Advanced Datapath Synthesis using Graph Isomorphism

Cunxi Yu\textsuperscript{1,3},
Mihir Choudhury\textsuperscript{2}, Andrew Sullivan\textsuperscript{2}, Maciej Ciesielski\textsuperscript{1}

University of Massachusetts, Amherst \textsuperscript{1}
IBM T.J Watson Research Center \textsuperscript{2}
LSI, EPFL \textsuperscript{3}
Motivation

- **Logic sharing**
  - Minimize logic area
  - High-level synthesis
    - e.g. 1MUX + 2ADD -> 2MUXs + 1 ADD
    - If Op1 ≠ Op2?
  - Logic synthesis?
    - Logic sharing in *sea of gates*
    - Identify MUX logic and common logic attached
Motivation

- **Logic sharing**
  - Minimize logic area
  - High-level synthesis
    - e.g. $1\text{MUX} + 2\text{ADD} \rightarrow 2\text{MUXs} + 1\text{ADD}$
    - If $\text{Op1} \neq \text{Op2}$?
  - Logic synthesis?
    - Logic sharing in sea of gates
    - Identify MUX functions and common logic
  - Approach
    - And-Inv-Graph (AIG) isomorphism [Yu et. al DAC16]
    - Standard-cell based isomorphism
Overview

- **Our approach**
  - Logic-level logic sharing
  - Graph isomorphism
  - **Before** technology mapping
  - Evaluation
    - After logic synthesis
    - After PnR
Pre-Processing

- **Identify multiplexers**
- **Vector multiplexers**

\[
Z = A \cdot \overline{S} + B \cdot S
\]

\[
m' = \overline{m} = i_1 \cdot \overline{i_2} = i_1 + i_2
\]

\[
m = \overline{i_1} \cdot i_2
\]

\[
i_i = A \cdot \overline{S} \quad i_2 = B \cdot S
\]

Boolean network diagram with multiplexers and boolean expressions.
Pre-Processing

- **Identify multiplexers**
  - Vector multiplexers
- **Create sub-network**
  - Where our approach applied
  - Bounded by PIs/Latches and outputs of MUXes

Boolean network

\[
\begin{align*}
Z &= A \cdot \overline{S} + B \cdot S \\
m' &= m = i_1 \cdot i_2 = i_1 + i_2 \\
m &= i_1 \cdot \overline{i_2} \\
i_1 &= A \cdot \overline{S} \quad i_2 = B \cdot S
\end{align*}
\]
Multiplexer Relocation

- **Problem challenge**
  - Identify the bound and match Boolean signals

- **Functional method**
  - Formal methods, e.g. *BDDs, Satisfiability*
    - Require known input boundary and matching
    - Scalability

- **Structural method**
  - Graph isomorphism
    - Add nodes in next level which maintain G1 and G2 in an isomorphism class (G1 ≃ G2)

---

Graph Isomorphism

- $G \cong H$
  - $G$ and $H$ is a bijection between the vertex sets
    - $a \leftrightarrow 1$, $b \leftrightarrow 6$, $c \leftrightarrow 8$, etc.
    - Quasi-polynomial? [L. Babai'15]
Multiplexer Relocation

- **Common specification logic**
  - Maximize and match the boundary
Multiplexer Relocation

- **Common specification logic**
  - Maximize and match the boundary
Common specification logic
- Maximize and match the boundary

Multiplexer Relocation
Multiplexer Relocation

- **Create new network**
  - Keep one common logic L1
  - Extra multiplexers may be added
    - Muxing the inputs of L1 and L2
AIG-based Isomorphism

- **Common Logic**
  - Node edge 01/10
  - Node edge 11/00
    - Two candidate pairs
      - 3 -> 13 or 3 -> 14
    - Check next level
      - Maximize the identical logic
      - Depth = 3

- **Input matching**
  - [5, 6, 7, 15, 16, 17]
Demo

MUX (node 4) identified
select: s
Bound : [ A1,A2; B1,B2 ]
Logic : [11,12,13,14,15]
Demo

Two MUXs (21, 24) created
Demo
STD-based Isomorphism

- **Limitations of AIG isomorphism**
  - Complexity is high for large common logic

- **Standard-cell DAG**
  - More types of nodes
  - Graph is unweighted
  - More compact
STD-based Isomorphism

- **Standard-cell DAG**
  - Level0: g1 – g6
    - g4 – g7
  - Level1: g2 – g8
    - e – e’
STD-based Isomorphism

- **Standard-cell DAG**
  - Weights of input edges are the same

```
  1
 /|
/  |\
  2 3
/   |
  4   5
|    |
|    |
|    |
|    |
  6   7

  8
 /|
/  |\
  9 10
/   |
  11
/   |
  12
/   |
  13
/   |
  14

  e
/|
/  |\
  c  d
/   |
  a   b
/   |
  a'  b'
/   |
  c'  d'
/   |
  e'
```
Improve CSL identification

- Side fanout information
  - Nodes with same number of fanouts
    - n0, n4 have 0 fanout; n1, n5 have 1 fanout
    - n2, n6 have 2 fanouts; n3, n have 3 fanouts
Approximate isomorphism

