BOOLEAN METHODS

©Giovanni De Micheli

Stanford University

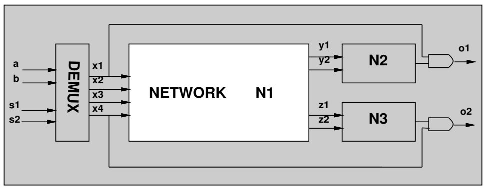
Boolean methods

- Exploit Boolean properties.
  - *Don't care* conditions.
- Minimization of the local functions.
- Slower algorithms, better quality results.

External *don't care* conditions

- *Controllability don't care* set CDC_{in}:
  - Input patterns never produced by the environment at the network's input.
- *Observability don't care* set ODC_{out}:
  - Input patterns representing conditions when an output is not observed by the environment.
  - Relative to each output.
  - Vector notation used: $\mathbf{ODC}_{out}$.

Example



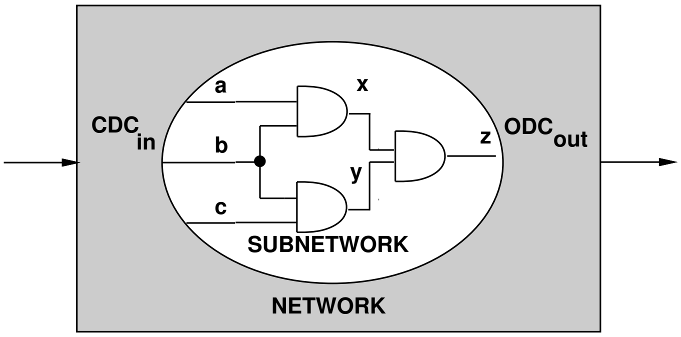
- Inputs driven by a de-multiplexer.
- $CDC_{in} = x_1'x_2'x_3'x_4' + x_1x_2 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + x_3x_4$.
- Outputs observed when $\begin{bmatrix} x_1 \\ x_4 \end{bmatrix} = \mathbf{1}$

$$\mathbf{ODC}_{out} = \begin{bmatrix} x_1' \\ x_1' \\ x_4' \\ x_4' \end{bmatrix}$$

Example
overall external *don't care* set

$$\mathbf{DC}_{ext} = \mathbf{CDC}_{in} + \mathbf{ODC}_{out} = \begin{bmatrix} x_1' + x_2 + x_3 + x_4 \\ x_1' + x_2 + x_3 + x_4 \\ x_4' + x_2 + x_3 + x_1 \\ x_4' + x_2 + x_3 + x_1 \end{bmatrix}$$

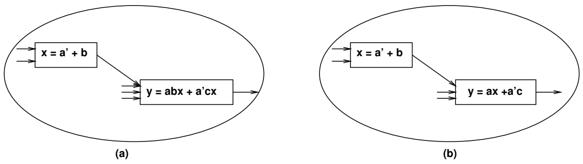
Internal *don't care* conditions



Internal *don't care* conditions

- Induced by the network structure.
- *Controllability don't care* conditions:
  - Patterns never produced at the inputs of a subnetwork.
- *Observability don't care* conditions:
  - Patterns such that the outputs of a subnetwork are not observed.

Example



- CDC of v_y includes $ab'x + a'x'$.
- Minimize f_y to obtain: $\widetilde{f_y} = ax + a'c$.

Satisfiability *don't care* conditions

© GDM

- Invariant of the network:
  - $x = f_x \to x \neq f_x \subseteq SDC$.
- $SDC \; = \sum_{v_x \in V^G} x \oplus f_x$
- Useful to compute controllability *don't cares* .

CDC computation

© GDM

- Network traversal algorithm:
  - Consider different *cuts* moving from input to output.
- Initial CDC is CDC_{in}.
- Move *cut* forward.
  - Consider SDC contributions of predecessors.
  - Remove unneded variables by *consensus*.

