Manufacture Testing of Digital Circuits

Prof. K-T Cheng

UC Santa Barbara

Prof. Srinivas Devadas

MIT

Prof. Kurt Keutzer

Mukul Prasad

University of California

Berkeley, CA

Class News

Assignment of grade range for midterm on Wednesday

Preliminary project report due 11/3

Second midterm

- Currently scheduled for 11/8 to be turned in 11/10!
- Reschedule 11/8 to be turned in 11/15

Design Process

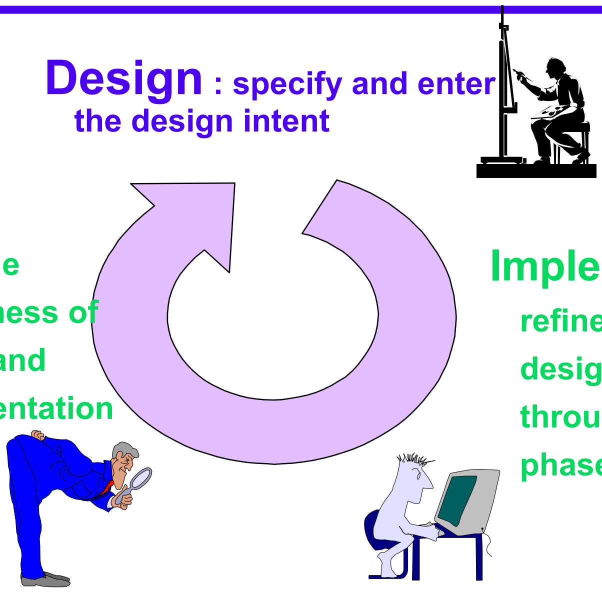
Design : specify and enter the design intent

Verify:

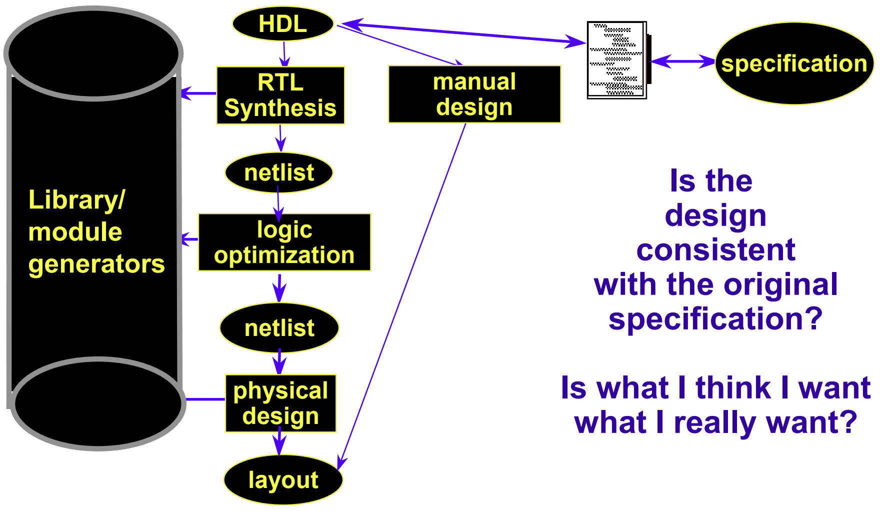
verify the correctness of design and implementation

Implement:

refine the design through all phases



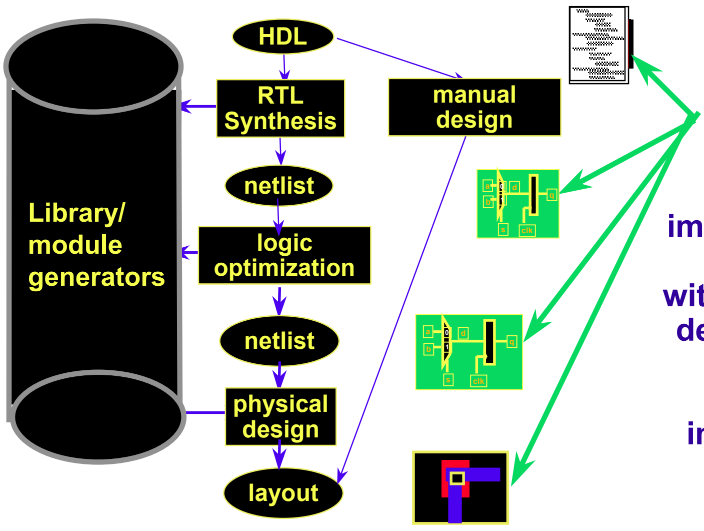
Design Verification



Is the design consistent with the original specification?

