Advanced Datapath Synthesis using Graph Isomorphism

Cunxi Yu$^{1,3}$, Mihir Choudhury$^{2}$, Andrew Sullivan$^{2}$, Maciej Ciesielski$^{1}$

University of Massachusetts, Amherst $^{1}$
IBM T.J Watson Research Center $^{2}$
LSI, EPFL $^{3}$
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. 1MUX + 2ADD -> 2MUXs + 1 ADD
    - If Op1 ≠ 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

![Diagram showing the overview of the design process with stages like Design Specification, Verilog or VHDL, High-level Synthesis, Logic Synthesis, Physical Design, Tape-out process, Chip, Compilation, Scheduling, Binding, Floorplanning, Placement, Clock Opt, and Global routing. The Our Approach section focuses on Decomposition, Structural Opt, and Technology map. Evaluation points are highlighted with an arrow.]
Pre-Processing

- **Identify multiplexers**

- **Vector multiplexers**

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

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

\[
m = i_1 \cdot i_2
\]

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

- Identify multiplexers
  - Vector multiplexers

- Create sub-network
  - Where our approach applied
  - Bounded by PIs/Latches and outputs of MUXes
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 [1]
    - Scalability

- **Structural method**
  - Graph isomorphism
    - Add nodes in next level which maintain $G_1$ and $G_2$ in an isomorphism class ($G_1 \cong G_2$)

[1] Katebi, Hadi, and Igor L. Markov. "Large-scale Boolean matching." DATE 2010
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
Multiplexer Relocation

- **Common specification logic**
  - Maximize and match the boundary
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]
\[ Z = A \cdot \bar{S} + B \cdot S \]
\[ m' = m = \bar{i}_1 \cdot \bar{i}_2 = i_1 + i_2 \]
\[ m = \bar{i}_1 \cdot \bar{i}_2 \]
\[ i_1 = A \cdot \bar{S} \quad i_2 = B \cdot S \]
Demo

MUX (node 4) identified

A1
B1

A2
B2

S

15
14
13
12
11

19
18
17
16
15
14
13
12
11

4

5

6

21

Z
Demo

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
Improve CSL identification

- **Side fanout information**
  - Nodes with same number of fanouts
    - $n_0, n_4$ have 0 fanout; $n_1, n_5$ have 1 fanout
    - $n_2, n_6$ have 2 fanouts; $n_3, n$ 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

![Diagram of circuit](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**

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

  \[ f(a, b, c, d) = \neg(a \land b \land c \land d) \]
  \[ f'(e, f, g, h) = \neg((e \land f) \lor (g \land h)) \]
  
  - `inv(n)` → `nor(n, 1)`, `nand(c, d)` → `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 = RelocationBoundary(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

RelocationBoundary(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-1} ← L0_m ∩ L0'(m), L1_{m-1} ← 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:         end if
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: P_0 ← the positions of all inverters till boundary level
2: P_1 ← the positions of all inverters till boundary level
3: return P_0 ∩ P_1
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

- **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}

- **Level2**
  - \{a_3, a_4, a_5\}_{s=1} - \{b_3, b_4, b_5\}_{s=0}
    - \([0,0,1,0,0\ldots,0,0,1,0,0,0]\]
    - \([0,0,0,0\ldots,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**
  - \( \{a_3, a_4, a_5\}_{s=1} - \{b_3, b_4, b_5\}_{s=0} \)
    - \([0,0,1,0,0\ldots,0,0,1,0,0,0]\)
    - \([0,0,0,0\ldots,0,0,1,0,0,0]\)

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

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

| (n-bit) Operators          | Origin. Flow       | Origin. Flow with AIG Opt | Origin. Flow with our approach |
|----------------------------|--------------------|---------------------------|--------------------------------|
|                            | Area   | Lev | Area   | Lev | Area   | Lev |
| (64), A<B: A<C             | 2280   | 11  | 2124   | 13  | 1855   | 15  |
| (64), A+B, A+C             | 10162  | 17  | 9333   | 15  | 5787   | 20  |
| (64), A+B:A-C              | 8697   | 19  | 8104   | 25  | 8062   | 21  |
| (64) A<B:A<=B              | 2464   | 12  | 2126   | 13  | 2198   | 12  |
| * (64) A×B:A×C             | 182917 | 83  | 482811 | 211 | 91245  | 89  |
| A×B/C[7:0]:A×B/C[15:8]     | 3626   | 26  | 5606   | 26  | 1760   | 27  |
| (32) A×B+C:B×C+A           | 52943  | 58  | 108402 | 120 | 26709  | 58  |
| (6) dec(A):dec(B)          | 1319   | 5   | 667    | 5   | 549    | 7   |

|                            |        |     |        |     |        |     |
|----------------------------|--------|-----|--------|-----|--------|-----|
| 1                          | +0 lev |     | 1.106  | +1.16 lev | 0.658 | +2 lev |
Results

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

| Benchmarks | Origin. Flow1 | Origin. Flow1 with our approach | Origin. Flow2 | Origin. Flow2 with our approach |
|------------|---------------|---------------------------------|---------------|---------------------------------|
|            | Area          | Delay                           | Area          | Delay                           |
|            | 3622          | 216.45                          | 2223          | 255.81                          |
|            | 5454          | 314.84                          | 3361          | 354.19                          |
|            | 9115          | 501.77                          | 5526          | 501.77                          |
|            | 12782         | 678.87                          | 7874          | 649.35                          |
|            | 18323         | 924.84                          | 11121         | 787.10                          |
|            | 27435         | 1170.81                         | 16843         | 983.87                          |
|            | 31069         | 1288.87                         | 19083         | 1082.26                         |
|            | **1**         | **0.613**                       | **1**         | **0.487**                       |

|            | Area          | Delay                           | Area          | Delay                           |
|            | 4587          | 295.16                          | 6879          | 432.90                          |
|            | 5610          | 541.13                          | 3366          | 383.71                          |
|            | 12342         | 875.65                          | 16803         | 1023.23                         |
|            | **1**         | **0.767**                       | **1**         | **0.613**                       |
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

| Benchmarks | Route Length | Power | Worst-case delay |
|------------|--------------|-------|------------------|
| ibm1       | 0.73         | 0.45  | 0.95             |
| ibm2       | 0.79         | 0.61  | 0.97             |
| ibm4       | 0.92         | 0.71  | 1.06             |
| ibm6       | 1.23         | 0.78  | 1.10             |
What next?

![Bar chart showing performance metrics for different stages of a process: Logic level, Tech map, Placement, Clock Opt, and Fine Opt. The y-axis represents performance with a scale from 0 to 1.2.]
Thank you !