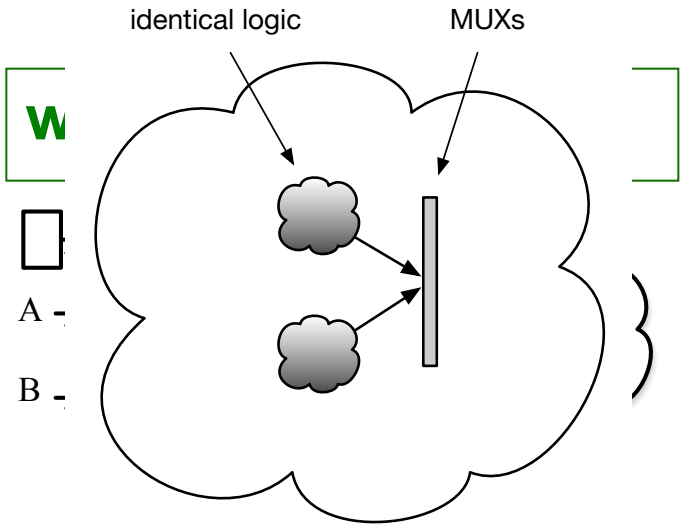
Advanced Datapath Synthesis using Graph Isomorphism

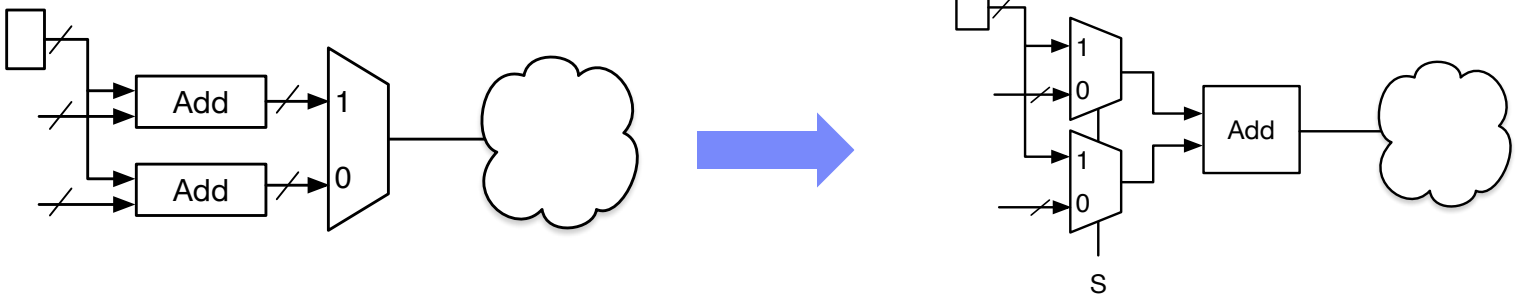
Cunxi Yu[1,3],
Mihir Choudhury[2], Andrew Sullivan[2], Maciej Ciesielski[1]

University of Massachusetts, Amherst [1]
IBM T.J Watson Research Center [2]
LSI, EPFL[3]

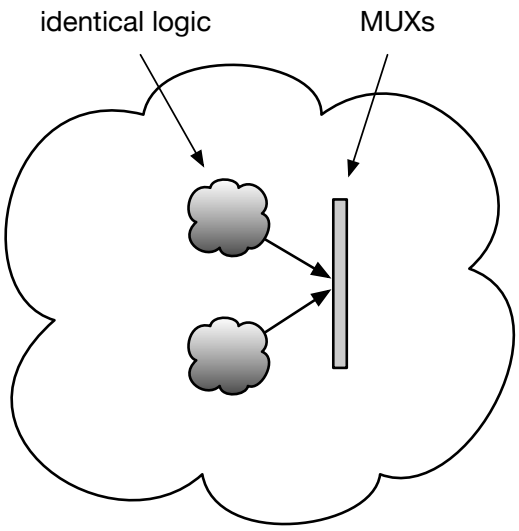
Motivation

- **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





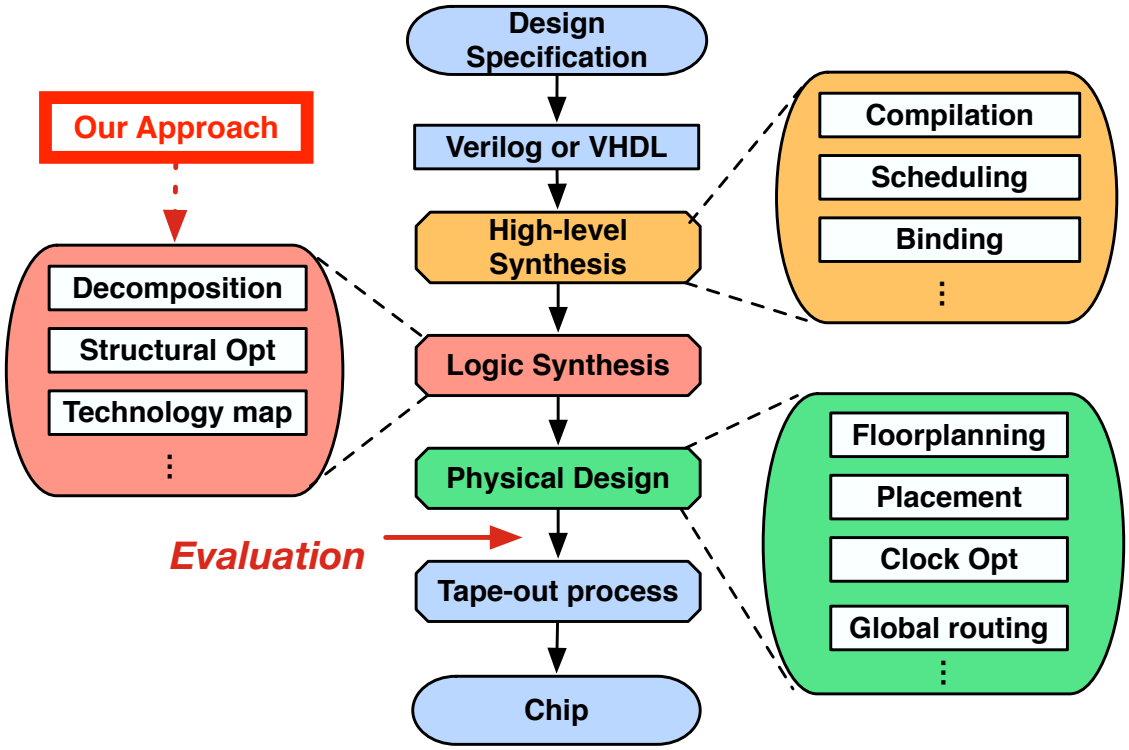
Motivation



- **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

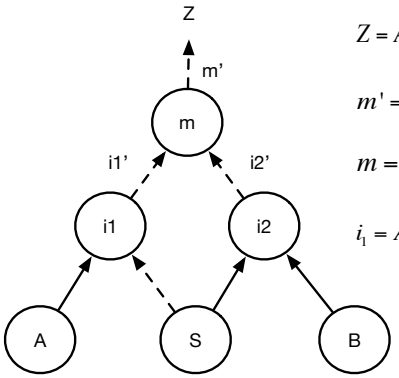
Overview

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



Pre-Processing

- **Identify multiplexers**
  - Vector multiplexers

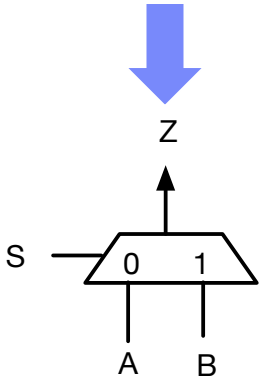


$$Z = A \cdot \overline{S} + B \cdot S$$

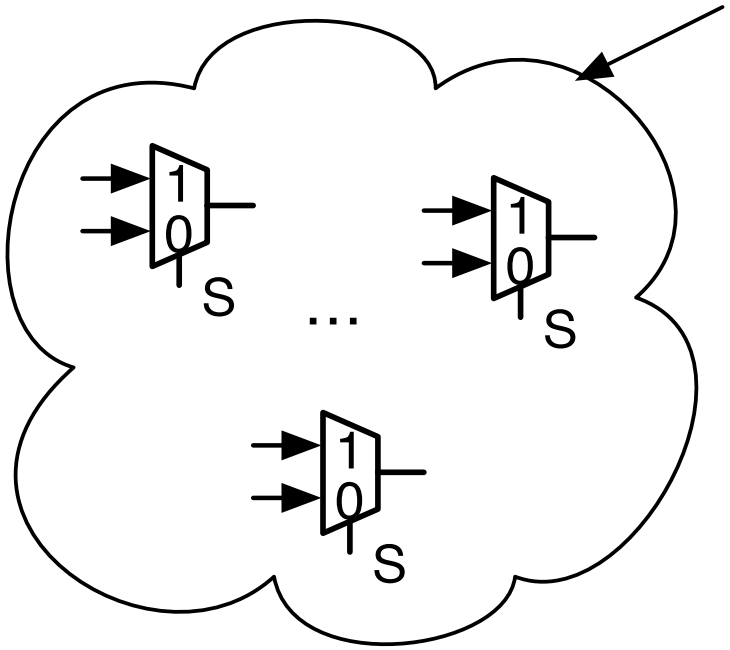
$$m' = \overline{m} = \overline{\overline{i_1} \cdot \overline{i_2}} = i_1 + i_2$$