Is what I think I want what I really want?

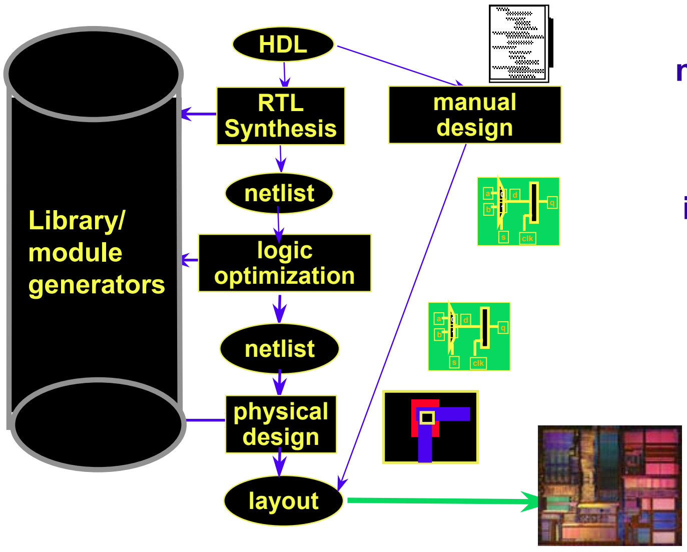
Implementation Verification



Is the implementation consistent with the original design intent?

Is what I implemented what I wanted?

Manufacture Verification (Test)



Is the manufactured circuit consistent with the implemented design?

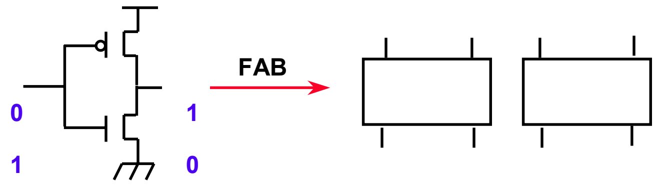
Did they build what I wanted?

Testing

Apply a sequence of inputs to a circuit

Observe the output response and compare the response with a precomputed or "expected" response

Any discrepancy is said to constitute an error, the cause of which is a physical defect



Defects and Fault models

Manufacturing defects can manifest in a variety of ways:

- Bridging
- Contaminants
- Shorts
- Opens
- Transistors stuck-open

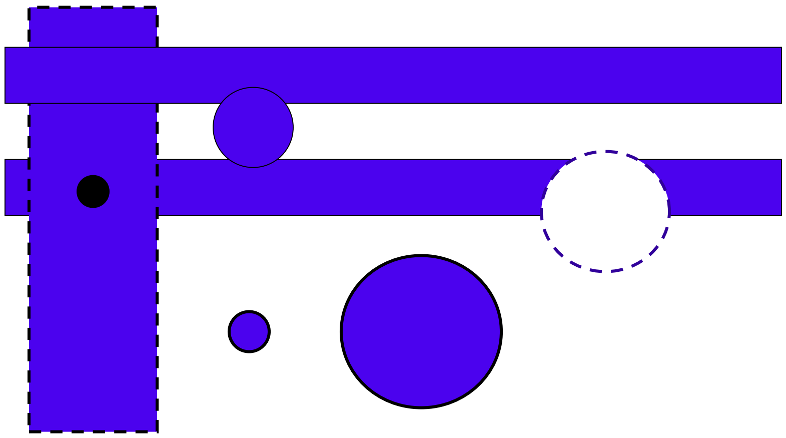
These need to be reduced to models:

- Single stuck-at-1, stuck-at-0
- Multiple stuck-at-1, stuck-at-0
- Delay fault models:
  - Gate
  - Path
  - x {hazard-free, hazard-free robust}

Presently:

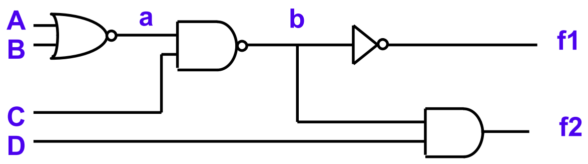
- single-stuck-at fault model ubiquitous
- some use of delay fault modeling

Defect-related Yield Loss



fatal defect types (two types of short circuits, one type of open)

Defect Model: Stuck-At Faults

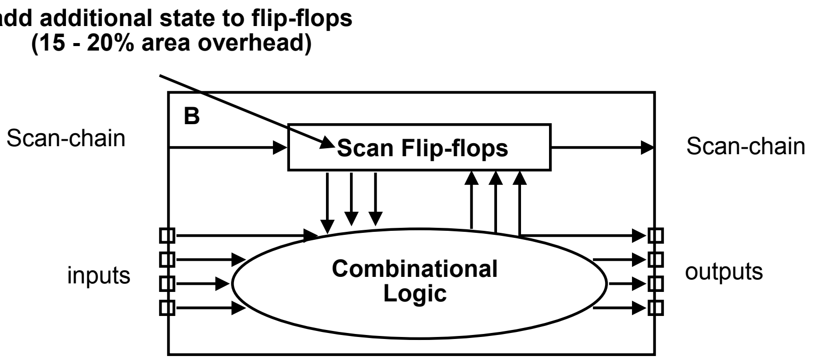


Any input or internal wire in circuit can be stuck-at-1 or stuck-at-0

Single stuck-at-fault model: In the faulty circuit, a single line/wire is S-a-0 or S-a-1

Multiple stuck-at fault model: In the faulty circuit any subset of wires are S-a-0/S-a-1 (in any combination)

Reduce to Combinational Logic Problem



Test Generation

Choose a fault model, e.g., single stuck-at fault model

Given a combinational circuit which realizes the function $f(x_1, x_2, \ldots x_n)$, a logical fault alters it to $f_\propto (x_1, x_2, \ldots x_n)$

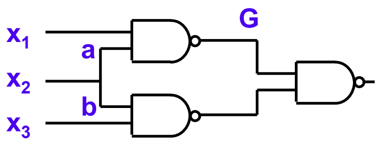
Inputs detecting $\propto$ are $f \oplus f_\propto$ $(= 1)$

Interested in <u>one</u> vector

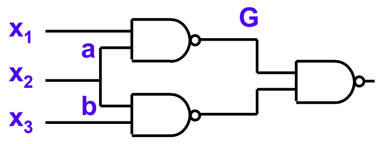
$A = (a_1, a_2, \ldots, a_n) \in f \oplus f_\propto$

Single Stuck-At Faults

A fault is assumed to occur only on a single line.



$Z = x_1 x_2 + x_2 x_3$



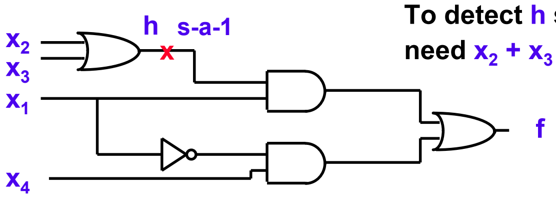
a s-a-1 $Z = x_1 + x_2 x_3$

G s-a-1 $Z = x_2 x_3$

This model is used because it has been found to be statistically correlated with defect-free circuits

Activation and Path Sensitization

In order for an input vector X to detect a fault a s-a-j, j = 0,1 the input X must cause the signal a in the normal (fault-free) circuit to take the value $\overline{j}$.

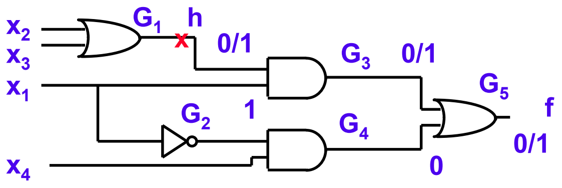


To detect h s-a-1, need $x_2 + x_3 = 0$, i.e., $\overline{x_2}\,\overline{x_3}$

The condition is necessary but not sufficient. Error signal must be propagated to output.