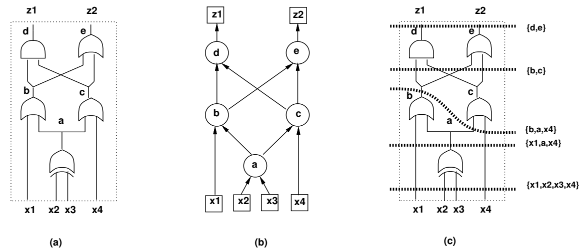
CDC computation

© GDM

CONTROLLABILITY($G_n(V,E)$, CDC_{in}) {
  $C = V^I$;
  $CDC_{cut} = CDC_{in}$;
  foreach vertex $v_x \in V$ in topological order {
    $C = C \cup v_x$;
    $CDC_{cut} = CDC_{cut} + f_x \oplus x$;
    $D = \{v \in C$ s.t. all dir. succ. of v are in $C\}$
    foreach vertex $v_y \in D$
      $CDC_{cut} = \mathcal{C}_y(CDC_{cut})$;
    $C = C \; - \; D$;
  };
  $CDC_{out} = CDC_{cut}$;
}

Example

© GDM



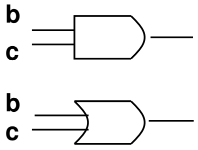
Example

- Assume $CDC_{in} = x_1' x_4'$.
- Select vertex v_a:
  - Contribution to CDC_{cut}: $a \oplus (x_2 \oplus x_3)$.
  - Drop variables $D = \{x_2, x_3\}$ by consensus:
  - $CDC_{cut} = x_1' x_4'$.
- Select vertex v_b:
  - Contribution to CDC_{cut}: $b \oplus (x_1 + a)$.
    * $CDC_{cut} = x_1' x_4' + b \oplus (x_1 + a)$.
  - Drop variable x_1 by consensus:
    * $CDC_{cut} = b' x_4' + b' a$.
- ...
- $CDC_{out} = e' = z_2'$.

CDC computation by image computation

- Network behavior at *cut*: **f**.
- CDC_{cut} is just the complement of the *image* of $(CDC_{in})'$ with respect to **f**.
- CDC_{cut} is just the complement of the *range* of **f** when $CDC_{in} = \emptyset$.
- Range can be computed recursively.
  - Terminal case: scalar function.
    * Range of $y = f(\mathbf{x})$ is $y + y'$ (any value) unless f (or f') is a tautology and the range is y (or y').

Example

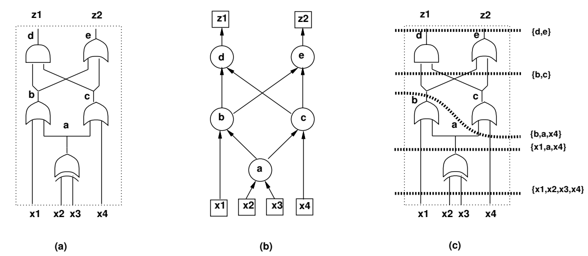


RANGE VECTORS

d	0	0	1
e	0	1	1

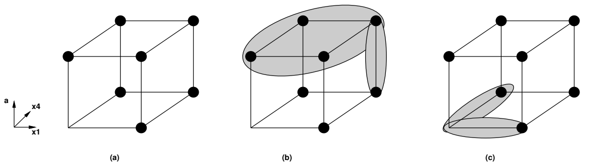
- $range(\mathbf{f}) = d\ range((b+c)|_{d=bc=1}) +$
  $+ d'\ range((b+c)|_{d=bc=0})$
- When $d = 1$, then $bc = 1 \rightarrow b + c = 1$ is TAUTOLOGY.
- If I choose 1 as top entry in output vector:
  - the bottom entry is also 1.
  - $\begin{bmatrix} 1 \\ ? \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
- When $d = 0$, then $bc = 0 \rightarrow b + c = \{0, 1\}$.
- If I choose 0 as top entry in output vector:
  - the bottom entry can be 0 or 1.
- $range(\mathbf{f}) = de + d'(e + e') = de + d' = d' + e$

Example



$$\mathbf{f} = \begin{bmatrix} f^1 \\ f^2 \end{bmatrix} = \begin{bmatrix} (x_1 + a)(x_4 + a) \\ (x_1 + a) + (x_4 + a) \end{bmatrix} = \begin{bmatrix} x_1 x_4 + a \\ x_1 + x_4 + a \end{bmatrix}$$

Example



$$
\begin{aligned}
range(\mathbf{f}) &= \\
&= d\ range(f^2|_{(x_1x_4+a)=1}) + \\
&\quad d'\ range(f^2|_{(x_1x_4+a)=0}) \\
&= d\ range(x_1+x_4+a|_{(x_1x_4+a)=1}) + \\
&\quad d'\ range(x_1+x_4+a|_{(x_1x_4+a)=0}) \\
&= d\ range(1) + d'\ range(a'(x_1 \oplus x_4)) \\
&= de + d'(e+e') \\
&= e + d'
\end{aligned}
$$

- $CDC_{out} = (e+d')' = de' = z_1 z_2'$.