$$m = \overline{i_1} \cdot \overline{i_2}$$

$$i_1 = A \cdot \overline{S} \quad i2 = B \cdot S$$

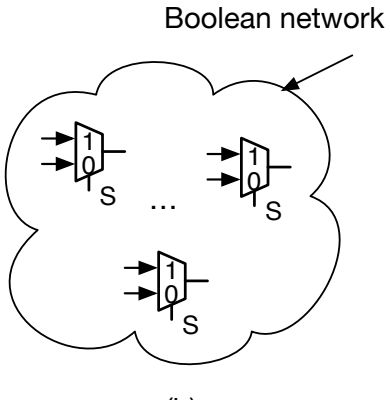


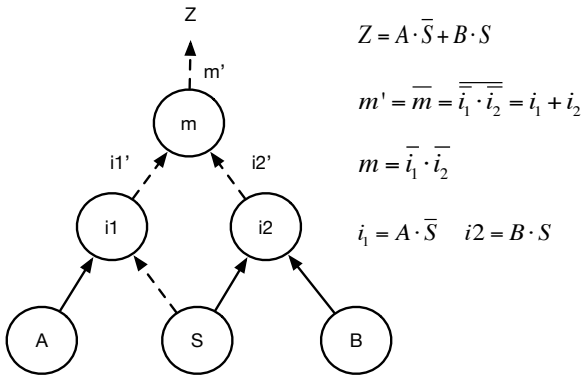
Boolean network

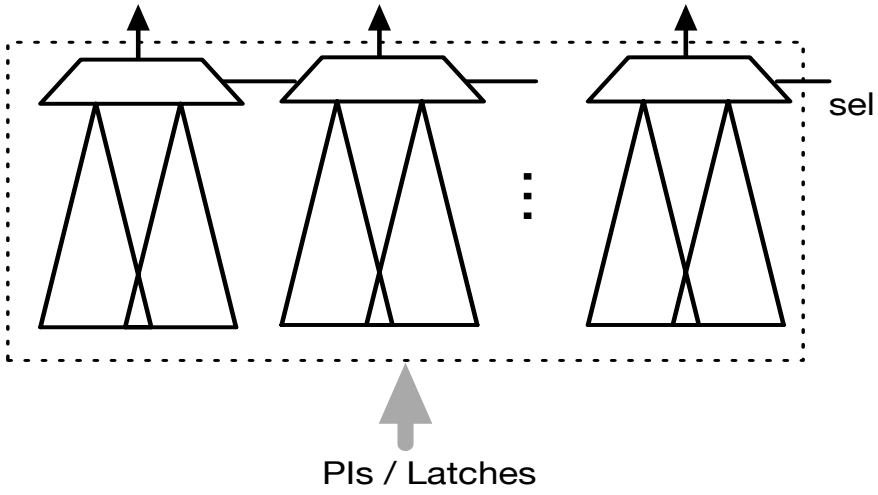


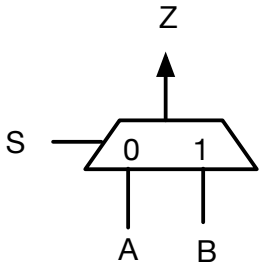
Pre-Processing

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



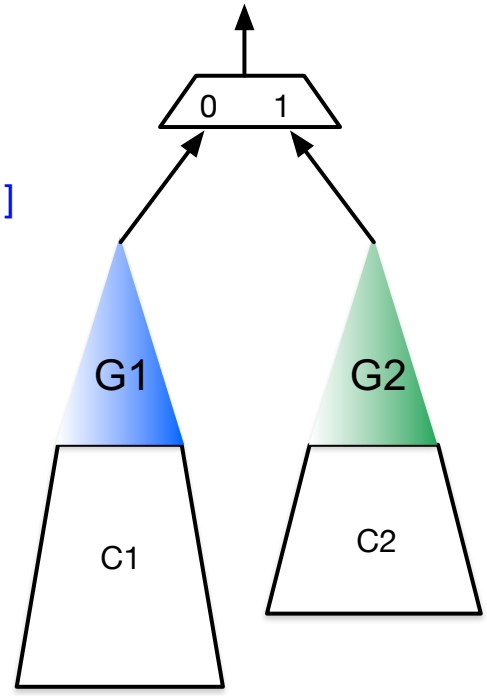






Multiplexer Relocation

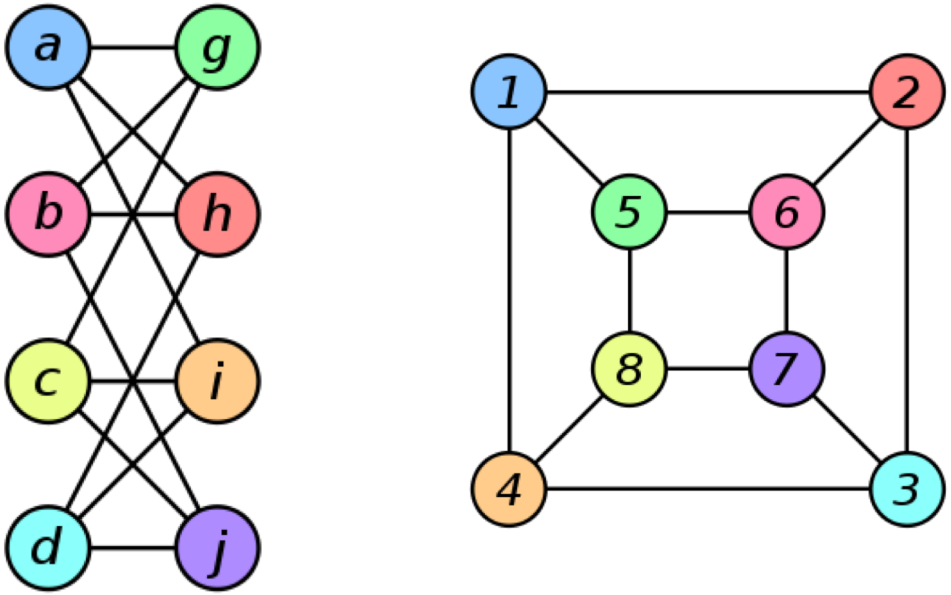
- **Problem challenge**
  - Identify the bound and match Boolean signals
- **Functional method**
  - Formal methods, e.g. *BDDs*, *Satisfiability*
    - Require known input boundary and matching [1]
    - Scalability
- **Structural method**
  - Graph isomorphism
    - Add nodes in next level which maintain G1 and G2 in an isomorphism class (G1≃G2)



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

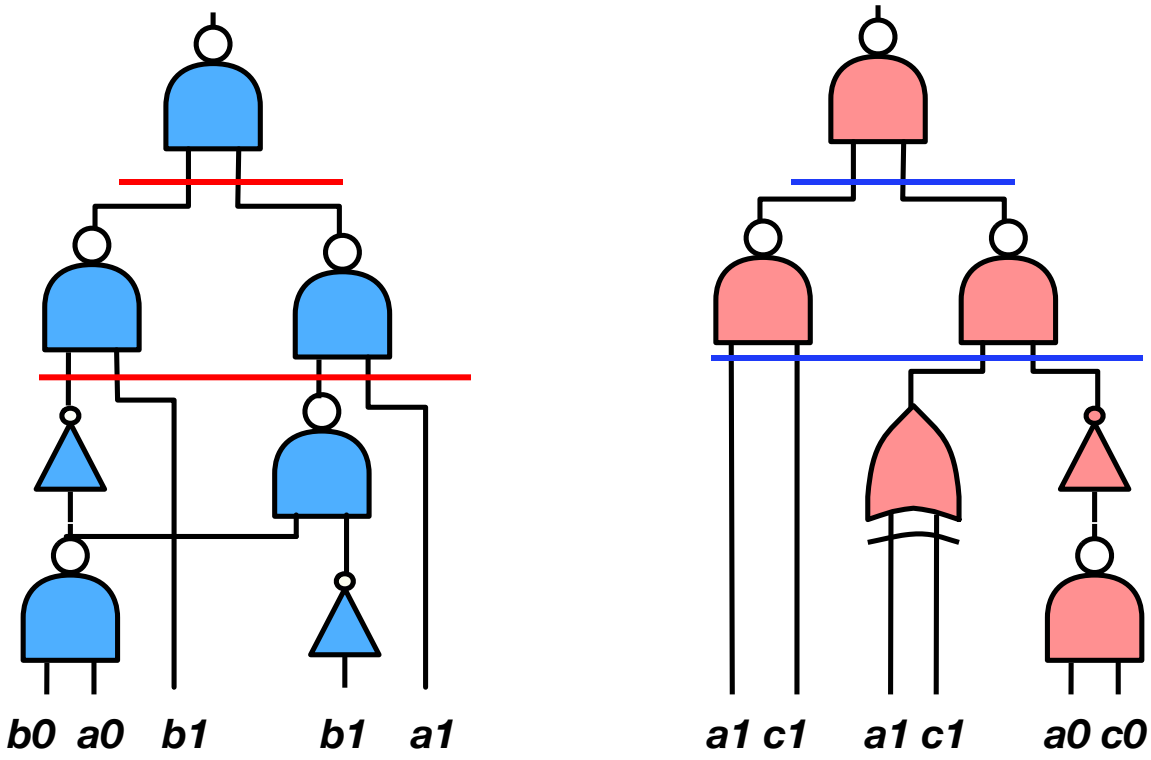
Graph Isomorphism

- **G ≃ H**
  - G and H is a bijection between the vertex sets
    - *a* ↔*1*, *b* ↔ *6*, *c* ↔ *8*, etc.
    - Quasi-polynomial ? [L. Babai'15]



