

# **A Practical and Efficient Method for Compare-Point Matching**

---

---

Demos Anastasakis, Robert Damiano,

Tony Ma, Ted Stanion

June 12, 2002

**Synopsys**

# Outline

---

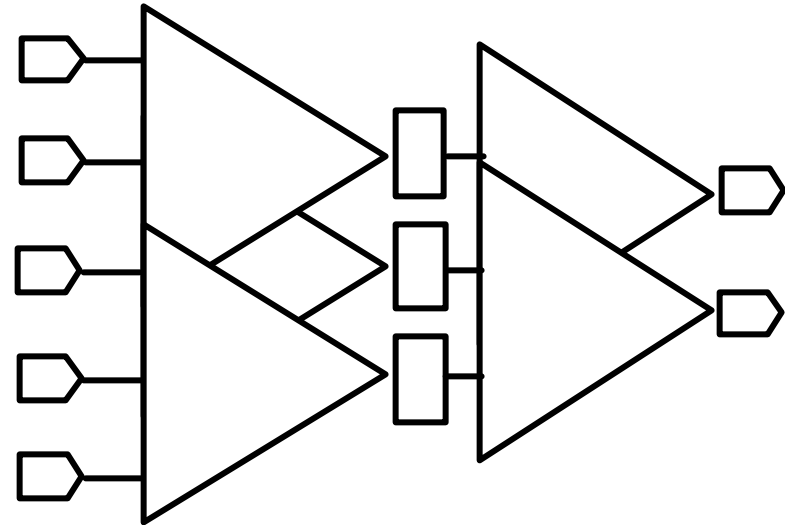
- definitions
- problem statement
- matching methods
- our approach
- search model
- don't cares
- results
- future work

# Definitions

---

**A sequential design can be segmented into combinational cones.**

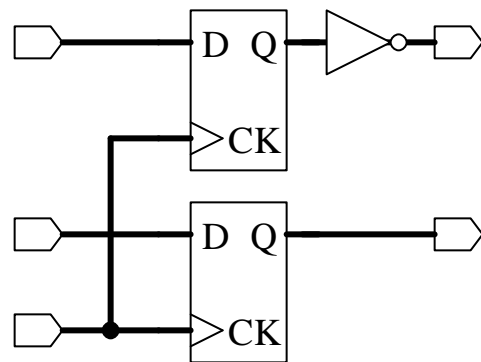
- cpoints are the outputs of these cones
  - ⊙ primary outputs
  - ⊙ latch or black box inputs
- ipoints are the inputs of these cones
  - ⊙ primary inputs
  - ⊙ latch or black box outputs



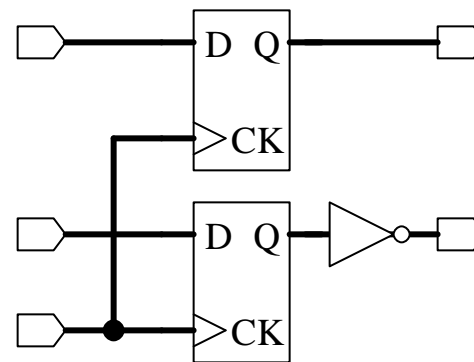
# Problem Statement

---

- For combinational equivalence checkers between designs A and B, it is important to identify and match cpoints and ipoints in the two sequential designs to be verified. We will concentrate on the cpoints.
- A priori, given  $n$  outputs and  $m$  latches ( $m+n$  cpoints) in each circuit, there are  $n! \times m!$  possibilities.



A



B

# Matching methods

---

- Non-Functional
- Functional

# Matching methods - Non-Functional

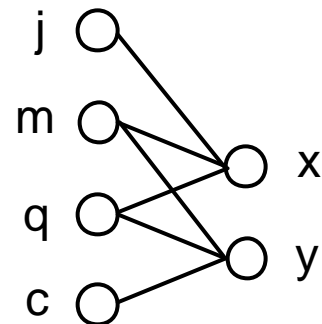
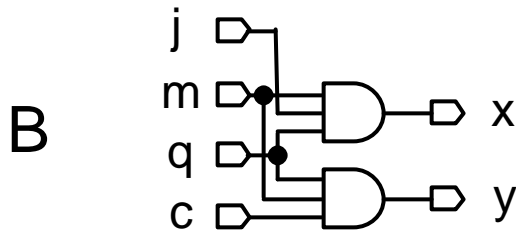
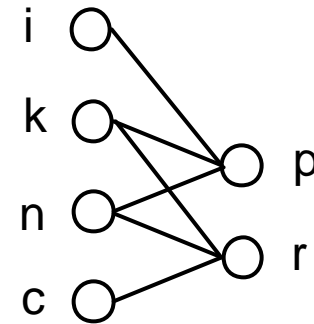
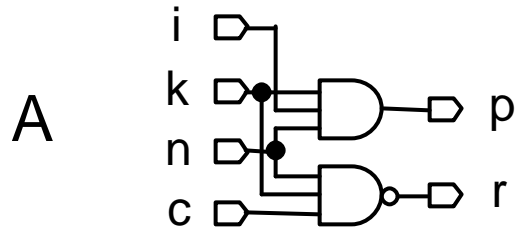
---