Fault Activation

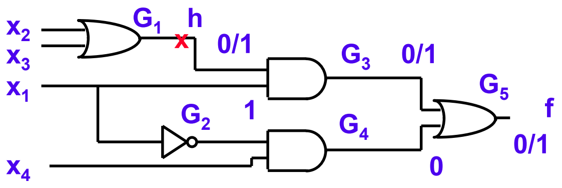
The faulty signal must be propagated along some path from its origin to an output



How to activate the fault?

Fault Activation

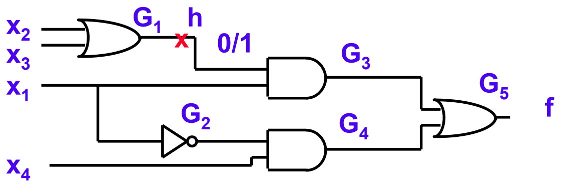
The faulty signal must be propagated along some path from its origin to an output



h s-a-1, for h to be 0, need $x_2 = x_3 = 0$ ($\overline{x_2}\ \overline{x_3}$)

Fault Propagation

The error signal must be propagated along some path from its origin to an output

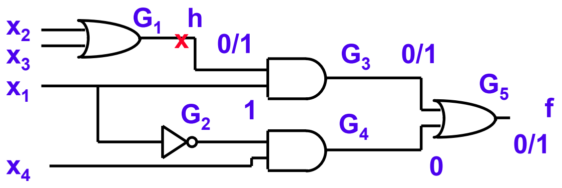


h s-a-1, for h to be 0, need $x_2 = x_3 = 0$ $(\overline{x_2}\ \overline{x_3})$

How to propagate the fault?

Fault Propagation

The error signal must be propagated along some path from its origin to an output



h s-a-1, for h to be 0, need $x_2 = x_3 = 0$ $(\overline{x_2}\ \overline{x_3})$

Only one path G_3, G_5

In order to propagate an error through AND gate G_3, other input $x_1 = 1$. To propagate through G_5, need $G_4 = 0$, $x_1 + \overline{x_4}$

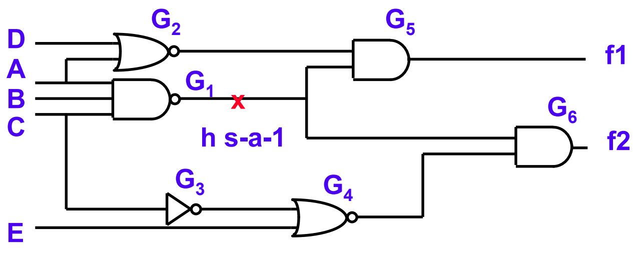
Single Path Sensitization (SPS)

1. Activate: Specify inputs so as to generate the appropriate value (0 for s-a-1, 1 for s-a-0) at the site of the fault.

2.Propagate: Select *a* path from the site of the fault to an output and specify additional signal values to propagate the fault signal along this path to the output
(error propagation).

3. Justify; Specify input values so as to produce the signal values specified in (2)
(line justification).

Sensitization Example - 1

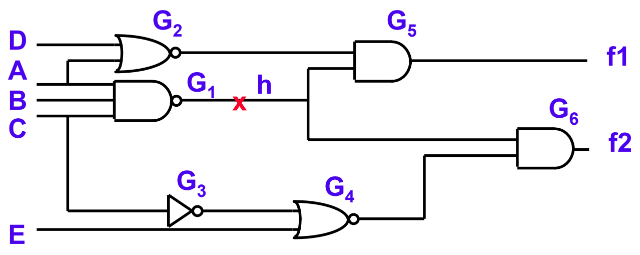


h s-a-1

Activate?

–

Sensitization Example - 2



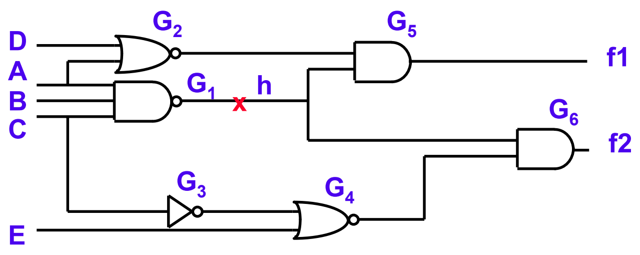
h s-a-1

Activate: To generate h = 0, need A = B = C = 1

Propagate?