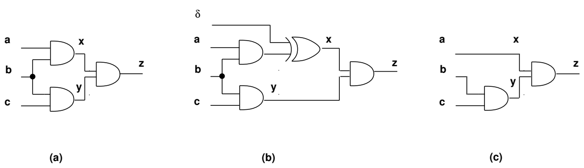
Perturbation method

© GDM

- Modify network by adding an extra input δ.
- Extra input can flip polarity of a signal x.
- Replace local function f_x by $f_x \oplus \delta$.
- Perturbed terminal behavior: $\mathbf{f}^x(\delta)$.

Example

© GDM



Observability *don't care* conditions

© GDM

- Conditions under which a change in polarity of a signal x is not perceived at the outputs.
- Complement of the Boolean difference:
  - $\partial f/\partial x \;=\; f|_{x=1} \oplus f|_{x=0}$.
- Equivalence of perturbed function: $\mathbf{f}^x(0) \overline{\oplus} \mathbf{f}^x(1)$.

Observability *don't care* computation

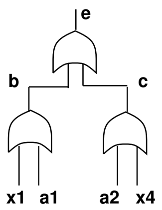
- Problem:
  - Outputs are not expressed as function of all variables.
  - If network is flattened to obtain **f**, it may explode in size.
- Requirement:
  - Local rules for ODC computation.
  - Network traversal.

Single-output network with tree structure

- Traverse network tree.
- At root:
  - ODC_{out} is given.
- At internal vertices:
  - $ODC_x \;=\; (\partial f_y/\partial x)' + ODC_y$

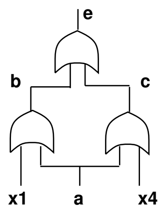
Example

$$\begin{aligned} e &= b + c \\ b &= x_1 + a_1 \\ c &= x_4 + a_2 \end{aligned}$$



- Assume $ODC_{out} = ODC_e = 0$.
- $ODC_b = (\partial f_e/\partial b)' = (b+c)|_{b=1}\overline{\oplus}(b+c)|_{b=0} = c.$
- $ODC_c = (\partial f_e/\partial c)' = b.$
- $ODC_{x_1} = ODC_b + (\partial f_b/\partial x_1)' = c + a_1.$
- ...

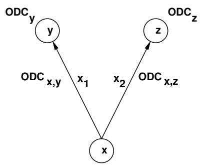
General networks



- Fanout reconvergence.
- For each vertex with two (or more) fanout stems:
  - The contribution of the ODC along the stems cannot be added *tout court*.
  - Interplay of different paths.
- More elaborate analysis.

Two-way fanout stem

© GDM



- Compute ODC sets associated with edges.
- Combine ODCs at vertex.
- Formula derivation:
  - Assume two equal perturbations on the edges.
  - $\mathbf{ODC}_x = \mathbf{f}^{x_1,x_2}(1,1) \;\overline{\oplus}\; \mathbf{f}^{x_1,x_2}(0,0)$

ODC formula derivation

© GDM

$$\begin{aligned}
\mathbf{ODC}_x &= \mathbf{f}^{x_1,x_2}(1,1) \;\overline{\oplus}\; \mathbf{f}^{x_1,x_2}(0,0) \\
&= \mathbf{f}^{x_1,x_2}(1,1) \;\overline{\oplus}\; \mathbf{f}^{x_1,x_2}(0,0) \\
&\quad \overline{\oplus}\; (\mathbf{f}^{x_1,x_2}(0,1) \;\overline{\oplus}\; \mathbf{f}^{x_1,x_2}(0,1)) \\
&= (\mathbf{f}^{x_1,x_2}(1,1) \;\overline{\oplus}\; \mathbf{f}^{x_1,x_2}(0,1)) \\
&\quad \overline{\oplus}\; (\mathbf{f}^{x_1,x_2}(0,1) \;\overline{\oplus}\; \mathbf{f}^{x_1,x_2}(0,0)) \\
&= \mathbf{ODC}_{x,y}|_{\delta_2=1} \;\overline{\oplus}\; \mathbf{ODC}_{x,z}|_{\delta_1=0} \\
&= \mathbf{ODC}_{x,y}|_{x_2=x'} \;\overline{\oplus}\; \mathbf{ODC}_{x,z}|_{x_1=x} \\
&= \mathbf{ODC}_{x,y}|_{x=x'} \;\overline{\oplus}\; \mathbf{ODC}_{x,z}
\end{aligned}$$