- Name-based analysis
  - Methods explore similarity in names of cpoints, and nets attached to them.
    - exact : names are identical
    - heuristic: (ex. "A/b/reg[0]" vs. "B/b/reg\_0\_")
  - Names Could be mangled beyond recognition by various tools both commercial and in-house
  - Probably the most dangerous matching method but also the most efficient

# Matching methods - Non-Functional

---

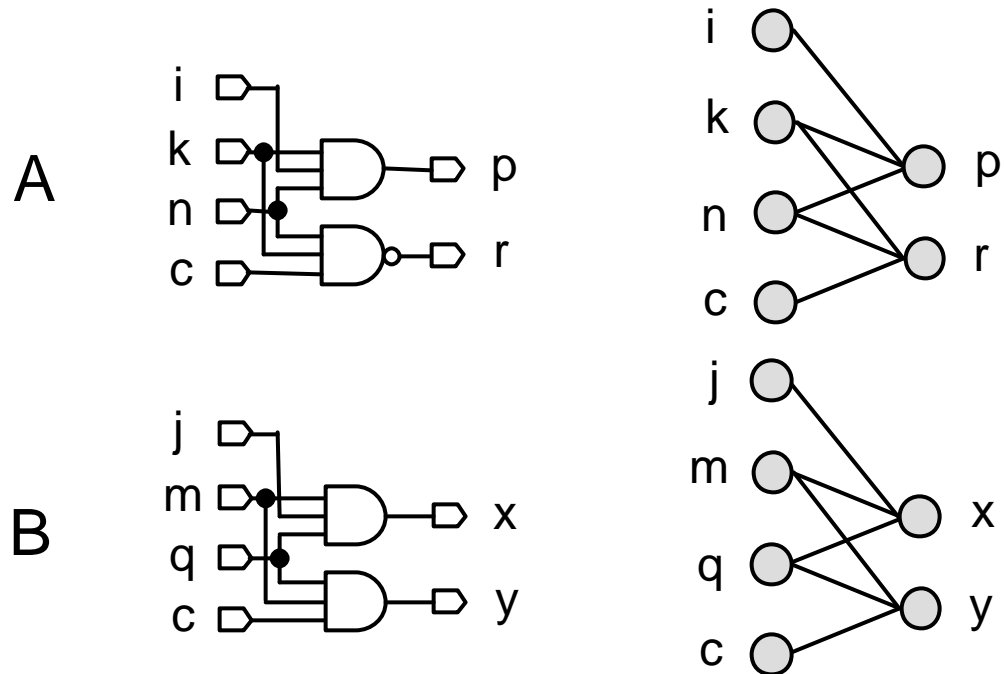
- Structural analysis partitioning criteria (cont.)
  - Identifies input/output dependencies of each unmatched object.



# Matching methods - Non-Functional

---

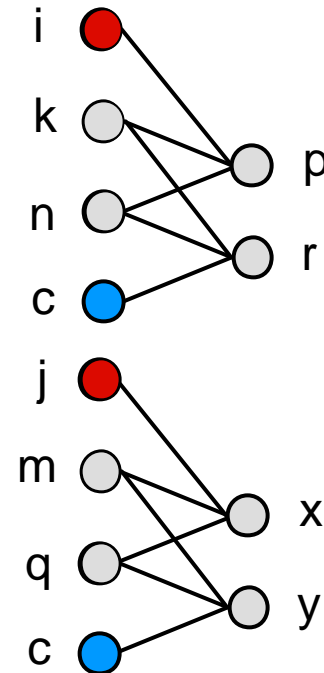
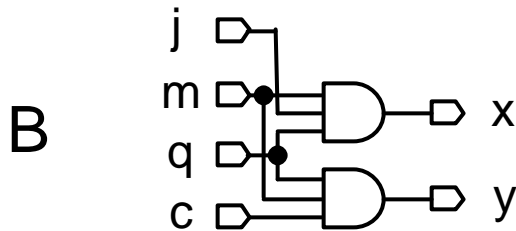
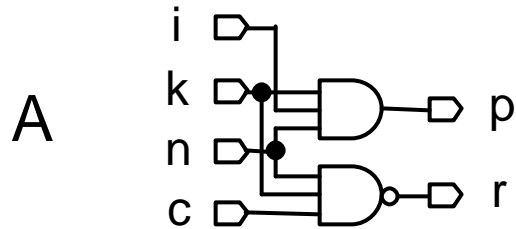
- Structural analysis partitioning criteria (cont.)
  - Identifies input/output dependencies of each unmatched object.