- **Exact isomorphic class**
  - Appx Case 1 – INV2XOR
    - S=1, g1 is inverted
    - S=0, g1 is not inverted
    - Replace INV with XOR2 with extra input s or s’
      - S=1, XOR2 = inverter
      - S=0, XOR2 = wire
Beyond Graph Isomorphism

- **Appx – Case 2**

\[
\begin{align*}
f(a, b, c, d) &= \neg(a \land b \land c \land d) \\
f'(e, f, g, h) &= \neg((e \land f) \lor (g \land h))
\end{align*}
\]

![Diagram showing digital logic circuits](image)
Beyond Graph Isomorphism

- **Appx – Case 2**

\[
\begin{align*}
  f(a, b, c, d) &= \neg(a \land b \land c \land d) \\
  f'(e, f, g, h) &= \neg((e \land f) \lor (g \land h))
\end{align*}
\]
Beyond Graph Isomorphism

- **Appx – Case 2**
  \[ f(a, b, c, d) = \neg (a \land b \land c \land d) \]
  \[ f'(e, f, g, h) = \neg ( (e \land f) \lor (g \land h) ) \]
  - \textit{inv}(n) \rightarrow \textit{nor}(n, 1)
Beyond Graph Isomorphism

- **Appx – Case 2**
  
  \[
  f(a, b, c, d) = \neg (a \land b \land c \land d)
  \]
  
  \[
  f'(e, f, g, h) = \neg ( (e \land f) \lor (g \land h) )
  \]

  - \text{inv}(n) \rightarrow \text{nor}(n, 1), \text{nand}(c, d) \rightarrow \text{nand}(1, 1, c, d)
Beyond Graph Isomorphism

- **Extension**

  Given Boolean functions $P(x)$ and $Q(y)$, there always exists one function $F(z)$, such that $F(x,0,1)=F(x)$ and $F(y,0,1)=Q(y)$.

\[
\exists F(z) \ (F(x,0,1) = P(x)) \land (F(y,0,1) = Q(y))
\]
Implementation

- **Multiplexer relocation**
  - Single mux relocation iteratively

- **Identify common logic**
  - Step1: ignore inverters but storing their positions
  - Step2: minimize the number of inv2xor replacements

```
Algorithm 2 Single Multiplexer Relocation
Input: Pre-processed sub-circuit C
Output: An optimized standard-cell netlist

Single_Mux_Relocate(C)
1: B = RelocationBoundray(PO)
2: C ← relocate multiplexer to level B, w/o considering inverters
3: P = inv2xorPosition(PO, B)
4: C ← insert XORs to P based on its location
5: return C

RelocationBoundray(PO)
1: m ← levels(PO) - 1; inverter is considered as 0 level.
2: while m ≥ 0 do
3:   L0,m ← the gates in (s = 0) logic at m level
4:   L1,m ← the gates in (s = 1) logic at m level
5:   if (L0,m, L1,m) ← uniqueFanoutPairs(L0,m, L1,m)
   then
6:     L0,m-1, L1,m-1 ← uniqueFanoutPairs(L0,m, L1,m)
7:     L0,m ← L0,m ∩ L0′(m), L1,m ← L1,m ∩ L1′(m)
8:     L0,m-1, L1,m-1 = isomorphsim(L0,m, L1,m)
9:   else
10:      if isomorphsim(L0,m, L1,m) then
11:        L0,m+1, L1,m+1 ← isomorphsim(L0,m, L1,m)
12:      else
13:        exit
14:      end if
15:   end if
16: end while
17: return (level(PO) - 1 - m), (L0,m-1, L1,m-1)

inv2xorPosition(PO, boundary)
1: P0 ← the positions of all inverts till boundary level
2: P1 ← the positions of all inverts till boundary level
3: return P0 ∩ P1
```
Example

- **MUX2**
  - AOI22+INV
Example

- **Level0**
  - \{a_0\}_{s=1} - \{b_0\}_{s=0}
Example

- **Level0**
  - $\{a_0\}_{s=1} - \{b_0\}_{s=0}$

- **Level1**
  - $\{a_1,a_2\}_{s=1} - \{b_1,b_2\}_{s=0}$
    - $a_2,b_2$ has two fanouts
  - INV skipped $\{0,0,1\}_{s=1} - \{0,0,0\}_{s=0}$
Example

- **Level 0**
  - \( \{a_0\}_{s=1} - \{b_0\}_{s=0} \)

- **Level 1**
  - \( \{a_1, a_2\}_{s=1} - \{b_1, b_2\}_{s=0} \)
    - \( a_2, b_2 \) has two fanouts
    - INV skipped \( \{0,0,1\}_{s=1} - \{0,0,0\}_{s=0} \)

- **Level 2**
  - \( \{a_3, a_4, a_5\}_{s=1} - \{b_3, b_4, b_5\}_{s=0} \)
    - \( [0,0,1,0,0...,0,0,1,0,0,0] \)
    - \( [0,0,0,0,0...,0,0,1,0,0,0] \)
Example

- **Level 0**
  - \( \{a_0\}_{s=1} - \{b_0\}_{s=0} \)

