.01</td>
<td>15.7</td>
<td>0.02</td>
<td>33.3</td>
</tr>
<tr>
<td>pdc</td>
<td>7527</td>
<td>9.4</td>
<td>0.03</td>
<td>24.8</td>
<td>0.12</td>
<td>67.4</td>
</tr>
<tr>
<td>s298</td>
<td>2514</td>
<td>7.9</td>
<td>0.00</td>
<td>17.5</td>
<td>0.02</td>
<td>44.0</td>
</tr>
<tr>
<td>s38417</td>
<td>12867</td>
<td>6.6</td>
<td>0.03</td>
<td>13.5</td>
<td>0.10</td>
<td>32.0</td>
</tr>
<tr>
<td>s38584</td>
<td>11074</td>
<td>6.1</td>
<td>0.03</td>
<td>11.4</td>
<td>0.06</td>
<td>22.4</td>
</tr>
<tr>
<td>seq</td>
<td>2761</td>
<td>7.5</td>
<td>0.00</td>
<td>15.2</td>
<td>0.02</td>
<td>31.7</td>
</tr>
<tr>
<td>spla</td>
<td>6556</td>
<td>9.6</td>
<td>0.03</td>
<td>25.8</td>
<td>0.11</td>
<td>73.9</td>
</tr>
<tr>
<td>tseng</td>
<td>1920</td>
<td>6.5</td>
<td>0.01</td>
<td>11.8</td>
<td>0.01</td>
<td>23.5</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>4954.6</strong></td>
<td><strong>7.56</strong></td>
<td><strong>0.01</strong></td>
<td><strong>16.22</strong></td>
<td><strong>0.05</strong></td>
<td><strong>38.05</strong></td>
</tr>
</tbody>
</table>
# Peak Memory in Mb with Cut Dropping

<table>
<thead>
<tr>
<th>Name</th>
<th>$K = 4$</th>
<th>$K = 5$</th>
<th>$K = 6$</th>
<th>$K = 7$</th>
<th>$K = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Drop</td>
<td>Total</td>
<td>Drop</td>
<td>Total</td>
</tr>
<tr>
<td>clma</td>
<td>2.56</td>
<td>0.10</td>
<td>6.60</td>
<td>0.22</td>
<td>18.09</td>
</tr>
<tr>
<td>ex1010</td>
<td>1.87</td>
<td>0.37</td>
<td>5.45</td>
<td>0.97</td>
<td>16.25</td>
</tr>
<tr>
<td>pdc</td>
<td>1.90</td>
<td>0.27</td>
<td>5.69</td>
<td>0.75</td>
<td>17.42</td>
</tr>
<tr>
<td>s38417</td>
<td>2.28</td>
<td>0.15</td>
<td>5.28</td>
<td>0.37</td>
<td>14.12</td>
</tr>
<tr>
<td>s38584.1</td>
<td>1.80</td>
<td>0.11</td>
<td>3.86</td>
<td>0.20</td>
<td>8.52</td>
</tr>
<tr>
<td>spla</td>
<td>1.68</td>
<td>0.21</td>
<td>5.15</td>
<td>0.59</td>
<td>16.63</td>
</tr>
<tr>
<td>Ratio</td>
<td>1.00</td>
<td>0.11</td>
<td>1.00</td>
<td>0.10</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Outline

1. Review of Cut-based Mapping
2. More Efficient Cut Computation
3. Lossless Synthesis
4. Priority Cuts
5. Area Recovery
6. WireMap
Structural Bias

The mapped netlist very closely resembles the subject graph

Every input of every LUT in the mapped netlist must be present in the subject graph ...

.. otherwise technology mapping will not find the match
Root problem: Best matches for mapping may not be found

Since the point q is not present in the subject graph, the match on the extreme right will not be found.
The Problem of Structural Bias

The match would be found with a different subject graph
No guarantee of optimality since each synthesis step is heuristic.

But structural bias means the mapped netlist depends heavily on the final network.