# Matching methods - Non-Functional

---

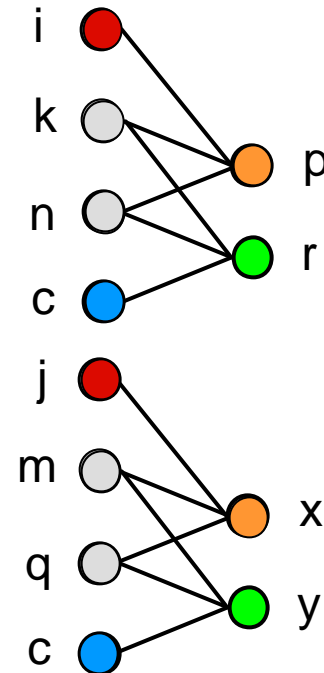
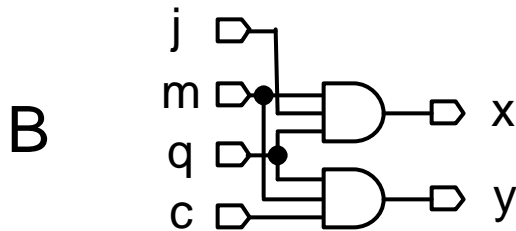
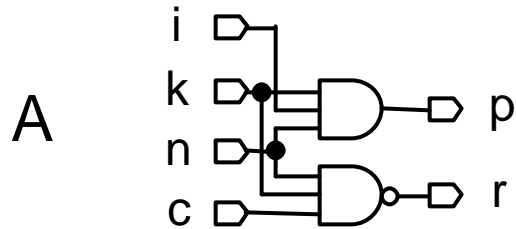
- Structural analysis partitioning criteria (cont.)
  - Identifies input/output dependencies of each unmatched object.



# Matching methods - Non-Functional

---

- Structural analysis partitioning criteria (cont.)
  - Identifies input/output dependencies of each unmatched object.

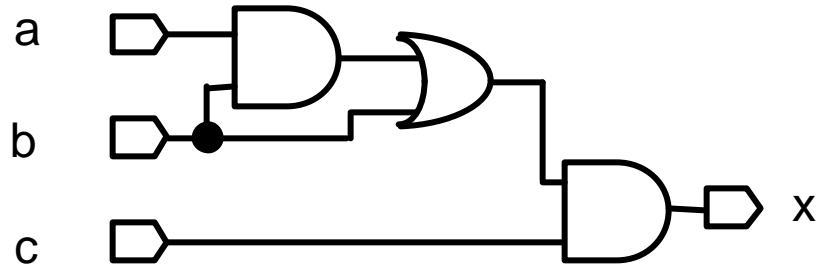


# Matching methods - Non-Functional

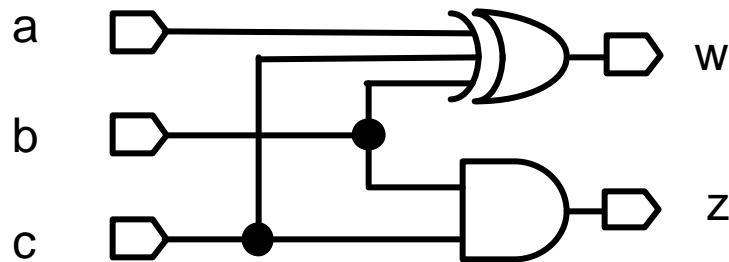
---

- Structural analysis (cont.)
  - Redundancies can be misleading

A



B



# Matching methods - Functional

---

- **Functional analysis**

- Attempts to find a set of functional characteristics (signatures) for each unmatched object that distinguishes it from all other unmatched objects.

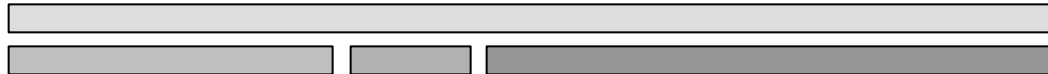


# Matching methods - Functional

---

- **Functional analysis**

- ⊙ Attempts to find a set of functional characteristics (signatures) for each unmatched object that distinguishes it from all other unmatched objects.
- ⊙ Process is iterative; the initial set of unmatched objects is continuously segmented into subgroups until no more progress can be made.

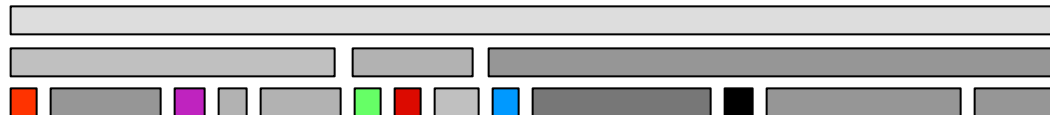


# Matching methods - Functional

---

- **Functional analysis**

- Attempts to find a set of functional characteristics (signatures) for each unmatched object that distinguishes it from all other unmatched objects.
- Process is iterative; the initial set of unmatched objects is continuously segmented into subgroups until no more progress can be made.
- A match is achieved when an object from the first design is in the same group with an object in the second design.