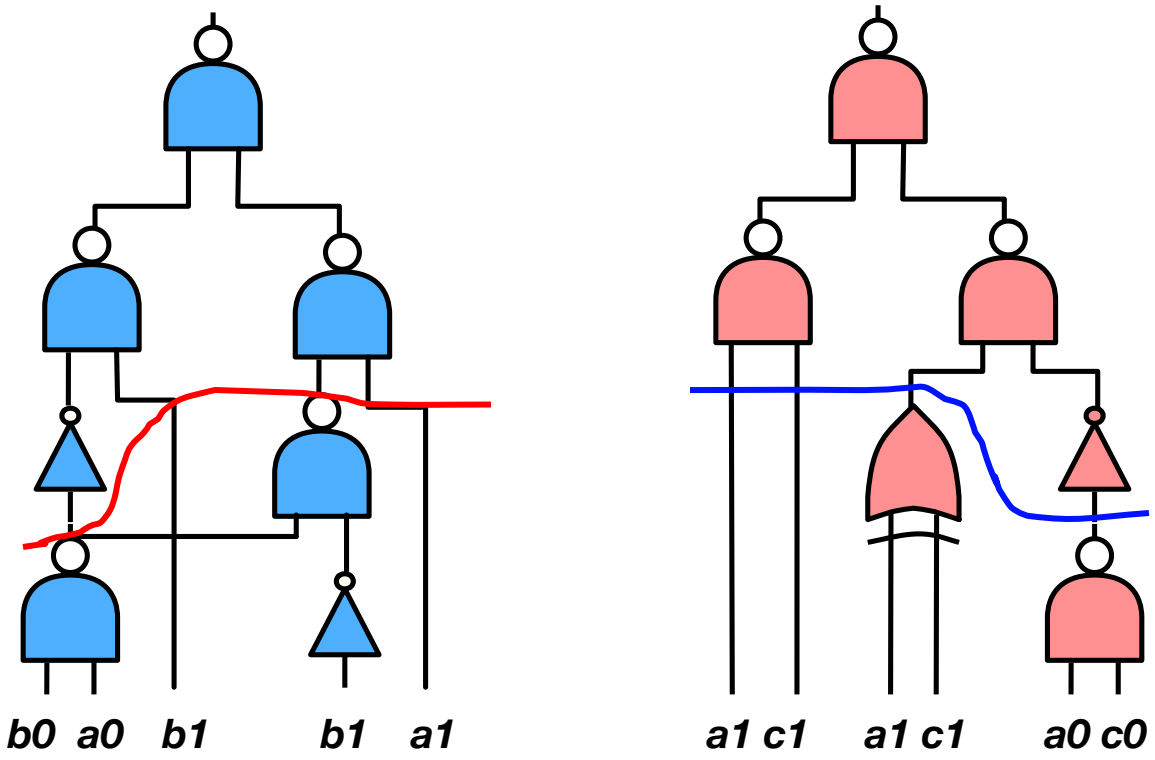
Multiplexer Relocation

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



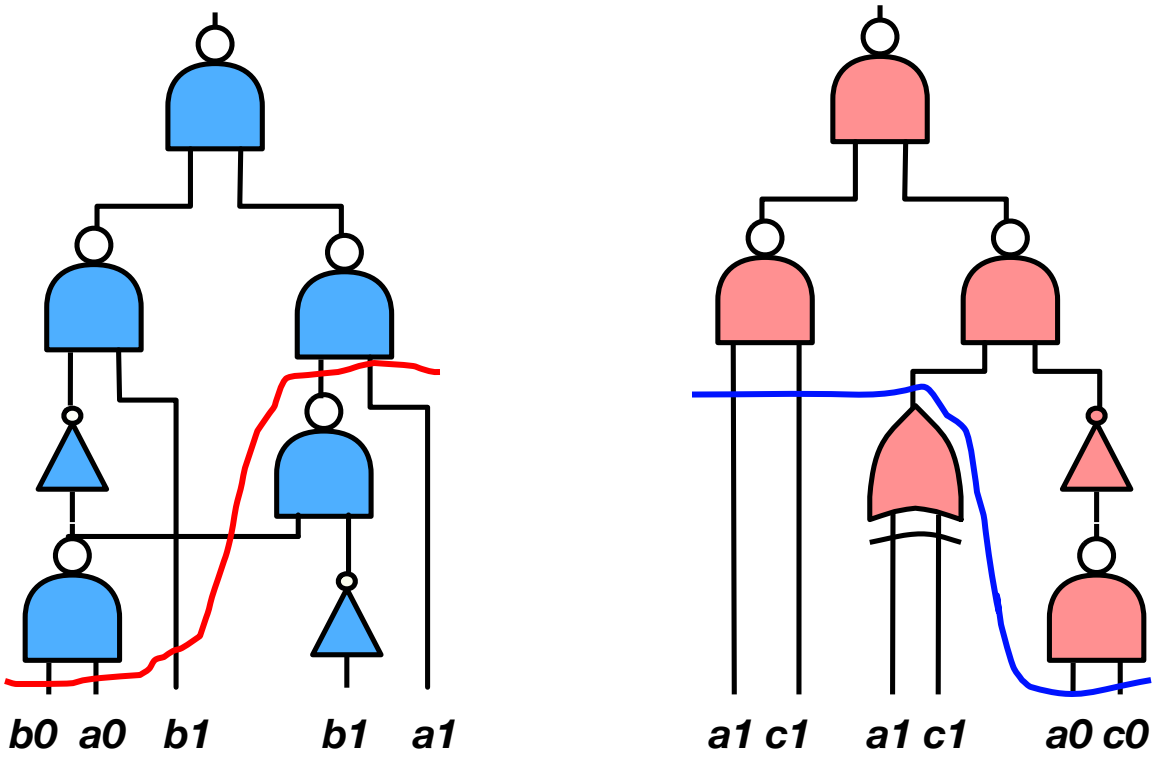
Multiplexer Relocation

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



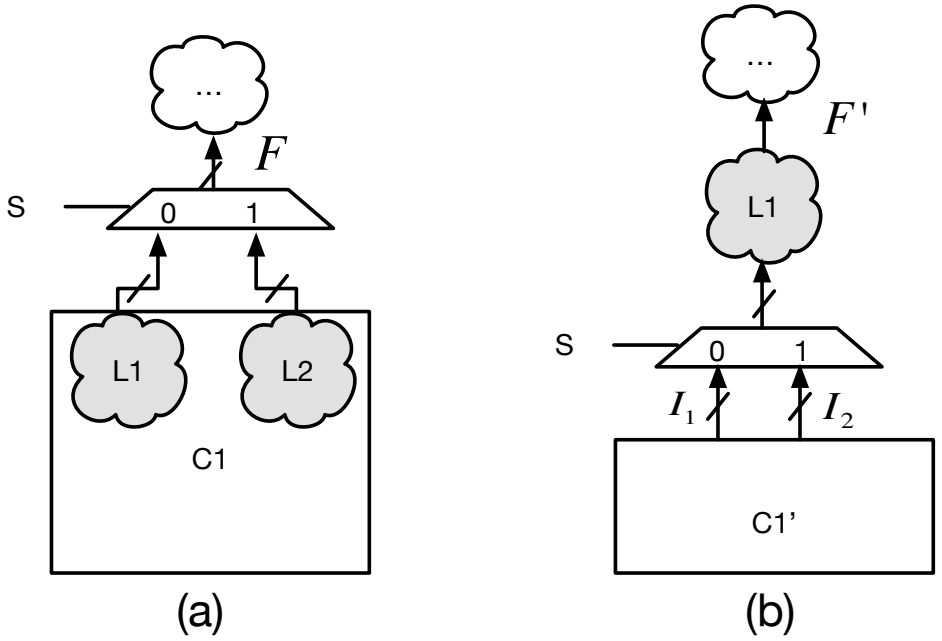
Multiplexer Relocation

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



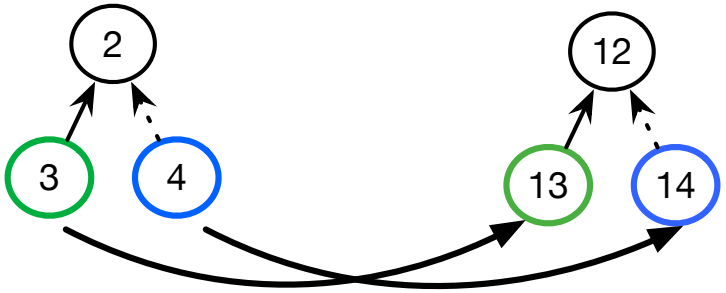
Multiplexer Relocation

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

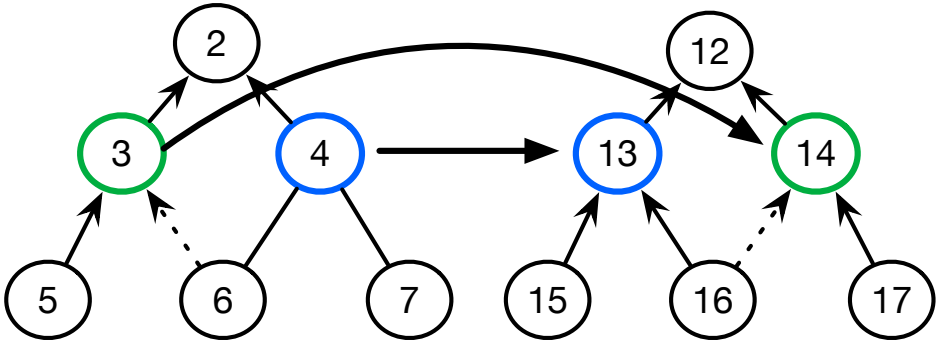


AIG-based Isomorphism

- **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]**

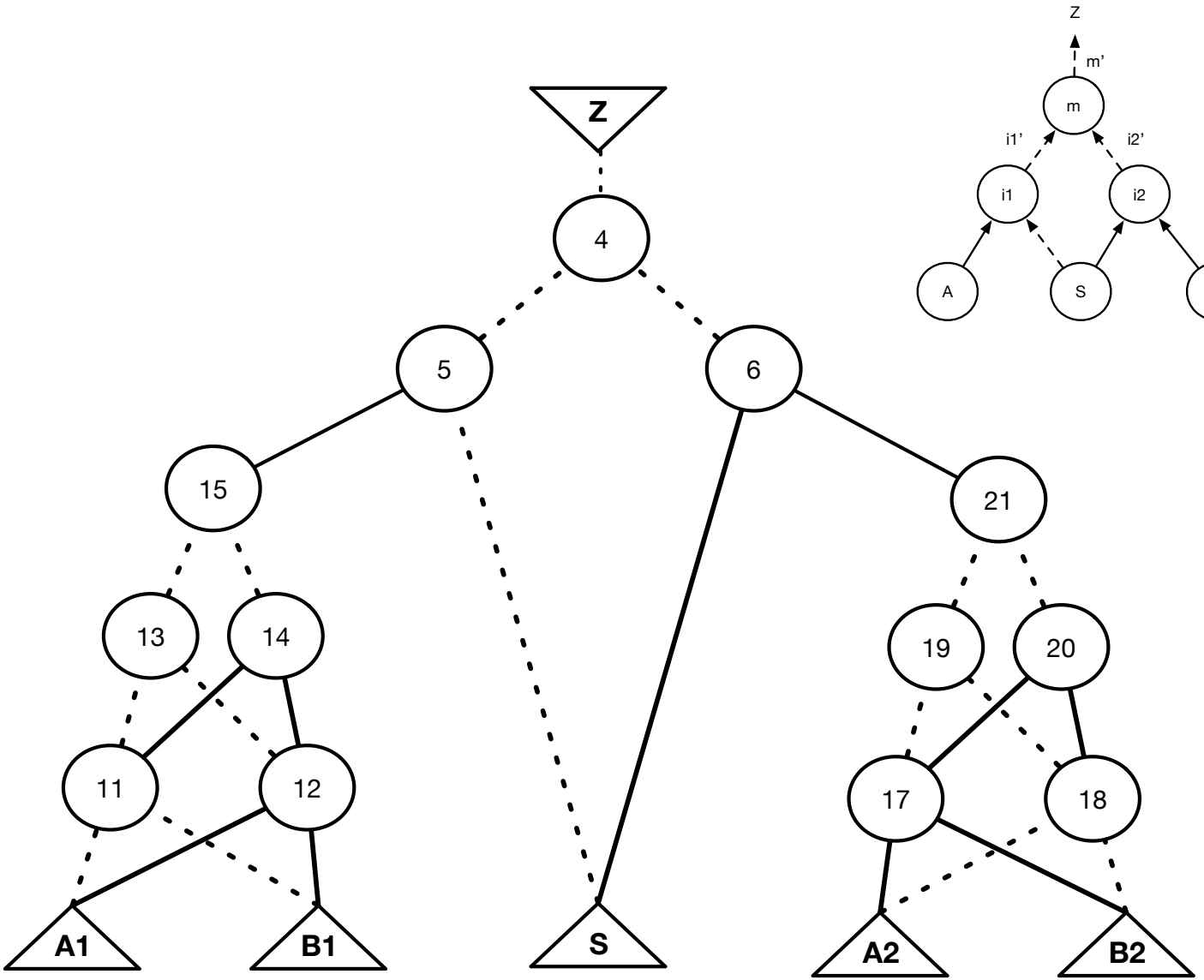


(a) Type 1



(b) Type 2

Demo



$$Z = A \cdot \overline{S} + B \cdot S$$

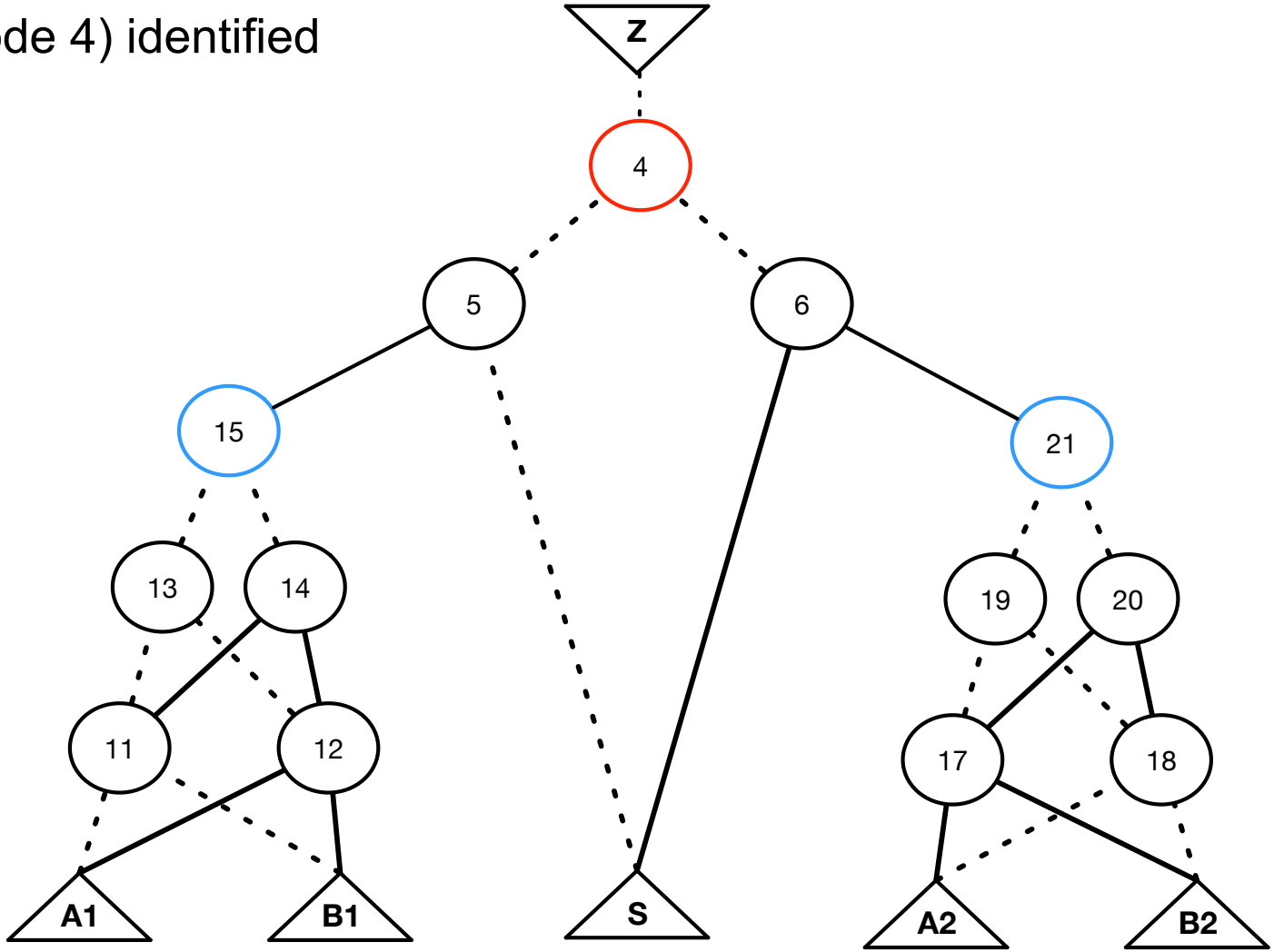
$$m' = \overline{m} = \overline{\overline{i_1} \cdot \overline{i_2}} = i_1 + i_2$$