- Because $x = x_1 = x_2$.

Multi-way stems Theorem

© GDM

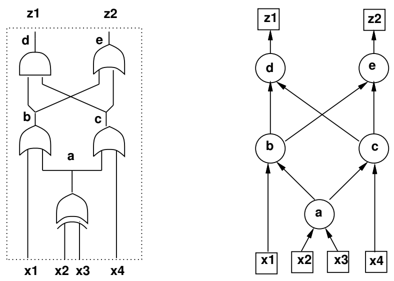
- Let $v_x \in V$ be any internal or input vertex.
- Let $\{x_i, i = 1, 2, \ldots, p\}$ be the edge vars corresponding to $\{(x, y_i)\ ;\ i = 1, 2, \ldots, p\}$.
- Let $\mathbf{ODC}_{x,y_i}\ ,\ i = 1, 2, \ldots, p$ the edge ODCs.
- $\mathbf{ODC}_x = \overline{\bigoplus}_{i=1}^{p} \mathbf{ODC}_{x,y_i}|_{x_{i+1}=\cdots=x_p\ =x'}$

Observability *don't care* algorithm

© GDM

OBSERVABILITY($G_n(V,E)$, $\mathbf{ODC}_{out}$) {
  foreach vertex $v_x \in V$ in reverse topological order {
    for ($i = 1$ to p)
      $\mathbf{ODC}_{x,y_i} = (\partial f_{y_i}/\partial x)'\mathbf{1} + \mathbf{ODC}_{y_i}$;
    $\mathbf{ODC}_x = \overline{\bigoplus}_{i=1}^{p} \mathbf{ODC}_{x,y_i}|_{x_{i+1}=\cdots=x_p\ =x'}$;
  }
}

Example



(a) (b)

$$\mathbf{ODC}_d = \begin{pmatrix} 0 \\ 1 \end{pmatrix}\ ; \mathbf{ODC}_e = \begin{pmatrix} 1 \\ 0 \end{pmatrix} ; \mathbf{ODC}_c = \begin{pmatrix} b' \\ b \end{pmatrix}\ ; \mathbf{ODC}_b = \begin{pmatrix} c' \\ c \end{pmatrix}$$

$$\mathbf{ODC}_{a,b} = \begin{pmatrix} c' + x_1 \\ c + x_1 \end{pmatrix} = \begin{pmatrix} a'x_4' + x_1 \\ a + x_4 + x_1 \end{pmatrix}$$

$$\mathbf{ODC}_{a,c} = \begin{pmatrix} b' + x_4 \\ b + x_4 \end{pmatrix} = \begin{pmatrix} a'x_1' + x_4 \\ a + x_1 + x_4 \end{pmatrix}$$

$$\mathbf{ODC}_a = \mathbf{ODC}_{a,b}|_{a=a'} \overline{\oplus} \mathbf{ODC}_{a,c} = \begin{pmatrix} ax_4' + x_1 \\ a' + x_4 + x_1 \end{pmatrix} \overline{\oplus} \begin{pmatrix} a'x_1' + x_4 \\ a + x_1 + x_4 \end{pmatrix} =$$

$$= \begin{pmatrix} x_1x_4 \\ x_1 + x_4 \end{pmatrix}$$

Transformations with *don't cares*

- Boolean simplification:
  - Use standard minimizer (Espresso).
  - Minimize the number of literals.
- Boolean substitution:
  - Simplify a function by adding an extra input.
  - Equivalent to simplification with global *don't care* conditions.