Since only network at the end of technology independent synthesis used for mapping, good intermediate netlists not used
Lossless Synthesis Flow

**Idea:** Merge intermediate networks into a single network with choices which can be explored during mapping

Technology-independent synthesis

- sweep
- eliminate
- resub
- simplify
- fx
- resub
- sweep
- eliminate
- sweep
- full simplify

Boolean Network

Choice operator

Technology mapping is *not* any harder with choices (Lehman-Watanabe ’95, Chen and Cong ‘01)

Technology Mapping

Mapped Netlist
Can combine results of different technology independent optimization scripts

Script optimizes area

sweep
eliminate
resub
simplify

fx
resub
sweep

eliminate
sweep
full simplify

Boolean Network

speed up
reduce depth

Script optimizes delay

Technology Mapping

Mapped Netlist
Question 1: How to implement an efficient choice operator?

Question 2: How to map quickly with choices?
Mapping with Choices

Question 1: How to implement an efficient choice operator?

Question 2: How to map quickly with choices?
Detecting Choices

Task: Given two Boolean networks, we need to create a network with choices

Network 1
\[ x = (a + b)c \]
\[ y = b.c.d \]

Network 2
\[ x = a.c + b.c \]
\[ y = b.c.d \]

Step 1: Make And-Inverter decomposition of networks
(dotted means inversion)
Step 2: Use combinational equivalence to detect functionally equivalent nodes up to complementation (*Kuehlmann* ’04, …)

- Random simulation to detect possibly equivalent nodes
- SAT-based decision procedure to prove equivalence

**Network 1**

\[ x = (a + b).c \]
\[ y = b.c.d \]

**Network 2**

\[ x = a.c + b.c \]
\[ y = b.c.d \]
Detecting Choices

Step 3: Merge equivalent nodes with choice edges

$x$ now represents a *class of nodes* that are functionally equivalent up to complementation.
Question 1: How to implement an efficient choice operator?

Question 2: How to map quickly with choices?
Mapping without Choices

Input: And-Inverter Graph

1. **Compute all $K$-feasible cuts**
2. **Compute best arrival time at each node**
   - In topological order (from PI to PO)
   - Assuming that each cut maps to a $K$-LUT
   - Assuming that each $K$-LUT has unit delay
3. **Chose the best cover**
   - In reverse topological order (from PO to PI)

Output: **Mapped Netlist**
Input: And-Inverter Graph with Choices

1. Compute all $K$-feasible cuts with choices
2. Compute best arrival time at each node
   - In topological order (from PI to PO)
   - Assuming that each cut maps to a $K$-LUT
   - Assuming that each $K$-LUT has unit delay
3. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

Only Step 1 requires modification
Cuts are now computed for *equivalence classes* of nodes

\[
\{ \{x_1\}, \{p, r\}, \{p, b, c\}, \{a, c, r\}, \{a, b, c\} \}
\]

\[
\{ \{x_2\}, \{q, c\}, \{a, b, c\} \}
\]

\[
\text{Cuts}(x) = \text{Cuts}(x_1) \cup \text{Cuts}(x_2)
\]

\[
= \{ \{x_1\}, \{p, r\}, \{p, b, c\}, \{a, c, r\}, \{a, b, c\}, \{x_2\}, \{q, c\} \}
\]
Input: And-Inverter Graph with Choices

1. Compute all $K$-feasible cuts with choices
2. Compute best arrival time at each node
   - In topological order (from PI to PO)
   - Assuming that each cut maps to a $K$-LUT
   - Assuming that each $K$-LUT has unit delay
3. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist

No changes needed except for Step 1
Lossless Synthesis Summary

Also called Mapping with Structure Choices

Advantages

- Equivalent netlist variations are recorded
  - mapping algorithm selects best among alternative structures to optimize a cost function
- Simple extension of mapping algorithm

Disadvantages

- Even more cuts to explore!
Outline

1. Review of Technology Mapping
2. More Efficient Cut Computation
3. Lossless Synthesis
4. Priority Cuts
5. Area Recovery
   1. Area-flow
   2. Exact Area
6. WireMap
Exhaustive Cut Enumeration Mapping