$$m = \overline{i_1} \cdot \overline{i_2}$$

$$i_1 = A \cdot \overline{S} \quad i2 = B \cdot S$$

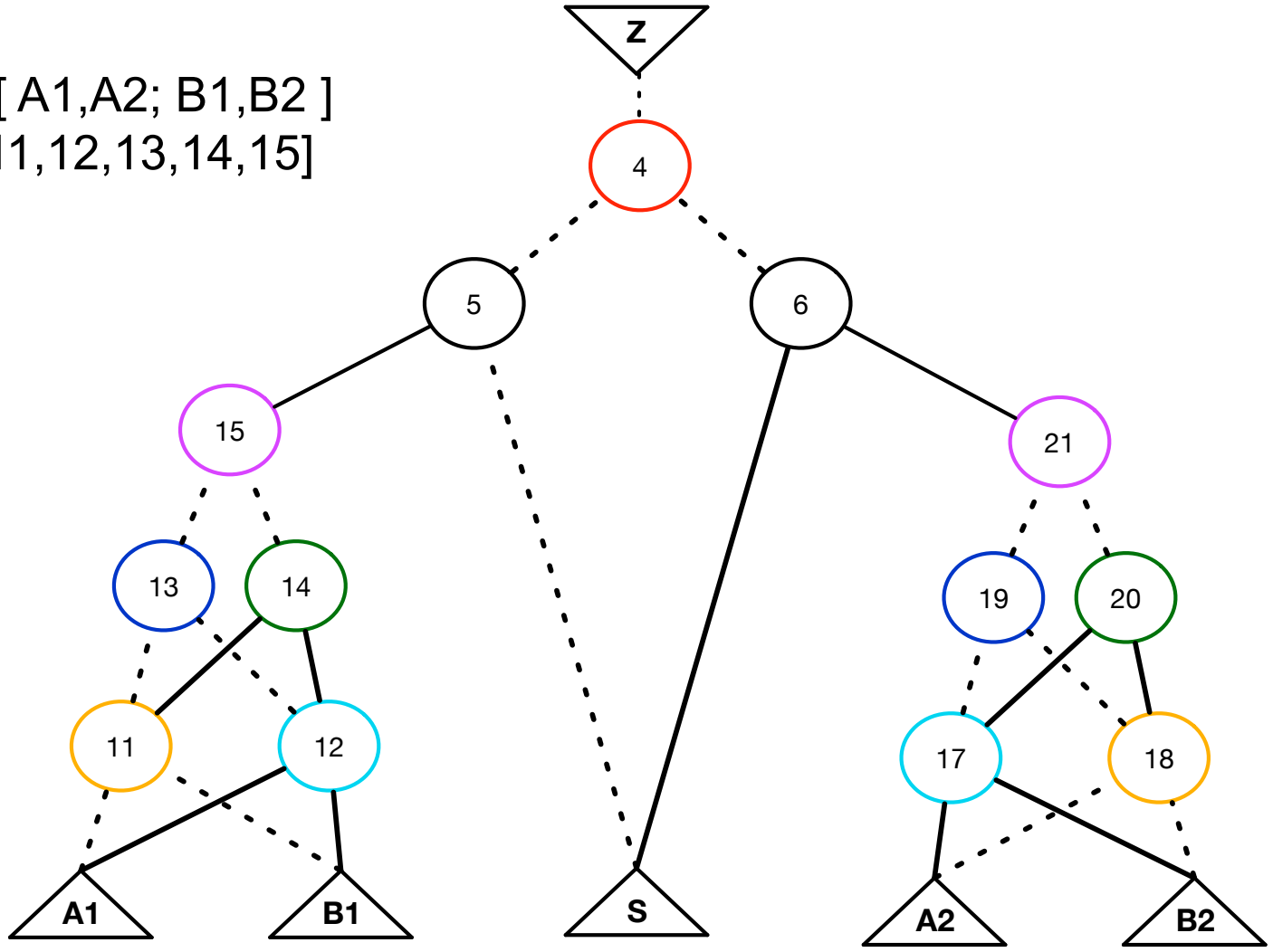
Demo

MUX (node 4) identified



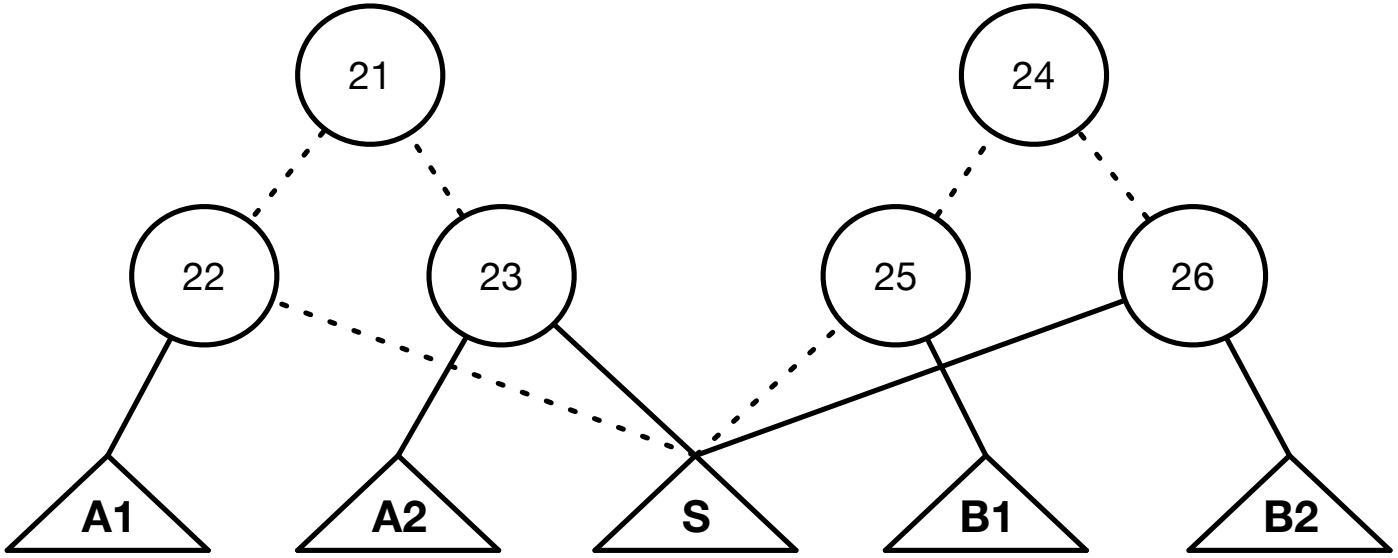
Demo

select: s
Bound : [A1,A2; B1,B2]
Logic : [11,12,13,14,15]