Example
Boolean substitution

- Substitute $q = a + cd$ into $f_h = a + bcd + e$ to get $f_h = a + bq + e$.
- SDC set: $q \oplus (a + cd) = q'a + q'cd + qa'(cd)'$.
- Simplify $f_h = a+bcd+e$ with $q'a+q'cd+qa'(cd)'$ as *don't care* .
- Simplification yields $f_h = a + bq + e$.
- One literal less by changing the support of f_h.

Single-vertex optimization

$SIMPLIFY\_SV(\ G_n(V,E)\ )\{$
  repeat $\{$
    $v_x =$ selected vertex ;
    Compute the local *don't care* set DC_x;
    Optimize the function f_x ;
  $\}$**until** (no more reduction is possible)
$\}$

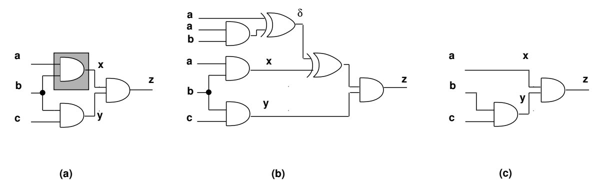
Optimization and perturbations

© GDM

- Replace f_x by g_x.
- Perturbation $\delta_x = f_x \oplus g_x$.
- Condition for feasible replacement:
  - Perturbation bounded by local *don't care* set
  - $\delta_x \subseteq \mathbf{DC}_{ext} + \mathbf{ODC}_x$
  - If x not a primary input consider also CDC set.

Example

© GDM



- No external *don't care* set.
- Replace AND by wire: $g_x = a$
- Analysis:
  - $\delta = f_x \oplus g_x \;=\; ab \oplus a \;=\; ab'$.
  - $ODC_x = y' = b' + c'$.
  - $\delta = ab' \subseteq DC_x = b' + c' \Rightarrow$ feasible!

Degrees of freedom

© GDM

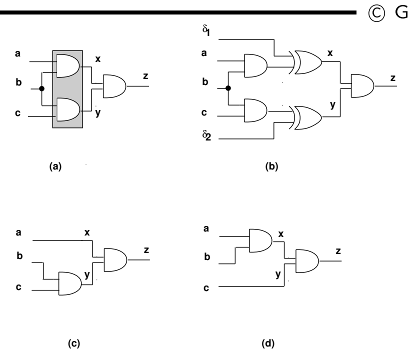
- Fully represented by *don't care* conditions:
  - External *don't cares* .
  - Internal observability and controllability.
- *Don't cares* represent an *upper bound* on the perturbation.
- Approximations:
  - Use smaller *don't care* sets to speed-up the computation.

Multiple-vertex optimization

© GDM

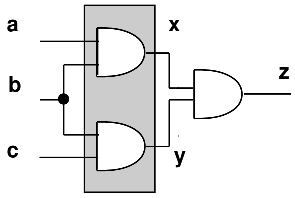
- Simplify more than one local function at a time.
- Potentially better (more general) approach.
- Analysis:
  - Multiple perturbations.
- Condition for feasible replacement:
  - *Upper and lower bounds* on the perturbation.
  - Boolean relation model.

Example



- The two perturbations are related.
- Cannot change simultaneously:
  - $ab \rightarrow a$.
  - $cb \rightarrow c$.

Multiple-vertex optimization Boolean relation model



a	b	c	x, y
0	0	0	{ 00, 01, 10 }
0	0	1	{ 00, 01, 10 }
0	1	0	{ 00, 01, 10 }
0	1	1	{ 00, 01, 10 }
1	0	0	{ 00, 01, 10 }
1	0	1	{ 00, 01, 10 }
1	1	0	{ 00, 01, 10 }
1	1	1	{ 11 }

Multiple-vertex optimization Boolean relation model

- Compute Boolean relation:
  - Flatten the network. Analyze patterns.
  - Derive equivalence relation from ODCs.
- Use relation minimizer.
- Example of minimum function:

a	b	c	x, y
1	$*$	$*$	10
$*$	1	1	01

Multiple-vertex optimization Boolean relation model


```
SIMPLIFY_MVR( G_n(V,E) ){
    repeat {
        U = selected vertex subset;
        foreach vertex v_x ∈ U
            Compute OCD_x;
        Determine the equiv. classes of the Boolean relation
          of the subnetwork induced by U;
        Find an optimal function compatible with the relation
          using a relation minimizer;
    }until (no more reduction is possible);
}
```

Multiple-vertex optimization compatible *don't cares*

© GDM

- Determine *compatible don't cares* :
  - CODCs: subset of ODCs.
  - Decouple dependencies.
  - Reduced degrees of freedom.
- Using compatible ODCs, only *upper bounds* on the perturbation need to be satisfied.