- **Level 1**
  - \( \{a_1, a_2\}_{s=1} - \{b_1, b_2\}_{s=0} \)
    - \( a_2, b_2 \) has two fanouts
    - INV skipped \( \{0,0,1\}_{s=1} \)

- **Level 2**
  - \( \{a_3, a_4, a_5\}_{s=1} - \{b_3, b_4, b_5\}_{s=0} \)
    - \([0,0,1,0,0...,0,0,1,0,0,0] \)
    - \([0,0,0,0...,0,0,1,0,0,0] \)

\[ m_i = x_i s + y_i s \]
Example

- **Level0**
  - \{a_0\}_{s=1} - \{b_0\}_{s=0}

- **Level1**
  - \{a_1, a_2\}_{s=1} - \{b_1, b_2\}_{s=0}
    - \(a_2, b_2\) has two fanouts
    - INV skipped \{0,0,1\}_{s=1}
  
- **Level2**
  - \{a_3, a_4, a_5\}_{s=1} - \{b_3, b_4, b_5\}_{s=0}
    - \[0,0,1,0,0...,0,0,1,0,0,0\]
    - \[0,0,0,0...,0,0,1,0,0,0\]
Results

- Evaluation after technology mapping
  - 14nm
  - Including complete high-level and logic synthesis
    - 35% area reduction

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Lev</td>
<td>Area</td>
</tr>
<tr>
<td>A:B:A&lt;C</td>
<td>2280</td>
<td>11</td>
<td>2124</td>
</tr>
<tr>
<td>A+B, A+C</td>
<td>10162</td>
<td>17</td>
<td>9333</td>
</tr>
<tr>
<td>A+B:A-C</td>
<td>8697</td>
<td>19</td>
<td>8104</td>
</tr>
<tr>
<td>A:B:A&lt;=B</td>
<td>2464</td>
<td>12</td>
<td>2126</td>
</tr>
<tr>
<td>A×B:A×C</td>
<td>182917</td>
<td>83</td>
<td>482811</td>
</tr>
<tr>
<td>A×B/C[7:0]:A×B/C[15:8]</td>
<td>3626</td>
<td>26</td>
<td>5606</td>
</tr>
<tr>
<td>A×B+C:B×C+A</td>
<td>52943</td>
<td>58</td>
<td>108402</td>
</tr>
<tr>
<td>dec(A):dec(B)</td>
<td>1319</td>
<td>5</td>
<td>667</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>+0 lev</td>
<td>1.106</td>
</tr>
</tbody>
</table>
Results

- **Complex designs**
  - Flow1: without AIG optimization
    - ~40% area reduction, delay remains the same
  - Flow2: with AIG optimization
    - ~50% area reduction, 25% delay improvement

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area</td>
<td>Delay</td>
<td>Area</td>
<td>Delay</td>
</tr>
<tr>
<td>ibm1</td>
<td>3622</td>
<td>216.45</td>
<td>2223</td>
<td>255.81</td>
</tr>
<tr>
<td>ibm2</td>
<td>5454</td>
<td>314.84</td>
<td>3361</td>
<td>354.19</td>
</tr>
<tr>
<td>ibm3</td>
<td>9115</td>
<td>501.77</td>
<td>5526</td>
<td>501.77</td>
</tr>
<tr>
<td>ibm4</td>
<td>12782</td>
<td>678.87</td>
<td>7874</td>
<td>649.35</td>
</tr>
<tr>
<td>ibm5</td>
<td>18323</td>
<td>924.84</td>
<td>11121</td>
<td>787.10</td>
</tr>
<tr>
<td>ibm6</td>
<td>27435</td>
<td>1170.81</td>
<td>16843</td>
<td>983.87</td>
</tr>
<tr>
<td>ibm7</td>
<td>31069</td>
<td>1288.87</td>
<td>19083</td>
<td>1082.26</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0.613</td>
<td>0.970</td>
</tr>
</tbody>
</table>
Results

- **Runtime: AIG vs. STD graph**
  - Implemented in C++, within IBM synthesis flow
  - Xeon CPU 7560 v6 x32, 4TB Memory
  - Benchmark: sel?mult1:mult2
Results

- Physical design evaluation
What next?

- Physical design evaluation
  - Large fanout signals generated
  - Extra MUXes placed tightly
    - E.g., two common logics have 1 output, 64-bit input
      - 1 mux vs. 64 extra muxes with identical controls

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Route Length</th>
<th>Power</th>
<th>Worst-case delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm1</td>
<td>0.73</td>
<td>0.45</td>
<td>0.95</td>
</tr>
<tr>
<td>ibm2</td>
<td>0.79</td>
<td>0.61</td>
<td>0.97</td>
</tr>
<tr>
<td>ibm4</td>
<td>0.92</td>
<td>0.71</td>
<td>1.06</td>
</tr>
<tr>
<td>ibm6</td>
<td>1.23</td>
<td>0.78</td>
<td>1.10</td>
</tr>
</tbody>
</table>
What next?

![Bar chart showing different optimization levels](image)

- Logic level
- Tech map
- Placement
- Clock Opt
- Fine Opt
Thank you!