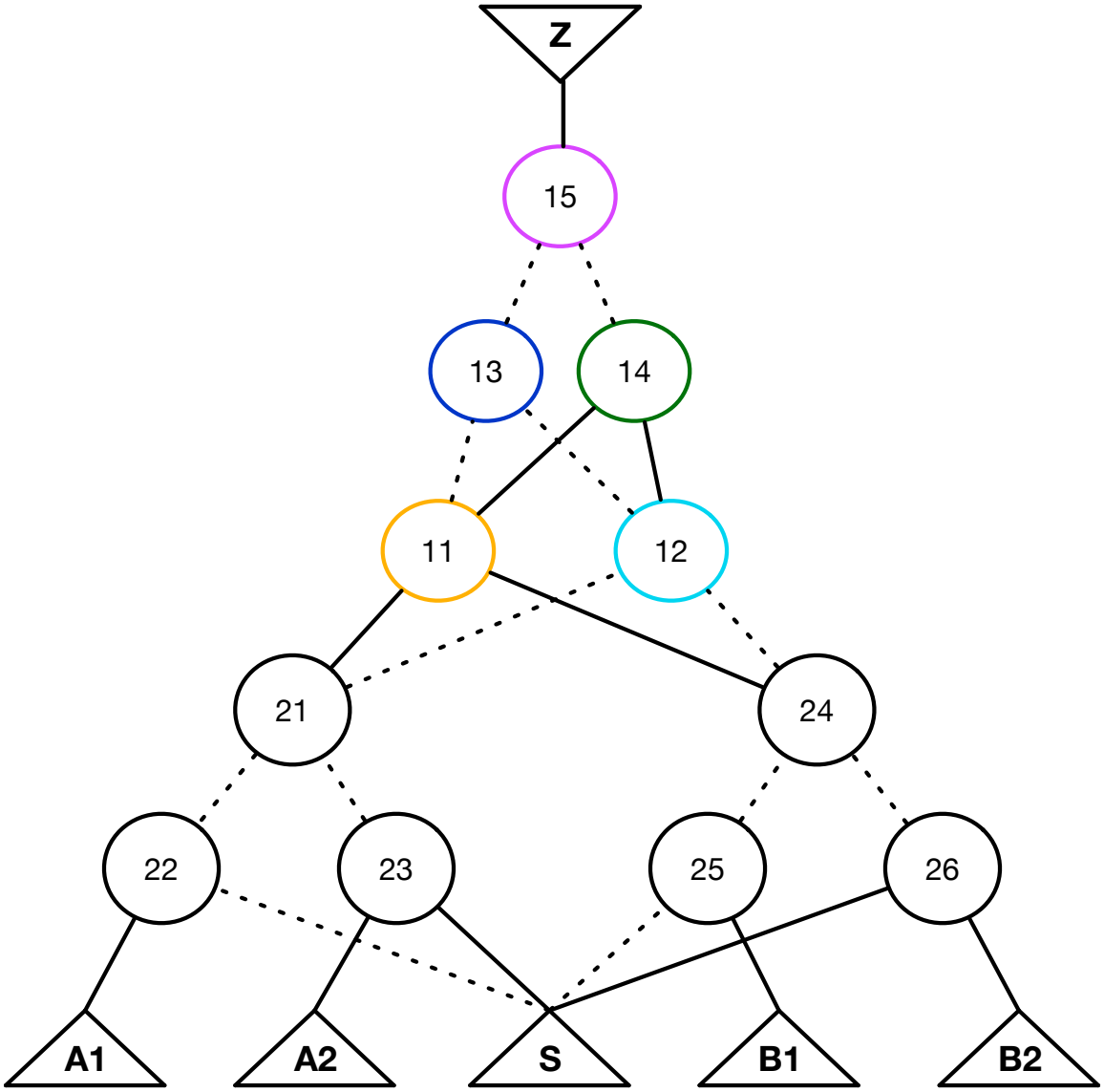


Demo

Two MUXs (21, 24) created

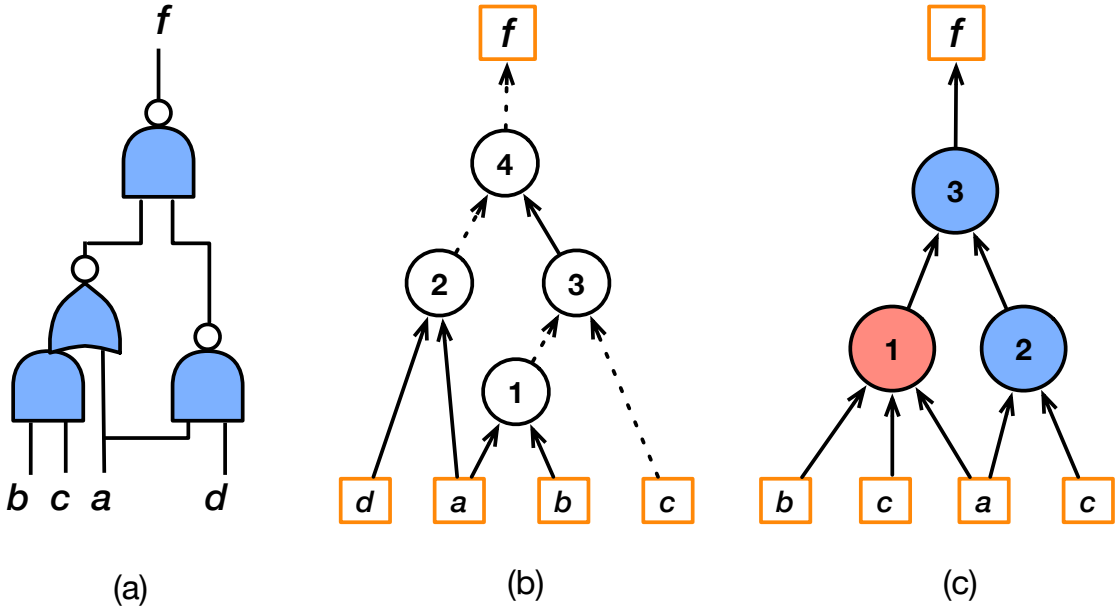


Demo



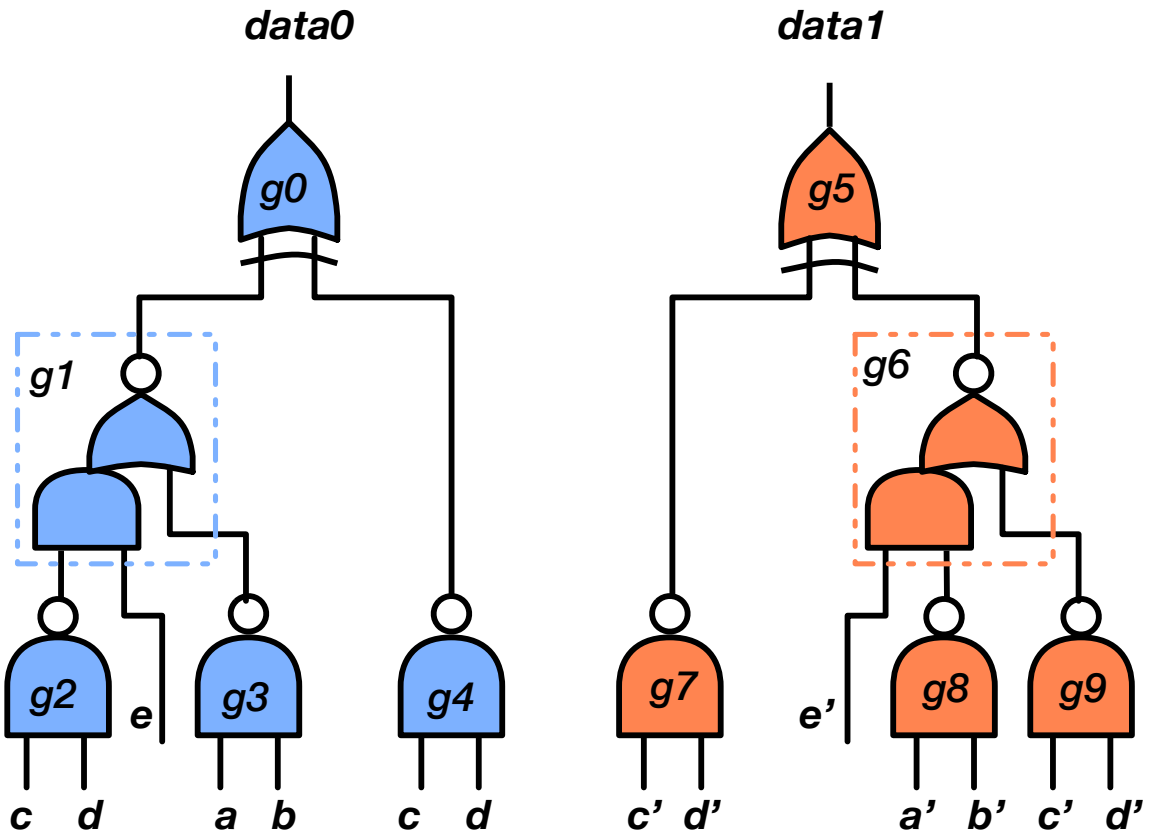
STD-based Isomorphism

- **Limitations of AIG isomorphism**
  - Complexity is high for large common logic
- **Standard-cell DAG**
  - More types of nodes
  - Graph is unweighted
  - More compact



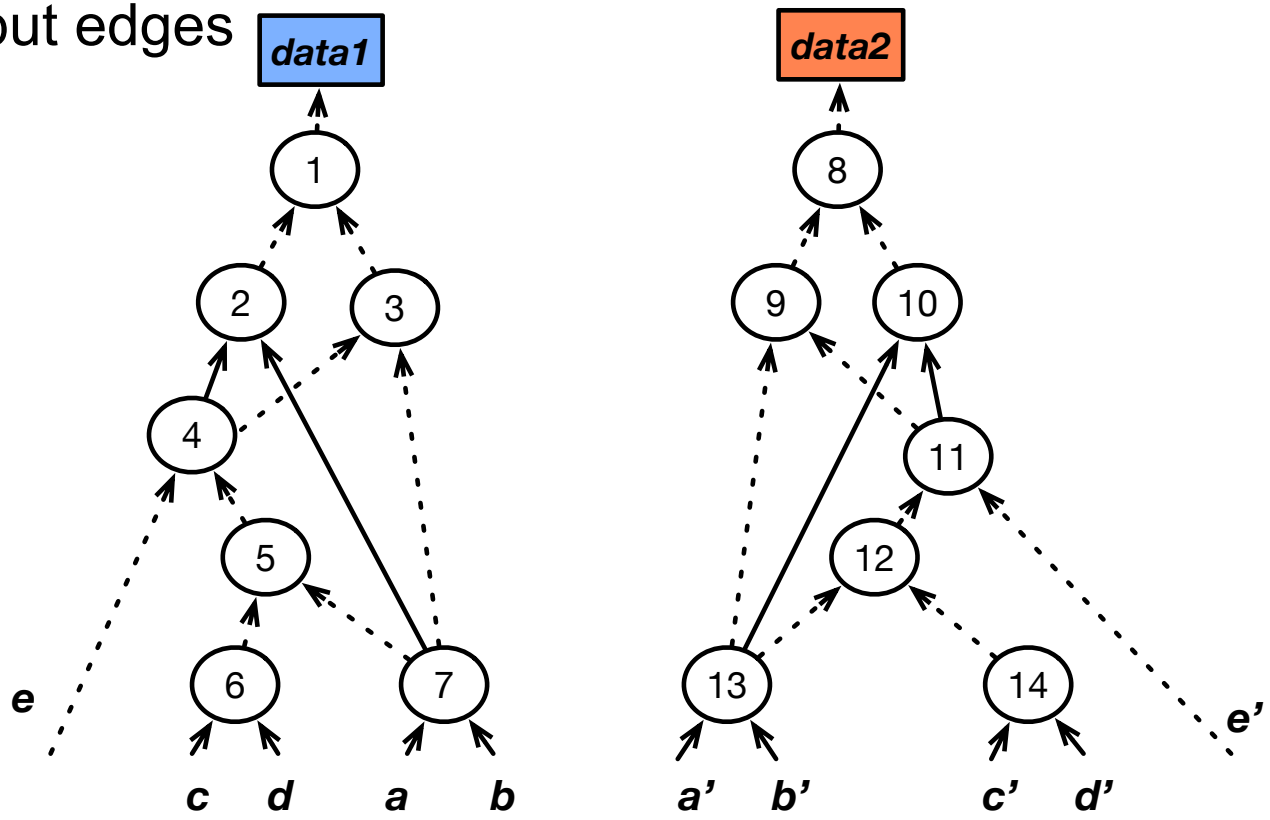
STD-based Isomorphism

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



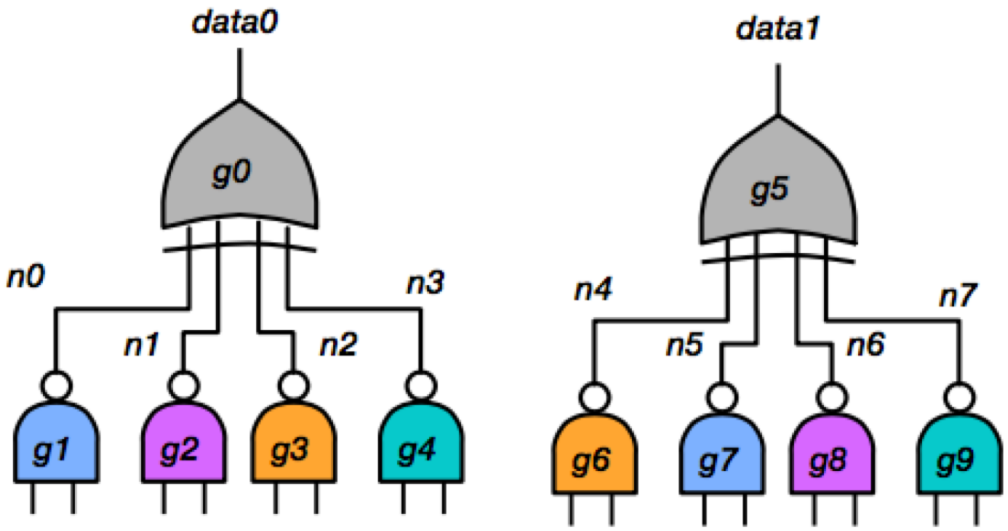
STD-based Isomorphism

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



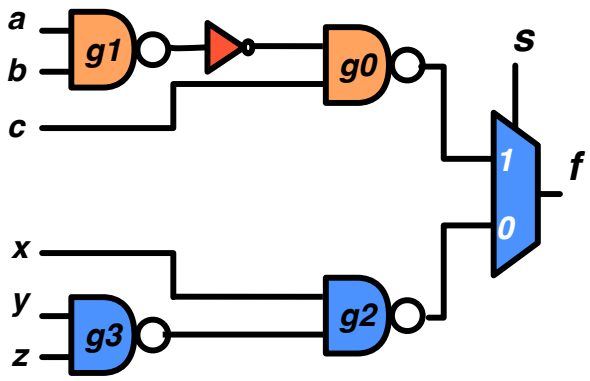
Improve CSL identification

- **Side fanout information**
  - Nodes with same number of fanouts
    - 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