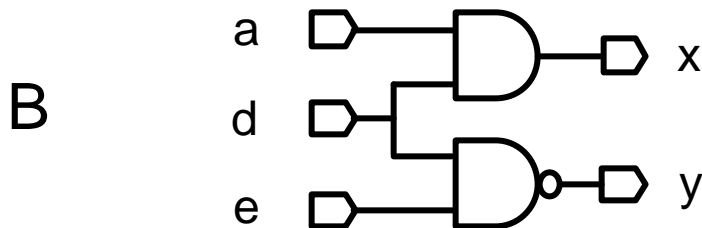
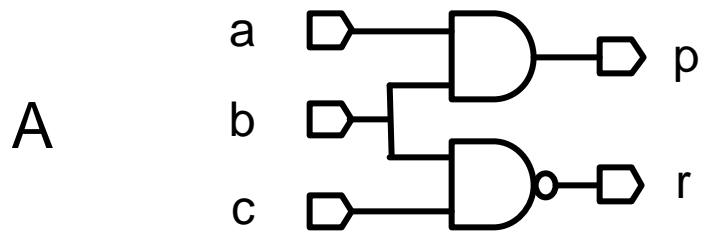


# Matching methods - Functional

---

- **Functional analysis** (cont.)

- Simulate random patterns, then assigns objects depending on simulation response.

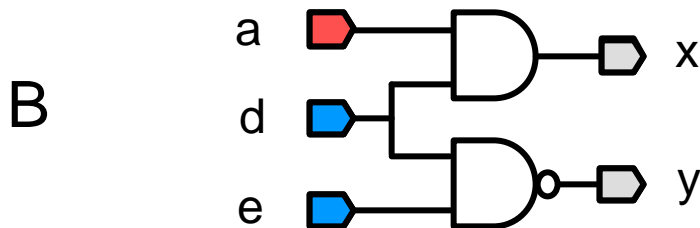
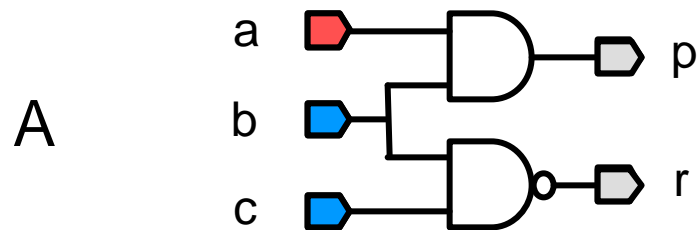


# Matching methods - Functional

---

- **Functional analysis** (cont.)

- Simulate random patterns, then assigns objects depending on simulation response. Note that ipoints in the same group always take the same random stimulus value.

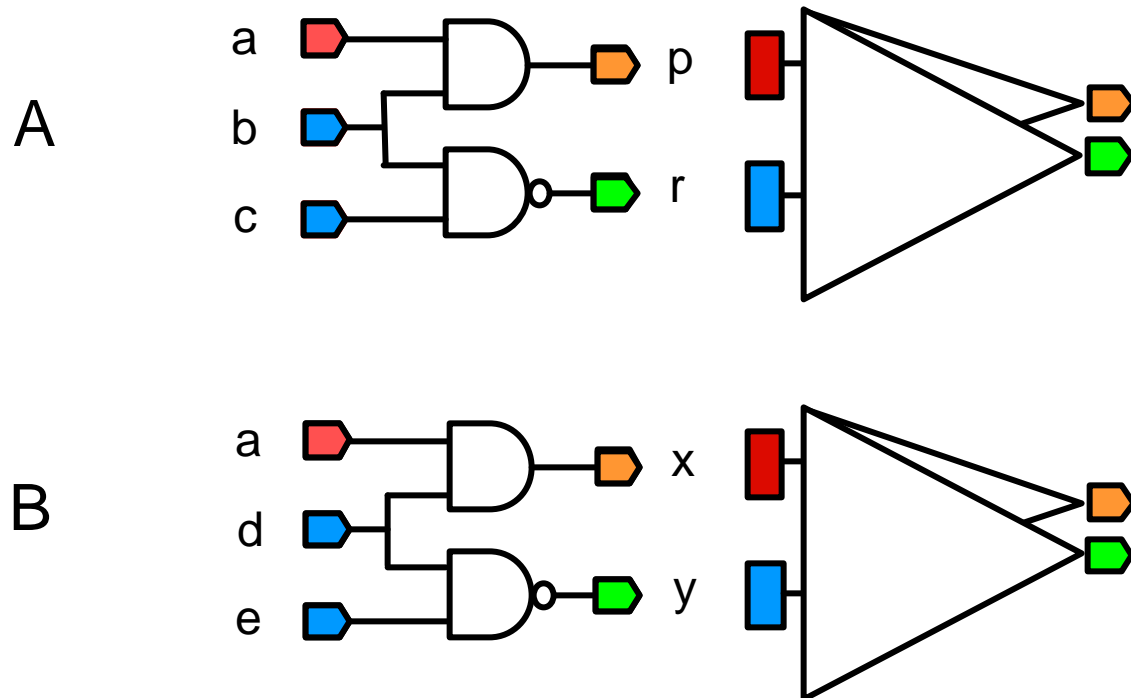


# Matching methods - Functional

---

- **Functional analysis** (cont.)