–

Sensitization Example - 2



h s-a-1

To generate h = 0, need A = B = C = 1

Have a choice of propagating through G_5 or via G_6.

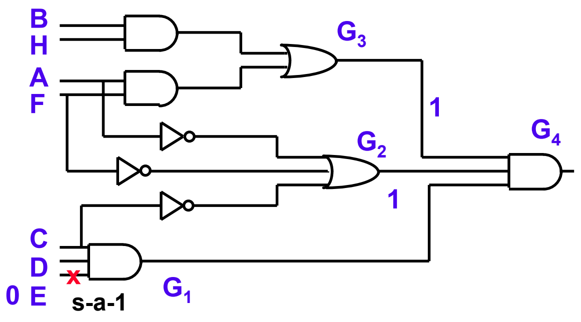
Propagating through G_5 requires $G_2 = 1$

$\Rightarrow$ A = D = 0 Contradiction

Propagating through G_6 requires $G_4 = 1$ $\Rightarrow$ C = 1, E = 0.

A valid test vector is $ABC\bar{E}$

Line Justification



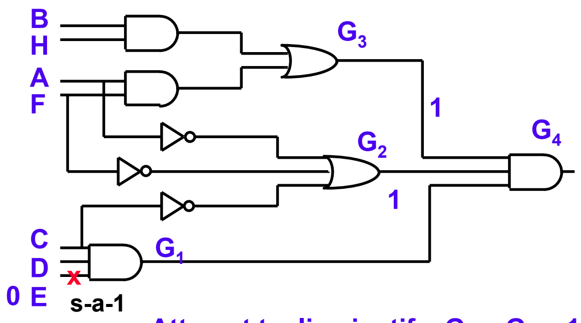
E s-a-1 $\Rightarrow$ E = 0

C = D = 1 to propagate through G_1.

To propagate through G_4, need $G_2 = G_3 = 1$

How do we justify these values?

Line Justification - 2



Attempt to line justify $G_2 = G_3 = 1$

$G_3 = 1$ possible if A = F = 1 or B = H = 1

If A = C = 1, then $G_2 = 0$.

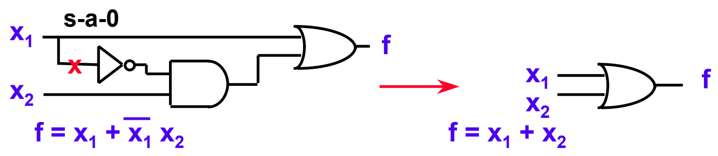
$G_3 = 1 \Rightarrow$ B = H = 1

$G_2 = 1$ needs A = 0 or F = 0

Tests are !ABCD!EH, BCD!E!FH

Redundancy

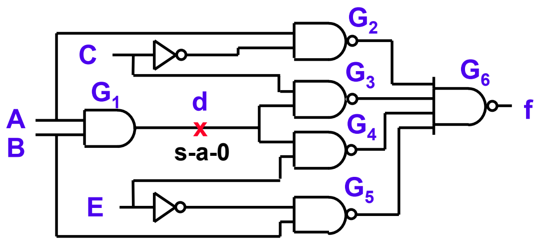
Existence of a fault does not change the functionality of a circuit $\Rightarrow$ redundant fault



$f = x_1 + \overline{x_1}\, x_2$ $\qquad$ $f = x_1 + x_2$

A test generation algorithm is deemed complete if it either finds a test for any fault or proves its redundancy, upon terminating.