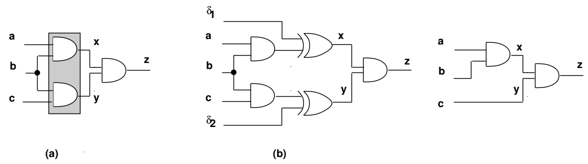
Example two perturbations

© GDM

- First vertex:
  - CODC equal to its ODC set.
  - $CODC_{x_1} = ODC_{x_1}$.
- The second vertex:
  - CODC smaller than its ODC to be safe enough to allow transformations permitted by the first ODC.
  - $CODC_{x_2} = \ \mathcal{C}_{x_1}(ODC_{x_2}) + ODC_{x_2}ODC'_{x_1}$
- Order dependence.

Example first vertex v_y

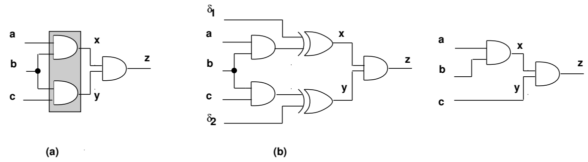
© GDM



- $CODC_y = ODC_y = x' = b' + a'$
- $ODC_x = y' = b' + c'$
- $CODC_x = \mathcal{C}_y(ODC_x) + ODC_x(ODC_y)' \ = \mathcal{C}_y(y') + y'x = y'x = (b' + c')ab = abc'$.

Example (2)

© GDM



- Allowed perturbation:
  - $f_y = bc \to g_y = c$.
  - $\delta_y = bc \oplus c = b'c \subseteq CODC_y = b' + a'$.
- Disallowed perturbation:
  - $f_x = ab \to g_x = a$.
  - $\delta_x = ab \oplus a = ab' \not\subseteq CODC_x = abc'$.
- The converse holds if v_x is the first vertex.

Multiple-vertex optimization compatible *don't cares*

$SIMPLIFY\_MV(\ G_n(V,E)\ )\{$
repeat $\{$
$U =$ selected vertex subset;
foreach vertex $v_x \in U$
Compute $COCD_x$ and the corresponding local *don't care* subset $\widetilde{DC}_x$;
Optimize simultaneously the functions at U;
$\}$**until** (no more reduction is possible);
$\}$

Summary Boolean methods

- Boolean methods exploit *don't care* sets and simplification of logic representations.
- *Don't care* set computation:
  - Controllability and observability.
- Single and multiple transformations.

Synthesis and testability

- Testability:
  - Ease of testing a circuit.
- Assumptions:
  - Combinational circuit.
  - Single or multiple *stuck-at* faults.
- Full testability:
  - Possible to generate test set for all faults.
  - Restrictive interpretation.

Test for *stuck-at*s

- Net y stuck-at 0.
  - Input pattern that sets y to true.
  - Observe output.
  - Output of faulty circuit differs.
- Net y stuck-at 1.
  - Same, but set y to false.
- Need *controllability* and *observability*.

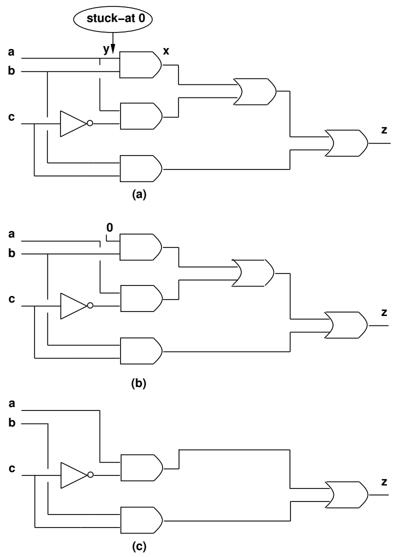
Test sets *don't care* interpretation

- Stuck-at 0 on net y.
  - $\{\mathbf{t} | y(\mathbf{t}) \cdot ODC'_y(\mathbf{t}) = 1\}$.
- Stuck-at 1 on net y.
  - $\{\mathbf{t} | y'(\mathbf{t}) \cdot ODC'_y(\mathbf{t}) = 1\}$.