- Large designs have many $K$-feasible cuts
  - 1M node AIG has $\sim$40M 6-cuts
  - Needs $\sim$2GB and $\sim$30 sec for computation

- Past ways of tackling the problem
  - Detect and remove dominated cuts
    - Does not help much
  - Perform cut pruning (store N cuts/node)
    - Throws away useful cuts even if $N = 1000$
  - Store only cuts on the frontier
    - Reduces memory but increases runtime
Priority Cuts: A Bag of Tricks

- Compute and prioritize cuts (select subset of all cuts)
  - Fast and memory efficient – affordable for multiple passes
- Potentially lower quality overcome via multiple passes
  - Use different sorting criteria in each mapping pass to explore additional cost criteria
  - Include the best cut from the previous pass into the set of candidate cuts of the current pass
- Efficient memory management
  - Only maintain complete set of priority cuts for nodes on the mapping frontier
    - Precompute frontier to create efficiently managed memory pool
  - Only save best cut for each node
Computing Priority Cuts

- Consider nodes in a topological order
  - At each node, merge two sets of fanin cuts (each containing up to C cuts) getting \((C+1) \times (C+1) + 1\) cuts
  - Sort these cuts using a given cost function, select C best cuts, and use them for computing priority cuts of the fanouts
  - Select one best cut, and use it to map the node

- **Sorting criteria**

<table>
<thead>
<tr>
<th>Mapping pass</th>
<th>Primary metric</th>
<th>Tie-breaker 1</th>
<th>Tie-breaker 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>depth</td>
<td>depth</td>
<td>cut size</td>
<td>area flow</td>
</tr>
<tr>
<td>area flow</td>
<td>area flow</td>
<td>fanin refs</td>
<td>depth</td>
</tr>
<tr>
<td>exact area</td>
<td>exact area</td>
<td>fanin refs</td>
<td>depth</td>
</tr>
</tbody>
</table>
Priority-Cut-Based Mapping

Input: And-Inverter Graph

1. Compute all $K$-feasible cuts for each node
2. Compute arrival time at each node
   - In topological order (from PI to PO)
   - Compute the depth of all cuts and choose the best one
   - Compute at most $C$ good cuts and choose the best one
3. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist
Complexity Analysis

- **Traditional mapping algorithm**
  - FlowMap $O(K\cdot mn)$ (J. Cong et al, TCAD ’94)
  - CutMap $O(2^{K\cdot mn})$ (J. Cong et al, FPGA ’95)
  - DAOmap $O(K^{n^K})$ (J. Cong et al, ICCAD’04)

- **Proposed mapping algorithm**
  - $O(K\cdot C^2n)$
    - 6-LUT mapping has about 5X speedup
    - 8-LUT mapping has up to 100X speedup

$K$ is max cut size
$C$ is max number of cuts
$n$ is number of nodes
$m$ is number of edges

$C$ between 8 and 16 achieves optimal delay with good runtime
Outline

1. Review of Technology Mapping
2. More Efficient Cut Computation
3. Lossless Synthesis
4. Priority Cuts
5. Area Recovery
   1. Area-flow
   2. Exact Area
6. WireMap
Overview of Area Recovery

- **Initial mapping is delay oriented**
  - Gets best delay for all paths
  - Area-based tie-breaking

- **Not all paths critical**
  - Area recovery tries to slow down non-critical paths to reduce area
  - Each node with positive slack: choose a different cut that reduces area
  - Done as subsequent passes after delay-oriented mapping

- **Question: how to measure area?**
**Naïve definition:** Area (cut) = 1 + [ Σ area (fan-in) ]

Area of cut \{p, c, d\}
= 1 + [1 + 0 + 0]
= 2

Area of cut \{a, b, q\}
= 1 + [0 + 0 + 1]
= 2

Naïve definition says both cuts are equally good in area

**Naïve definition ignores sharing due to multiple fan-outs**
Area-flow

\[ AF(n) = 1 + \sum_i \frac{AF(Leaf_i(n))}{NumFanout(Leaf_i(n))} \]

Area-flow of cut \{p, c, d\}
= 1 + [1 + 0 + 0]
= 2

Area-flow of cut \{a, b, q\}
= 1 + [0/1 + 0/1 + ½]
= 1.5

Area-flow recognizes that cut \{a, b, q\} is better.