- Simulate random patterns, then assigns objects depending on simulation response. Note that ipoints in the same group always take the same random stimulus value.



# Our approach

---

- Extend the functional analysis technique.
- Solve the problem of non-equivalence checking rather than the one of equivalence checking.
- Use simulation and satisfiability techniques
- An ipoint vector is a set of 0's and 1's that obeys the following constraint.
  - ⊙ All ipoints in a partition have the same value

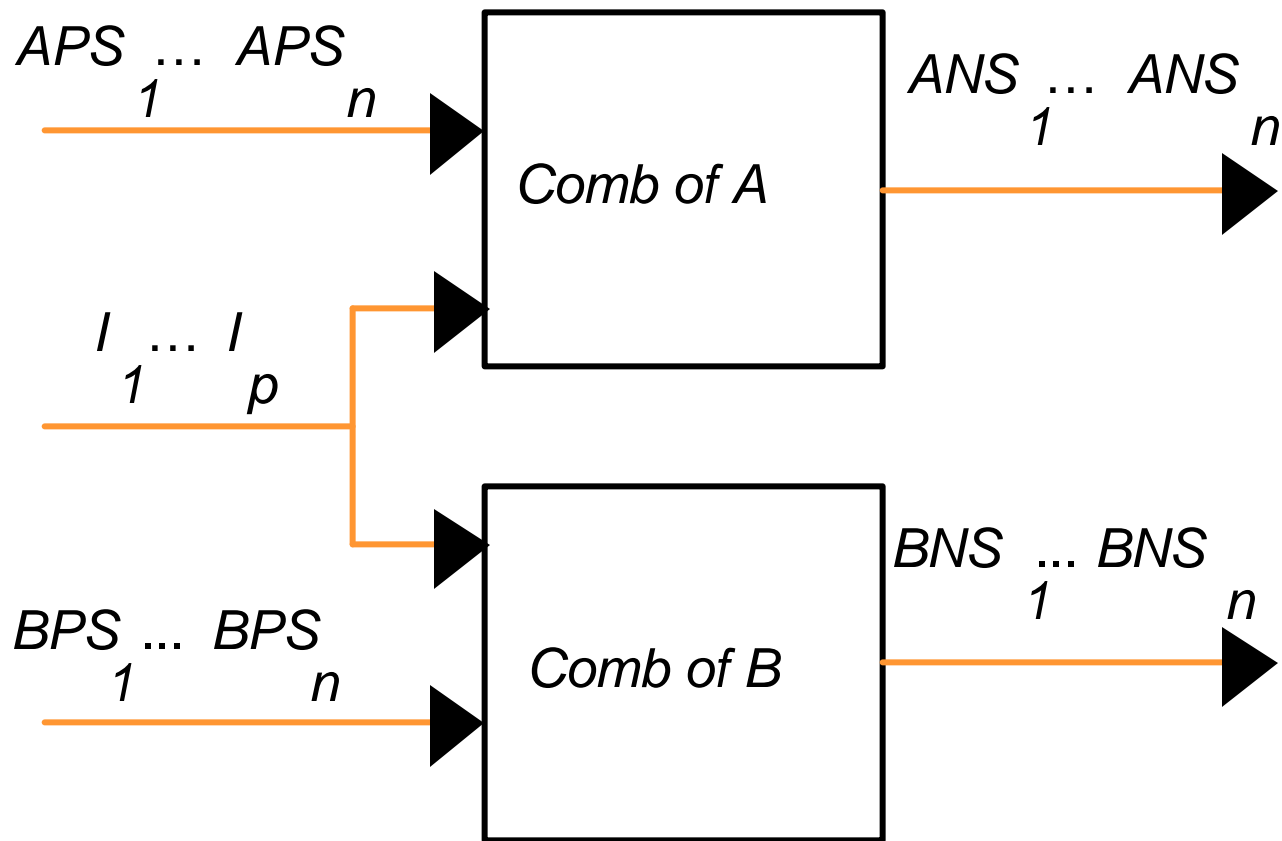
# Our approach

---

- Given two circuits with a total of  $n$  cpoints, if there exists an ipoint vector such that it sets  $m$  cpoints to 0's and  $(n-m)$  cpoints to 1's, create two partitions, one containing the 0-cpoints and one containing the 1-cpoints.
- Successively divide cpoint partitions into two new ones as long as any partition containing more than 2 cpoints. A cpoint partition of 2 elements, represents a pair of potentially equivalent points.
- For latches, repartition the ipoints corresponding to the partitioned cpoints.
- Given a set of  $n$  cpoints, creating  $n/2$  2-element partitions requires a maximum of  $(n/2 - 1)$  steps.