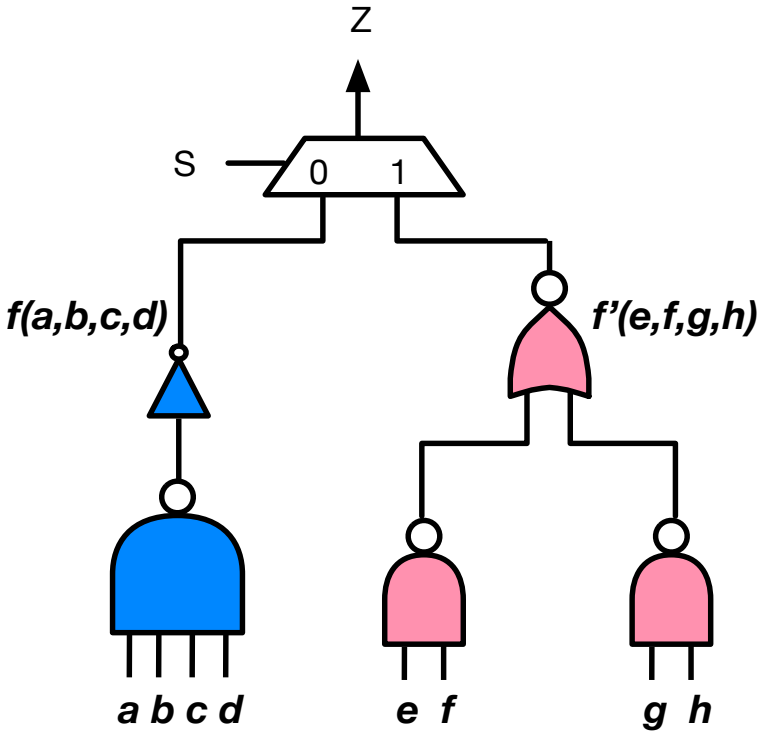


(a)

Beyond Graph Isomorphism

- **Appx – Case 2**

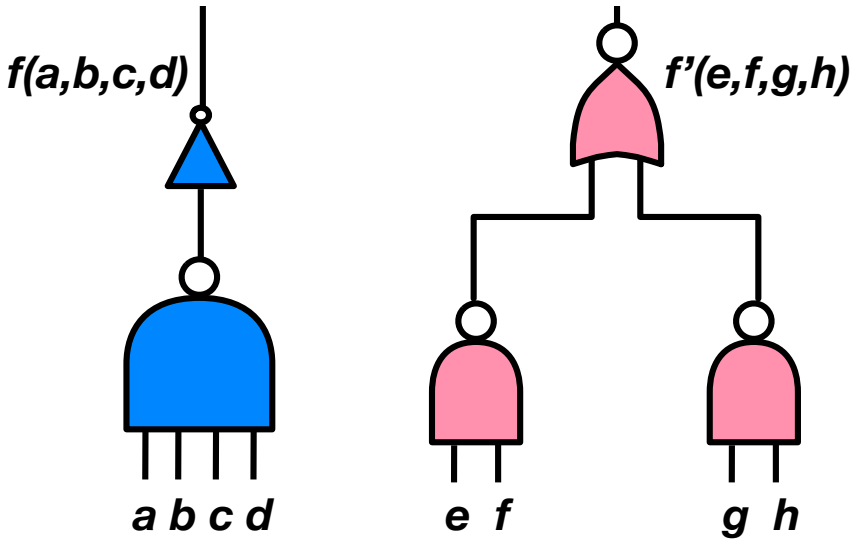
$$f(a,b,c,d) = \neg(a \wedge b \wedge c \wedge d)$$
$$f'(e,f,g,h) = \neg(\,(e \wedge f) \vee (g \wedge h)\,)$$



Beyond Graph Isomorphism

- **Appx – Case 2**

$$f(a,b,c,d) = \neg(a \wedge b \wedge c \wedge d)$$
$$f'(e,f,g,h) = \neg(\,(e \wedge f) \vee (g \wedge h)\,)$$

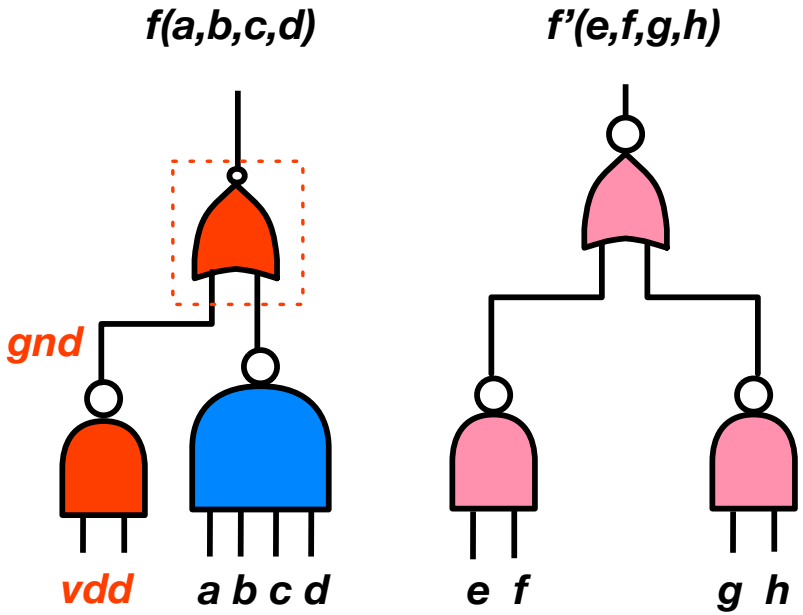


Beyond Graph Isomorphism

- **Appx – Case 2**

$$f(a,b,c,d) = \neg(a \wedge b \wedge c \wedge d)$$
$$f'(e,f,g,h) = \neg((e \wedge f) \vee (g \wedge h))$$

  - *inv(n) -> nor(n,1)*

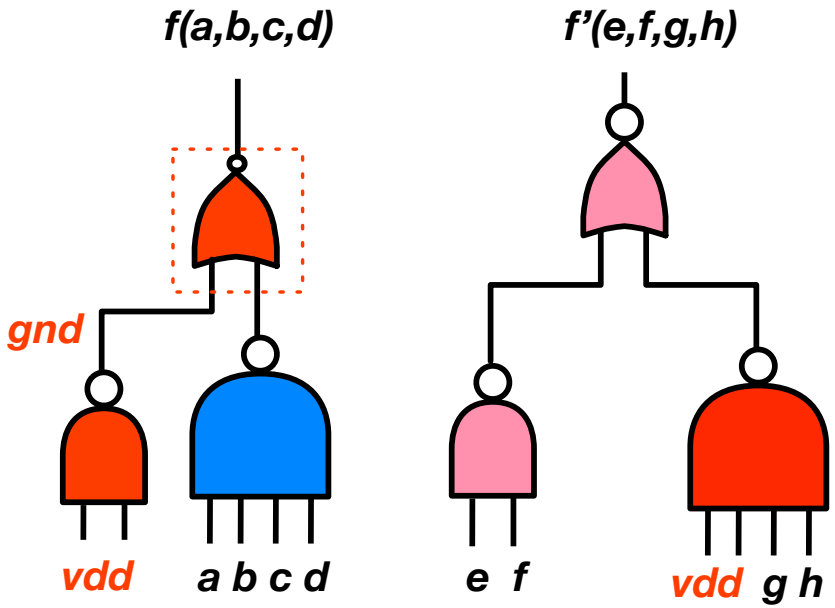


Beyond Graph Isomorphism

- **Appx – Case 2**

$$f(a,b,c,d) = \neg(a \wedge b \wedge c \wedge d)$$
$$f'(e,f,g,h) = \neg((e \wedge f) \vee (g \wedge h))$$

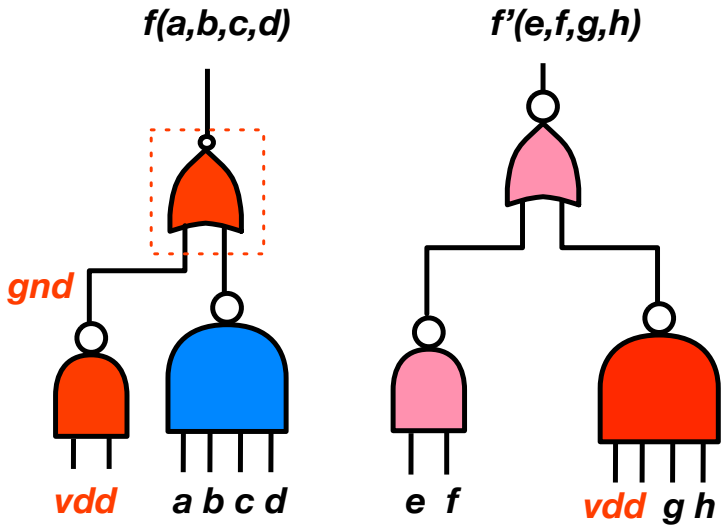
  - *inv(n) -> nor(n,1), nand(c,d) -> nand(1,1,c,d)*



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).*

$$\exists F(z)\ (F(x,0,1) = P(x)) \wedge (F(y,0,1) = Q(y))$$



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

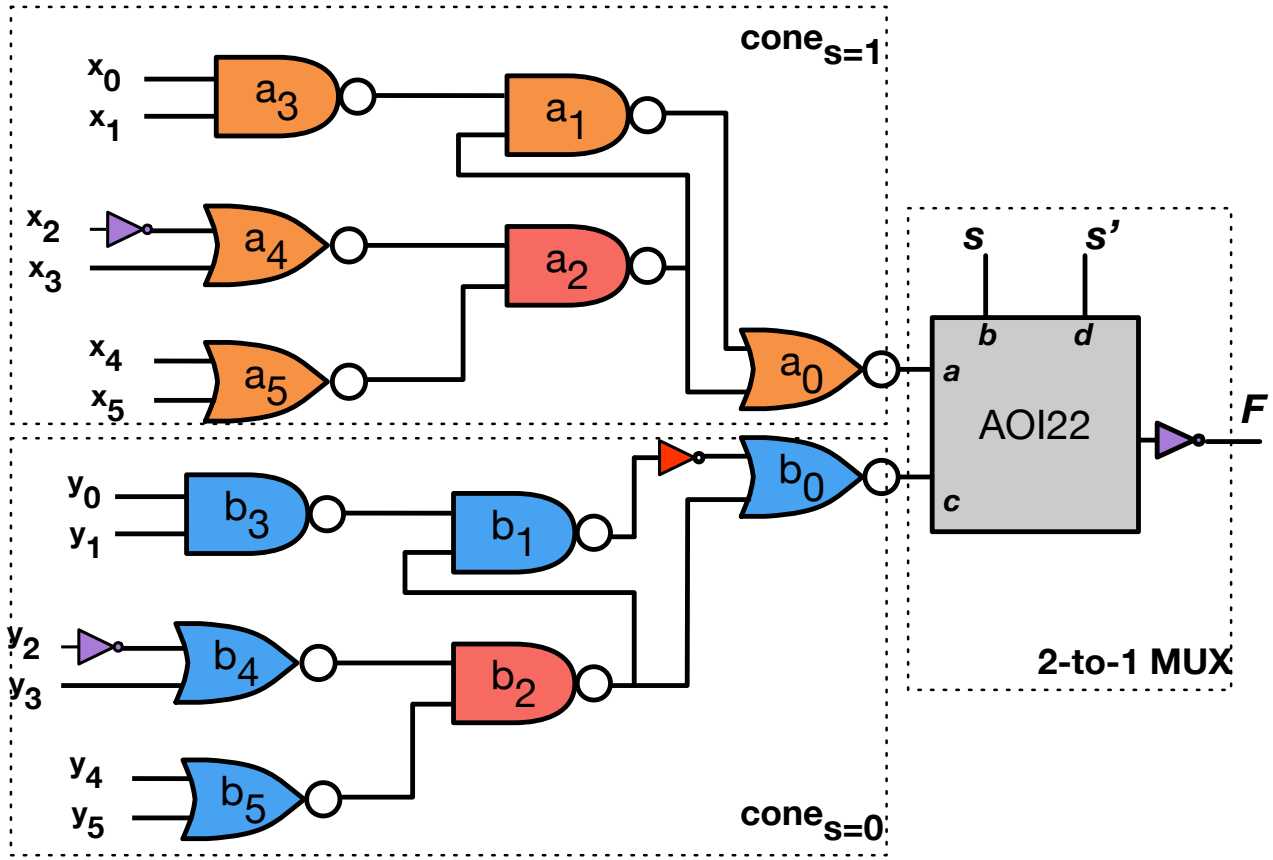
Single_Mux_Relocate(C)
1: B = RelocationBoundray(PO)
2: $C \leftarrow$ relocate multiplexer to level B, w/o considering inverters
3: P = inv2xorPosition(PO, B)
4: $C \leftarrow$ insert XORs to P based on its location
5: **return** C