Area-flow “correctly” accounts for sharing and penalizes replication.
It is a floating point value!
Area Recovery with Area-flow

1. Do delay-optimal mapping
2. Compute slack at each node
3. Do area recovery with area-flow
   - Done in topological order from PI to PO
   - Among all the cuts which do not exceed slack budget choose cut with smallest area-flow
   - Fan-out of a node is estimated from delay optimal mapping
   - We only do one pass
     - Saw only marginal improvement on subsequent passes
Exact Area

Exact-area (cut) = 1 + [ \sum \text{exact-area (fan-in with no other fan-out)} ]
- Gives minimum area solution within an MFFC

Cut \{p, e, f\}
Area flow = 1 + [(0.25 + 0.25 + 3)/2] = 2.75
Exact area = 1 + 0 (p is used elsewhere)
Exact area will choose this cut.

Cut \{s, t, q\}
Area flow = 1 + [0.25 + 0.25 + 1] = 2.5
Exact area = 1 + 1 = 2 (due to q)
Area flow will choose this cut.
Area Recovery with Exact-area

1. Do delay-optimal mapping
2. Compute slack at each node
3. Do area recovery with area-flow
4. Do area recovery with exact-flow
   - Done in topological order from PI to PO
   - Among all the cuts which do not exceed slack budget choose cut with smallest exact-area
   - Note: Unlike area-flow, no estimation involved
   - We only do one pass
     • Saw only marginal improvement on subsequent passes
Input: And-Inverter Graph

1. Compute all $K$-feasible cuts for each node
2. Compute arrival time at each node
   - In topological order (from PI to PO)
   - Compute the depth of all cuts and choose the best one
   - Compute at most $C$ good cuts and choose the best one
3. Perform area recovery
   - Using area flow
   - Using exact local area
   - In each iteration, re-compute at most $C$ good cuts and choose the best one
4. Chose the best cover
   - In reverse topological order (from PO to PI)

Output: Mapped Netlist
Area Recovery Summary

- Two step area recovery
- Area-flow has global view
- Exact area has local view
  - Ensures local minimum is reached
- Order in which nodes are processed for both steps is important
- Order of the two passes is important
Experimental Comparison

- **Compare area-recovery with state-of-the-art academic mapper DAOmap**
  - DAOmap uses many (~10) different area recovery heuristics
  - Some more effective than others
- **Just the two heuristics of area-recovery and exact-area give better results on their benchmarks**
- **Also separate comparison with choices obtained from lossless synthesis flow**
  - Six snapshots of MVSIS script.rugged
  - Not the best FPGA optimization script 😊
  - Improves both area and delay
### Comparison with DAOmap