# Search Model

---



# Example of Partitioning

---

Partitioning Vectors

Partitions

$I_1 I_2 APS_1 APS_2 APS_3 BPS_1 BPS_2 BPS_3$      $[ANS_1 ANS_2 ANS_3 BNS_1 BNS_2 BNS_3]$   
 0 1 1 1 1 1 1 1 1    0 0 1 0 0 1



1 0 0 0 1 0 0 1     $[ANS_1 ANS_2 BNS_1 BNS_2]$   $[ANS_3 BNS_3]$   
 0 1 0 0 1 0 0



$[ANS_1 BNS_1]$   $[ANS_2 BNS_2]$   $[ANS_3 BNS_3]$

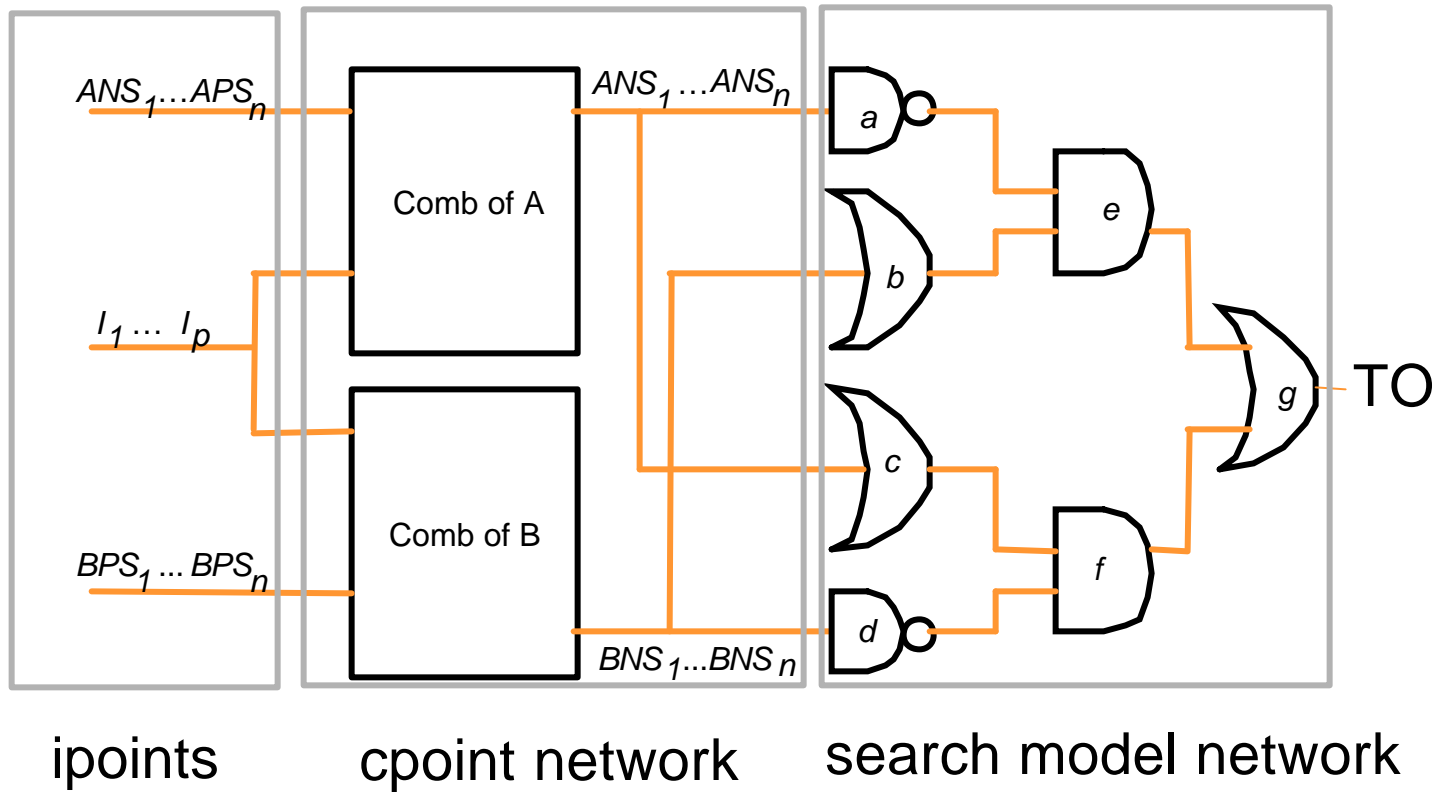
# Search Model For ATPG

---

- In order to break a cpoint partition into two, we need to find an ipoint vector that sets some cpoints to 0's and others to 1's.
- Create a search model with a single output (**TO**) such that an ipoint vector that sets **TO** to 1 is guaranteed to break a cpoint partition into 2 new partitions.