RelocationBoundray(PO)
1: $m \leftarrow levels(PO) - 1$; inverter is considered as 0 level.
2: **while** $m \geq 0$ **do**
3: $L0_m \leftarrow$ the gates in $(s = 0)$ logic at m level
4: $L1_m \leftarrow$ the gates in $(s = 1)$ logic at m level
5: **if** $(L0'_m, L1'_m) \leftarrow$ uniqueFanoutPairs($L0_m, L1_m$) **then**
6: $L0_{m-1}, L1_{m-1} \leftarrow$ uniqueFanoutPairs($L0_m, L1_m$)
7: $L0_m \leftarrow L0_m \cap L0'(m)$, $L1_m \leftarrow L1_m \cap L1'(m)$
8: $L0_{m-1}, L1_{m-1}$ += isomorphsim($L0_m, L1_m$)
9: **else**
10: **if** isomorphsim($L0_m, L1_m$) **then**
11: $L0_{m+1}, L1_{m+1} \leftarrow$ 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: $P_0 \leftarrow$ the positions of all inverts till *boundary* level
2: $P_1 \leftarrow$ the positions of all inverts till *boundary* level
3: **return** $P_0 \cap P_1$

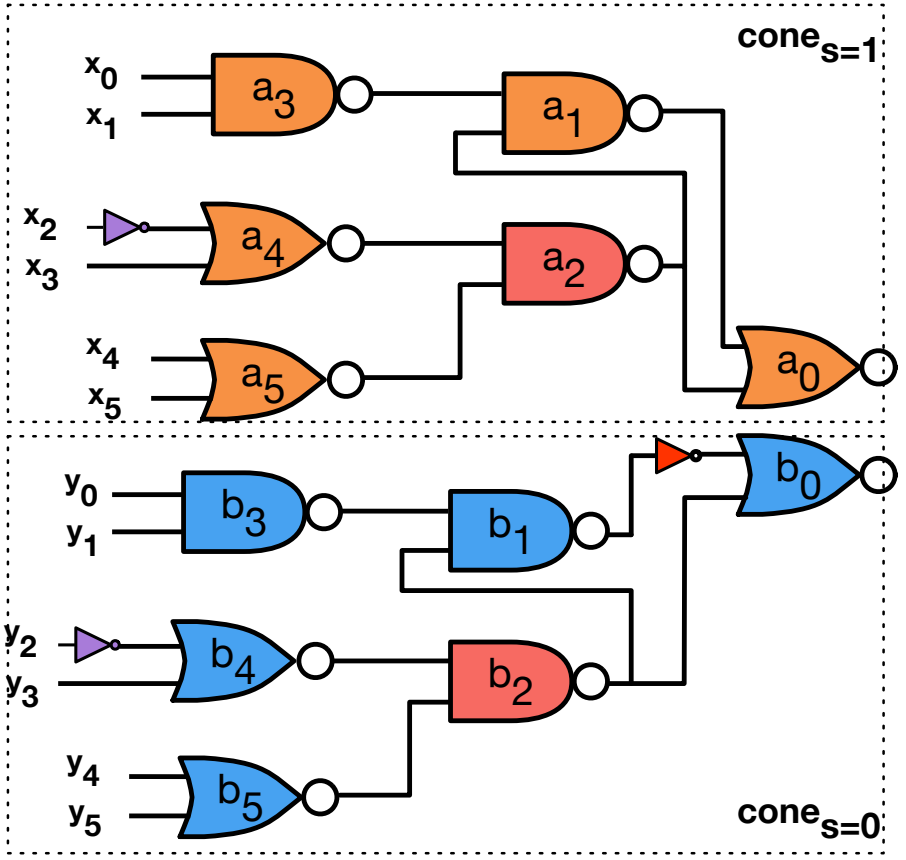
Example

- **MUX2**
  - AOI22+INV



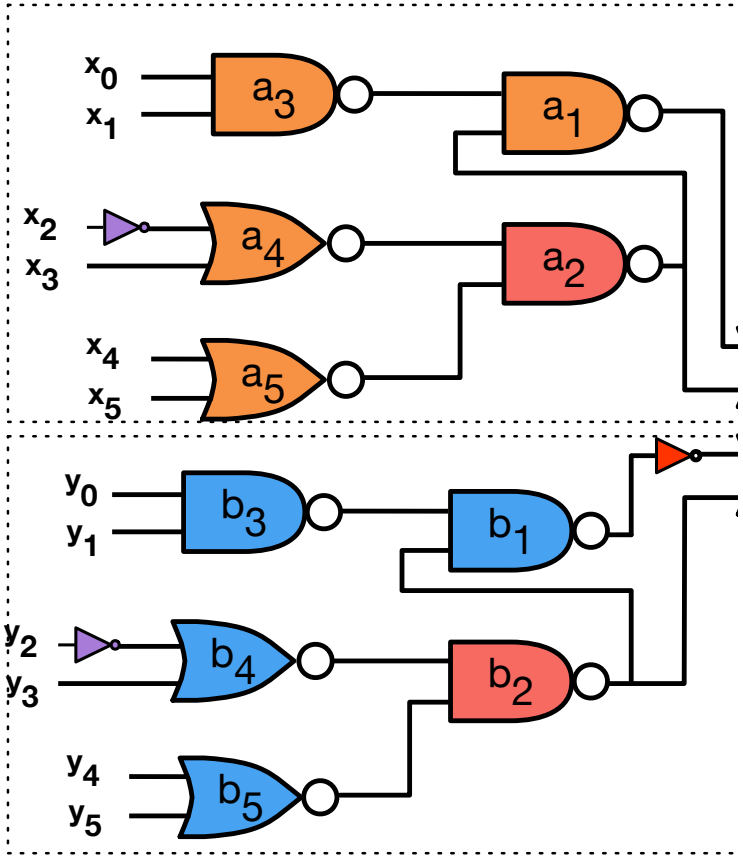
Example

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



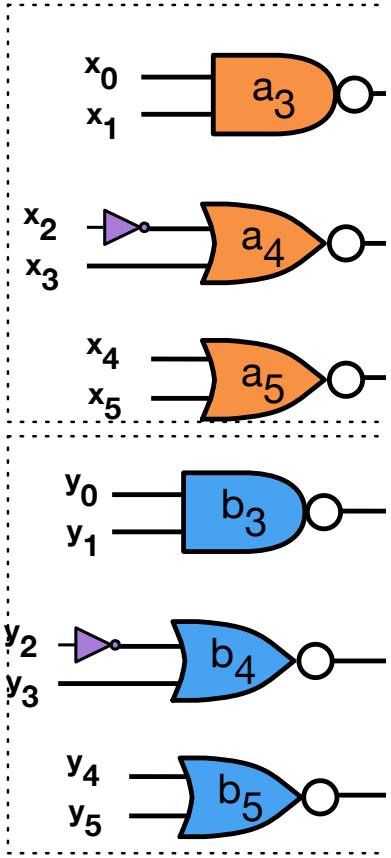
Example

- **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

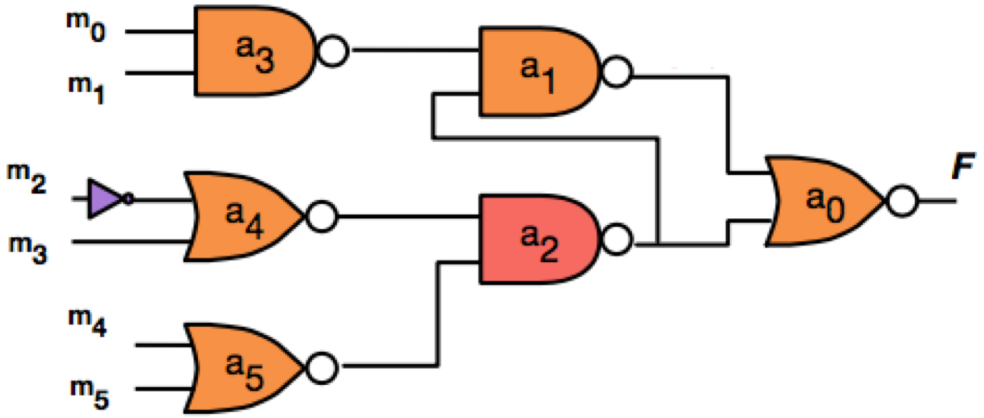
- **Level0**
  - $\{a_0\}_{s=1}$ - $\{b_0\}_{s=0}$
- **Level1**
  - $\{a_1,a_2\}_{s=1}$ - $\{b_1,b_2\}_{s=0}$
    - a_2,b_2 has two fanouts
  - INV skipped $\{0,0,1\}_{s=1}$ - $\{0,0,0\}_{s=0}$
- **Level2**
  - $\{a_3,a_4,a_5\}_{s=1}$ - $\{b_3,b_4,b_5\}_{s=0}$
    - [0,0,**1**,0,0….,0,0,**1**,0,0,0]
    - [0,0,**0**,0,0….,0,0,**1**,0,0,0]



Example

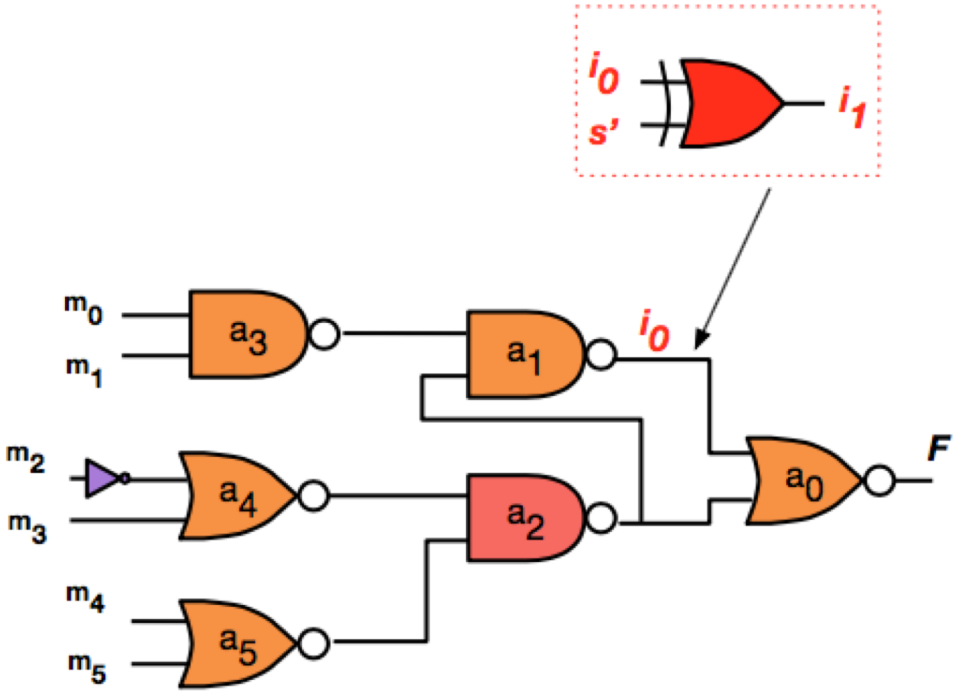
$$m_i = x_i \bar{s} + y_i s$$

- **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….,0,0,**1**,0,0,0]