<table>
<thead>
<tr>
<th>Example</th>
<th>DAOmap</th>
<th>MVSIS-baseline</th>
<th>MVSIS-choices</th>
<th>MVSIS-choices 2x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Depth</td>
<td>LUTs</td>
<td>T, s</td>
<td>Depth</td>
</tr>
<tr>
<td>alu4</td>
<td>6</td>
<td>1065</td>
<td>0.5</td>
<td>6</td>
</tr>
<tr>
<td>apex2</td>
<td>7</td>
<td>1352</td>
<td>0.6</td>
<td>7</td>
</tr>
<tr>
<td>apex4</td>
<td>6</td>
<td>931</td>
<td>0.7</td>
<td>6</td>
</tr>
<tr>
<td>bigkey</td>
<td>3</td>
<td>1245</td>
<td>0.6</td>
<td>3</td>
</tr>
<tr>
<td>clma</td>
<td>13</td>
<td>5425</td>
<td>5.9</td>
<td>13</td>
</tr>
<tr>
<td>des</td>
<td>5</td>
<td>965</td>
<td>0.8</td>
<td>5</td>
</tr>
<tr>
<td>diffeq</td>
<td>10</td>
<td>817</td>
<td>0.6</td>
<td>10</td>
</tr>
<tr>
<td>dsip</td>
<td>3</td>
<td>686</td>
<td>0.5</td>
<td>3</td>
</tr>
<tr>
<td>elliptic</td>
<td>12</td>
<td>1965</td>
<td>2.0</td>
<td>12</td>
</tr>
<tr>
<td>ex1010</td>
<td>7</td>
<td>3564</td>
<td>4.0</td>
<td>7</td>
</tr>
<tr>
<td>ex5p</td>
<td>6</td>
<td>778</td>
<td>1.0</td>
<td>6</td>
</tr>
<tr>
<td>frisc</td>
<td>16</td>
<td>1999</td>
<td>1.9</td>
<td>15</td>
</tr>
<tr>
<td>misex3</td>
<td>6</td>
<td>980</td>
<td>0.8</td>
<td>6</td>
</tr>
<tr>
<td>pdc</td>
<td>7</td>
<td>3222</td>
<td>4.6</td>
<td>8</td>
</tr>
<tr>
<td>s298</td>
<td>13</td>
<td>1258</td>
<td>2.4</td>
<td>13</td>
</tr>
<tr>
<td>s38417</td>
<td>9</td>
<td>3815</td>
<td>3.8</td>
<td>9</td>
</tr>
<tr>
<td>s38584</td>
<td>7</td>
<td>2987</td>
<td>27.0</td>
<td>7</td>
</tr>
<tr>
<td>seq</td>
<td>6</td>
<td>1188</td>
<td>0.8</td>
<td>6</td>
</tr>
<tr>
<td>spla</td>
<td>7</td>
<td>2734</td>
<td>4.0</td>
<td>7</td>
</tr>
<tr>
<td>tseng</td>
<td>10</td>
<td>706</td>
<td>0.6</td>
<td>10</td>
</tr>
<tr>
<td><strong>Ratio</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>0.93</strong></td>
</tr>
</tbody>
</table>
Outline

1. Review of Cut-based Mapping
2. More Efficient Cut Computation
3. Lossless Synthesis
4. Priority Cuts
5. Area Recovery
6. WireMap
Motivation

- Cut-based mapping algorithms do well in minimizing LUT levels and area (LUT count)
  - Performance of circuit correlates to LUT levels
  - Logic block utilization correlates well to LUT count
- Could we change cut based mapping to improve netlist for packing, placement, routing?
- Area calculation gives each LUT equal weight – but should this be the case?
Virtex-5 LUT6
Can we produce smaller LUTs without increasing LUT levels?
Placement and Routing

- Routing is done for connections between inputs and outputs of a LUT (and other design elements)
- Fewer connections to route should make the design easier to place and route
- Can we come up with a mapping algorithm to minimize the total # of connections in a design?
Motivation Revisited

- Could we use cut based mapping to improve netlist for clustering, placement, routing?
  - Can we come up with a mapping algorithm to minimize the total # of connections in a design?
  - Can we produce smaller LUTs without increasing LUT levels?

- Area calculation gives equal weight to all LUTs – should that be the case?
Edge Recovery Overview

**Key:** Find a simple to compute cut metric that minimizes edge counts and creates more small LUTs

\[ EF(n) = \text{NumFanin}(n) + \sum_i \frac{EF(Leaf_i(n))}{\text{NumFanout}(Leaf_i(n))} \]

1. **Edge flow phase:** Use edge flow cost function to minimize global edge counts
2. **Exact edge phase:** Use optimal algorithm to minimize edge counts within MFFCs
   - Contrast with Area Flow eqn:

\[ AF(n) = 1 + \sum_i \frac{AF(Leaf_i(n))}{\text{NumFanout}(Leaf_i(n))} \]
Edge Flow Phase

1. Do delay-optimal mapping
2. Compute slack at each node
3. Do area recovery with area-flow with one change in how cuts are selected
   - Done in topological order from PI to PO
   - Among all the cuts which do not exceed slack budget choose cut with smallest area-flow
   - If 2 cuts have the same area-flow then choose the cut with the lower edge-flow
     • Edge flow is a tie breaker when area is within epsilon
Exact Edge Phase