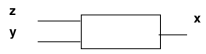
Using testing methods for synthesis

- Redundancy removal.
  - Use TPG to search for untestable faults.
- If stuck-at 0 on net y is untestable:
  - Set $y = 0$.
  - Propagate constant.
- If stuck-at 1 on y is untestable:
  - Set $y = 1$.
  - Propagate constant.

Example



Redundancy removal and perturbation analysis



- Stuck-at 0 on y.
  - y set to 0. Namely $g_x = f_x|_{y=0}$.
  - Perturbation:
    - $\delta = f_x \oplus f_x|_{y=0} \; = \; y \cdot \partial f_x / \partial y$.
- Perturbation is feasible $\Leftrightarrow$ fault is untestable.
  - No input vector $\mathbf{t}$ can make $y(\mathbf{t}) \cdot ODC'_y(\mathbf{t})$ true.
  - No input vector can make $y(\mathbf{t}) \cdot ODC'_x(\mathbf{t}) \cdot \partial f_x / \partial y$ true.
    - because $ODC_y = ODC_x + (\partial f_x / \partial y)'$.

Redundancy removal and perturbation analysis

- Assume untestable stuck-at 0 fault.
- $y \cdot ODC'_x \cdot \partial f_x / \partial y \subseteq SDC$.
- Local *don't care* set:
  - $DC_x \supseteq ODC_x + y \cdot ODC'_x \cdot \partial f_x / \partial y$.
  - $DC_x \supseteq ODC_x + y \cdot \partial f_x / \partial y$.
- Perturbation $\delta \; = \; y \cdot \partial f_x / \partial y$.
  - Included in the local *don't care* set.

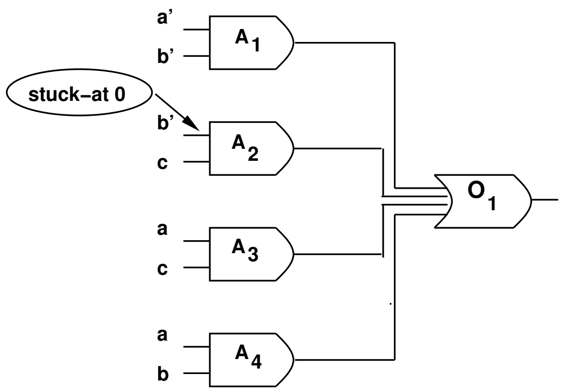
Synthesis for testability

- Synthesize networks that are fully testable.
  - Single stuck-at faults.
  - Multiple stuck-at faults.
- Two-level forms.
- Multiple-level networks.

Two-level forms

- Full testability for single stuck-at faults:
  - Prime and irredundant cover.
- Full testability for multiple stuck-at faults:
  - Prime and irredundant cover when:
    - Single-output function.
    - No product term sharing.
    - Each component is PI.

Example
$f = a'b' + b'c + ac + ab$



Multiple-level networks
Definitions

- A logic network $G_n(V, E)$,
  with local functions in *sum of product* form.

- Prime and irredundant (PI):
  - No literal nor implicant of any local function can be dropped.

- Simultaneously prime and irredundant (SPI):
  - No subset of literals and/or implicants can be dropped.

Multiple-level networks
Theorems

- A logic network is PI and only if:
  - its AND-OR implementation is fully testable for single stuck-at faults.

- A logic network is SPI if and only if:
  - its AND-OR implementation is fully testable for multiple stuck-at faults.

Multiple-level networks
Synthesis

- Compute full local *don't care* sets.
  - Make all local functions PI w.r. to *don't care* sets.

- Pitfall:
  - *Don't cares* change as functions change.

- Solution:
  - Iteration (Espresso-MLD).

- If iteration converges, network is fully testable.

Multiple-level networks
Synthesis

- Flatten to two-level form.
  - When possible – no size explosion.

- Make SPI by disjoint logic minimization.

- Reconstruct multiple-level network:
  - *Algebraic transformations preserve multifault testability.*

Summary

© GDM

- Synergy between synthesis and testing.
- Testable networks correlate to small-area networks.
- *Don't care* conditions play a major role.