# Search Model For ATPG

---



# Partitioning Process

---

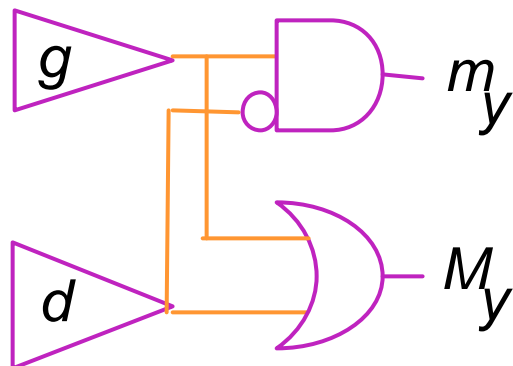
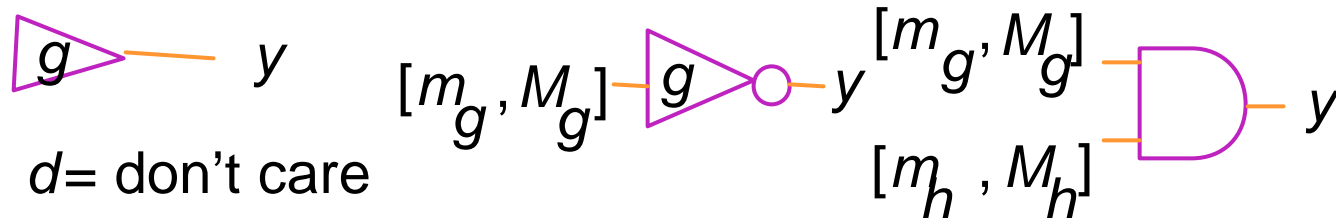
```
While (There exist unmarked cpoint partitions with
        size > 2) {
    Pick a cpoint partition;
    Create the search model for it;
    Search for an ipoint vector that sets TO to 1;
    If (found) {
        break the cpoint partition into two;
        repartition the ipoint partition for
            corresponding latches;
    }
}
```

# Designs with Don't Care (DC) Cells

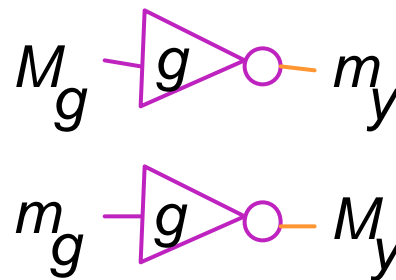
---

- Produce don't care (x) values at the cpoints.
- Modify cpoint partition circuit to encode don't care values for 2-value search engines.
- Enhance search model.

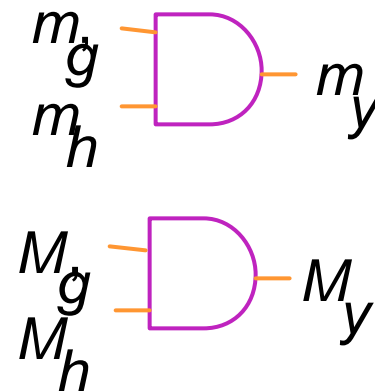
# DC Modification Rules



(a)



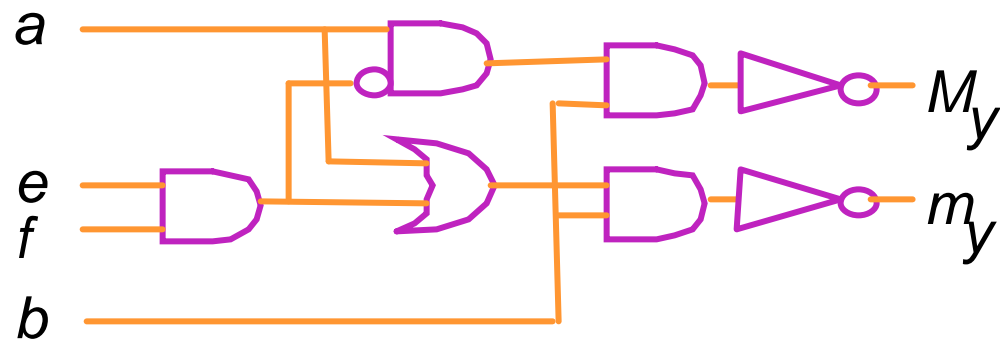
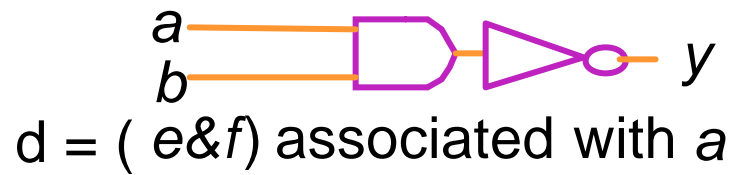
(b)