1. Do delay-optimal mapping
2. Compute slack at each node
3. Do edge recovery with edge-flow
4. Do edge recovery with exact edge with one change
   - Done in topological order from PI to PO
   - Among all the cuts which do not exceed slack budget choose cut with smallest area, and to break ties choose cuts with lower number of edges
   - Note: Unlike edge-flow, no estimation involved
Modified Cut Prioritization Heuristics

- Consider nodes in a topological order
  - At each node, merge two sets of fanin cuts (each containing C cuts) getting \((C+1) \times (C+1) + 1\) cuts
  - Sort these cuts using a given cost function, select C best cuts, and use them for computing priority cuts of the fanouts
  - Select one best cut, and use it to map the node

- **Sorting criteria**

<table>
<thead>
<tr>
<th>Mapping pass</th>
<th>Primary metric</th>
<th>Tie-breaker 1</th>
<th>Tie-breaker 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>depth</td>
<td>cut size</td>
<td>area flow</td>
</tr>
<tr>
<td>area/edge flow</td>
<td>area flow</td>
<td>edge flow</td>
<td>depth</td>
</tr>
<tr>
<td>exact area/edge</td>
<td>exact area</td>
<td>exact edge</td>
<td>depth</td>
</tr>
</tbody>
</table>
Experimental Method

- Implemented WireMap using ABC
- Compared against two ABC mapping algorithms
  - Baseline – mapping with area recovery
  - Mapping with Structure Choices (MSC) – area-recovery mapping with alternative netlists produced by synthesis
- WireMap was built on top of MSC
- Performed packing of single-output LUTs to dual-output LUTs using maximum cardinality matching
- Used VPR to place/route design for wirelength and critical path delays
WireMap Results

- **MSC is superior to baseline mapping**
  - Single-output LUT count reduced by 9.1%
  - Edge count reduced by 8.1% and dual-output LUT count reduced by 7.7% - similar level of reduction as single-output LUT count

- **WireMap leads to further reduction in edges by 9.3% and dual-output LUT count by 9.4% versus MSC**
  - Single-output LUT count only reduced by 1.3% wrt. MSC

- **WireMap improvements to edges and dual-output LUTs not directly related to single-output LUT count reduction**
The histogram below shows how the single-output LUT size distribution was modified leading to a 9.4% reduction in dual output LUT6s.
WireMap Results – Place and Route

• Wirelength was reduced by 8.5% vs. MSC

• Minimum channel width reduced by 6%.

• Critical path delay reduced by 2.3%.
WireMap Summary

- Edge recovery cut-based mapping algorithm that extends area recovery heuristic with an edge cost function
  - area flow -> edge flow
  - exact area -> exact edge

- Minimizes total # of connections in the design

- Improves packing by increasing frequency of smaller LUTs
Overall Summary

- **Cut-based mapping is efficient and flexible**
- **Lossless Synthesis**
  - Map over multiple synthesis snapshots
- **Priority Cuts**
  - Limit # of cuts explored
    - Runtime and memory scalability
    - Without compromising QoR
- **Improved area recovery**
  - Global area-flow and local exact area
  - Order of application is important
- **WireMap**
  - Pack/place/route friendly cut-based mapping algorithm
Key Takeaways

- Pay attention to runtime and memory scalability
- Defer choices between alternative implementations to later phases that make better decisions
- Global optimization followed by exact local optimization is effective
- Overcome suboptimal solution via multiple passes that explore different corners of the optimization space
- Best solutions consider what is done in synthesis, mapping, placement and routing
References

- S. Jang, B. Chan, K. Chung, and A. Mishchenko, "WireMap: FGPA technology mapping for improved routability". Proc. FPGA '08. PDF
- S. Cho, S. Chatterjee, A. Mishchenko, and R. Brayton, "Efficient FPGA mapping using priority cuts". (Poster.) Proc. FPGA '07. PDF
- All publications for ABC: http://www.eecs.berkeley.edu/~alanmi/publications/