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 $\{0,0,1\}_{s=1}$
- **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….,0,0,**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

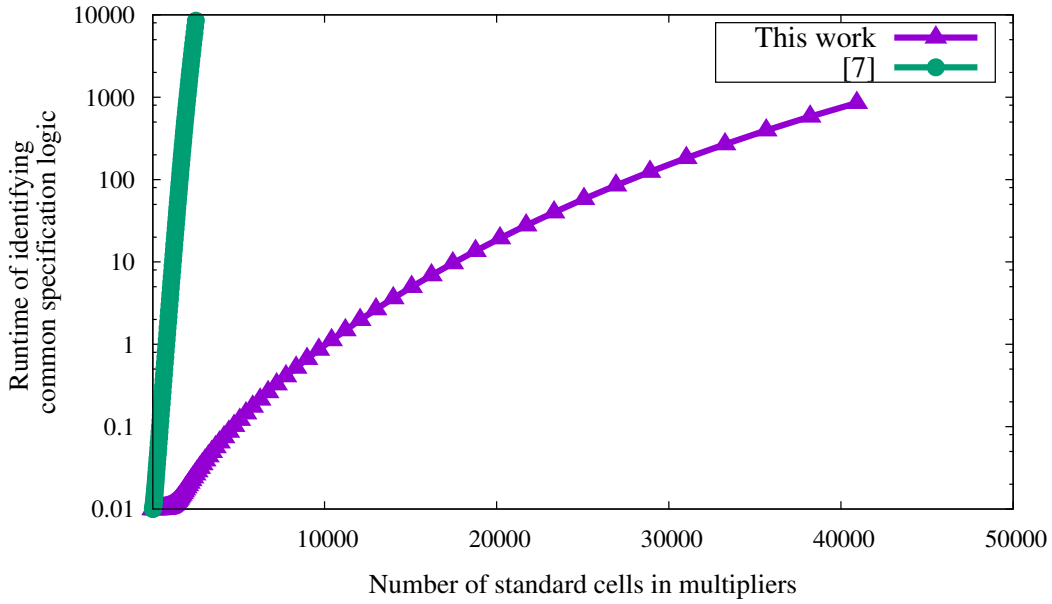
Results

- **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	Area	Delay	Area	Delay
ibm1	3622	216.45	2223	255.81	4587	295.16	2235	255.81
ibm2	5454	314.84	3361	354.19	6879	432.90	3366	383.71
ibm3	9115	501.77	5526	501.77	11463	688.71	5610	541.13
ibm4	12782	678.87	7874	649.35	16047	924.84	7854	688.71
ibm5	18323	924.84	11121	787.10	22923	1200.32	12342	875.65
ibm6	27435	1170.81	16843	983.87	34383	1505.32	16803	1023.23
ibm7	31069	1288.87	19083	1082.26	38967	1603.71	19074	1082.26
	1	1	**0.613**	**0.970**	1	1	**0.487**	**0.767**

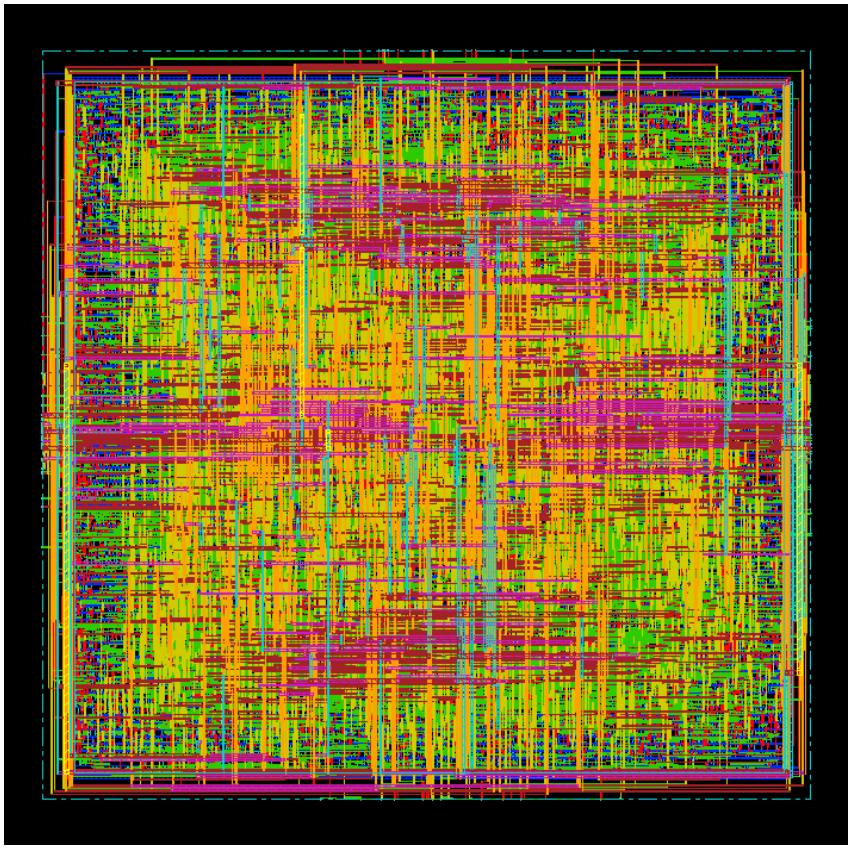
Results

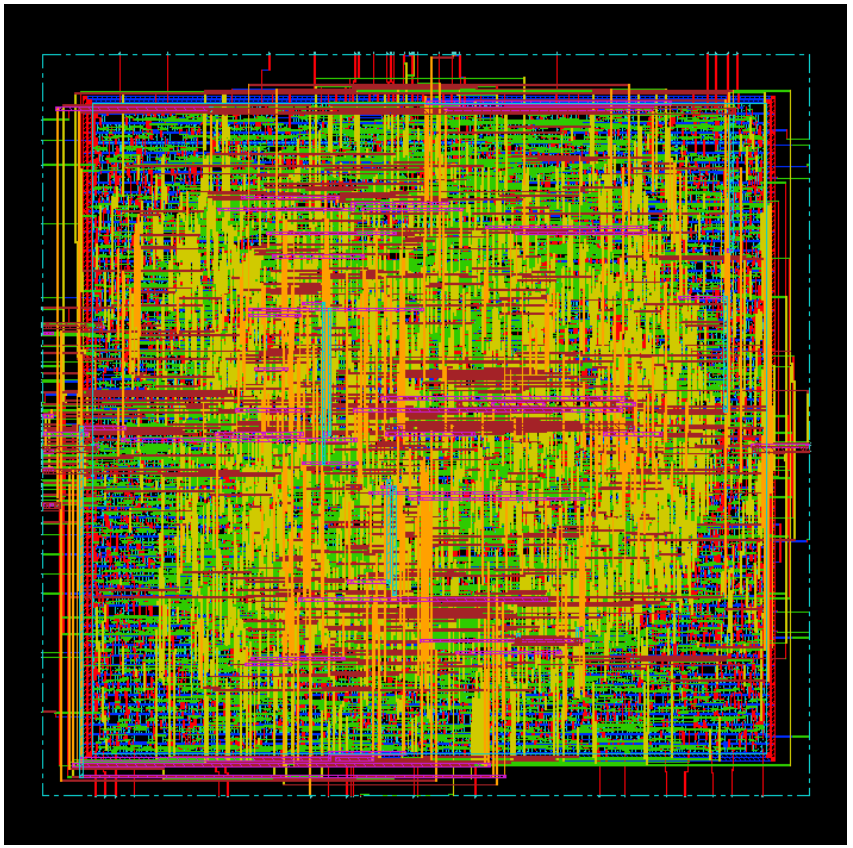
- **Runtime: AIG vs. STD graph**
  - Implemented in C++, within IBM synthesis flow
  - Xeon CPU 7560 v6 x32, 4TB Memory
  - Benchmark: sel?mult1:mult2



Results

- **Physical design evaluation**



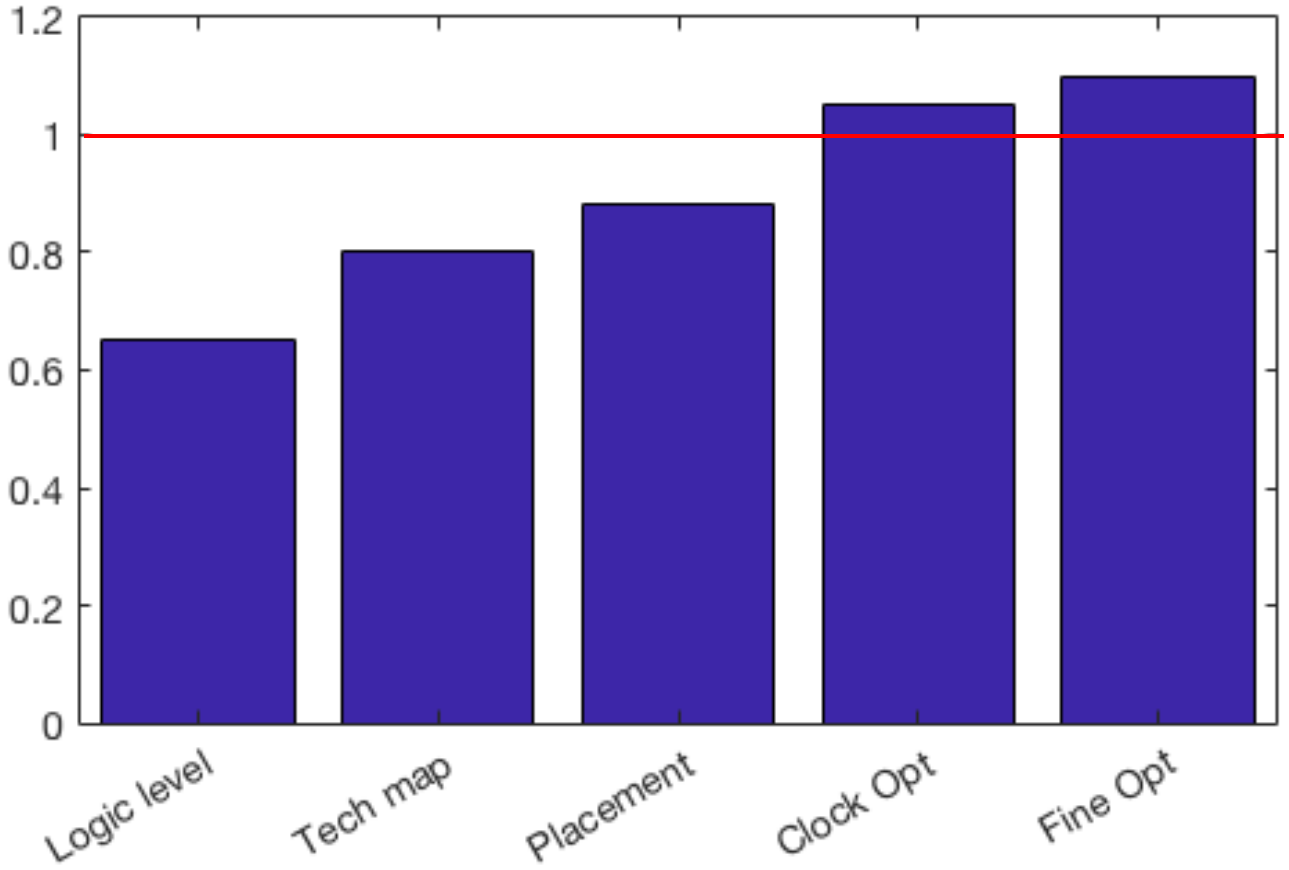


What next?

- **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?


1.2
1
0.8
0.6
0.4
0.2
0
Logic level
Tech map
Placement
Clock Opt
Fine Opt

Thank you !