(c)

# Example of Use of DC Rules

---



# Example of Partitioning with DC

---

Partitioning Vectors

Partitions

$I_1$	$I_2$	$APS_1$	$APS_2$	$APS_3$	$BPS_1$	$BPS_2$	$BPS_3$	[ $ANS_1$ $ANS_2$ $ANS_3$ $BNS_1$ $BNS_2$ $BNS_3$ ]					
0	1	1	1	1	1	1	1	0	x	x	0	0	1
1	0	0	0	0	0	0	0	x	1	x	1	1	0
1	1	0	0	0	0	0	0	x	x	1	0	0	1

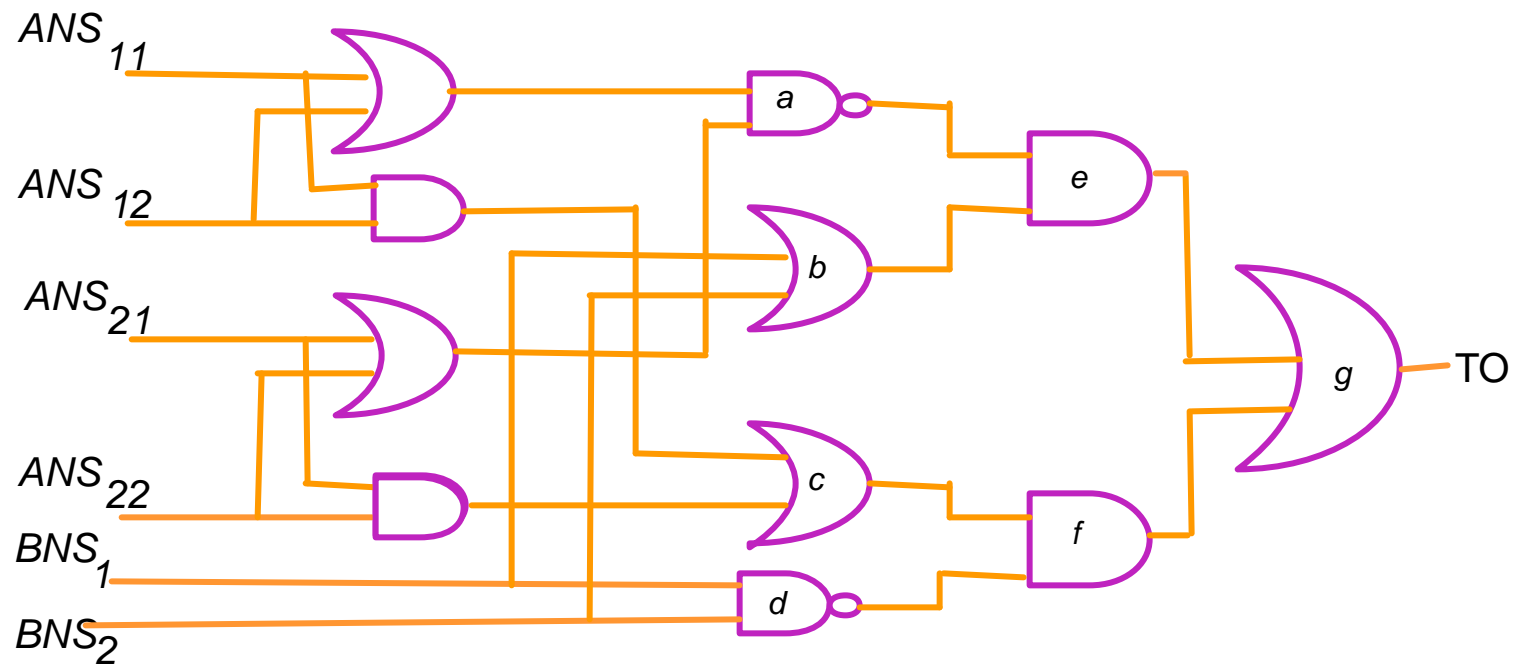


[ $ANS_1$   $ANS_2$   $BNS_1$   $BNS_2$ ] [ $ANS_3$   $BNS_3$ ]

# Augmented DC Search Network

---

## Example



# Results

---

## Experimental Results

D e s i g n s	Num. of gates	Num.of compare- point pairs	Matched by non- function- based methods	Matched by our heuristic method	CPU time (sec)
1	140k	3583	2762	821	50
2	112k	1843	667	1176	45
3	71k	859	411	448	32
4	86k	646	369	277	29
5	40k	718	417	301	6
6	156k	1975	1815	160	28
7	364k	4378	1795	2583	176

# Future Work

---

- Research Heuristics for further resolution of remaining unmatched pairs.
- Use the techniques to determine other matching structures
  - ⦿ For example, a multiplier in a flattened design