Completeness of SPS method ?



d s-a-0 $\Rightarrow$ A = B = 1

Propagate along G_3, G_6 $\Rightarrow$ C = 1

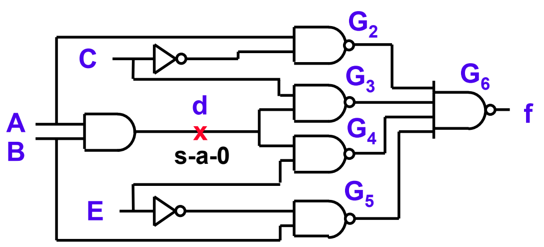
$G_2 = G_4 = G_5 = 1$

For $G_4 = 1$ either $G_1 = 0$ or E = 0

If $G_1 = 0$ fault is not activiated

If E = 0 (B must be 1) $\Rightarrow$ $G_5 = 0$ Inconsistency

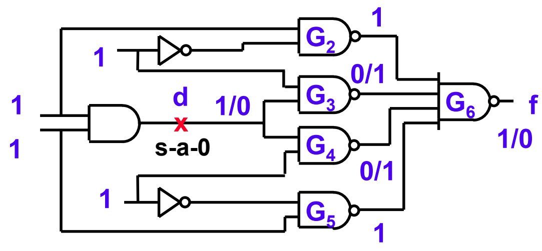
Completeness of SPS? - 2



Propagation along G_4, G_6 also results in inconsistencies by symmetric argument

Is there no test?

Multiple Path Sensitization



Error propagates down two paths G_3, G_6 and G_4, G_6 to output

It's natural to work backwards (justifying) and forwards (propagating) from point of fault activation but this focuses on sensitizing a single path

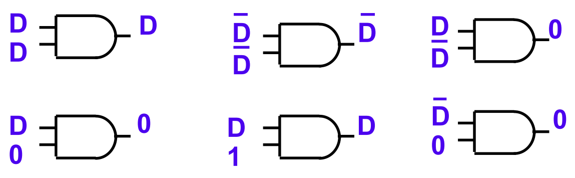
Attempting to sensitize a single path will not find a test for this fault

First notation: D-Algebra/ D-calculus

Need to be able to deal with multiple "errors" at the inputs to a gate

D represents a signal which has value 1 in normal circuit, and value 0 in faulty circuit.

$\overline{D} \equiv 0/1$



D, $\overline{D}$ behave like Boolean variables

Podem strategy – Goel

Podem uses a brilliant simplification to avoid the single-path sensitization trap - only primary inputs are assigned a value

Values are assigned to primary inputs, then propagated forward – need a compatibility between required value and PI value

Continue to assign PI values one at a time

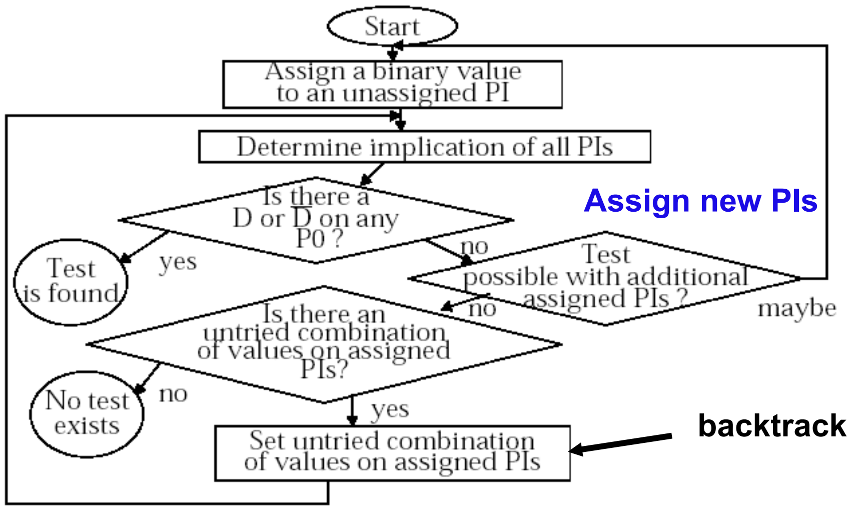
- Implicate values forward
- check to see if the faulty value has propagated to an output – if so then you have a test

If at any point there is a conflict between the PIs and

- Exciting the faulty value
- Propagating the faulty value forward

backtrack – but only at the primary inputs, if you have tried all combinations then halt with failure to find test

Podem decision procedure


Start
Assign a binary value to an unassigned PI
Determine implication of all PIs
Is there a D or D̄ on any P0 ?
yes
Test is found
no
Test possible with additional assigned PIs ?
maybe
no
Is there an untried combination of values on assigned PIs?
no
No test exists
yes
Set untried combination of values on assigned PIs

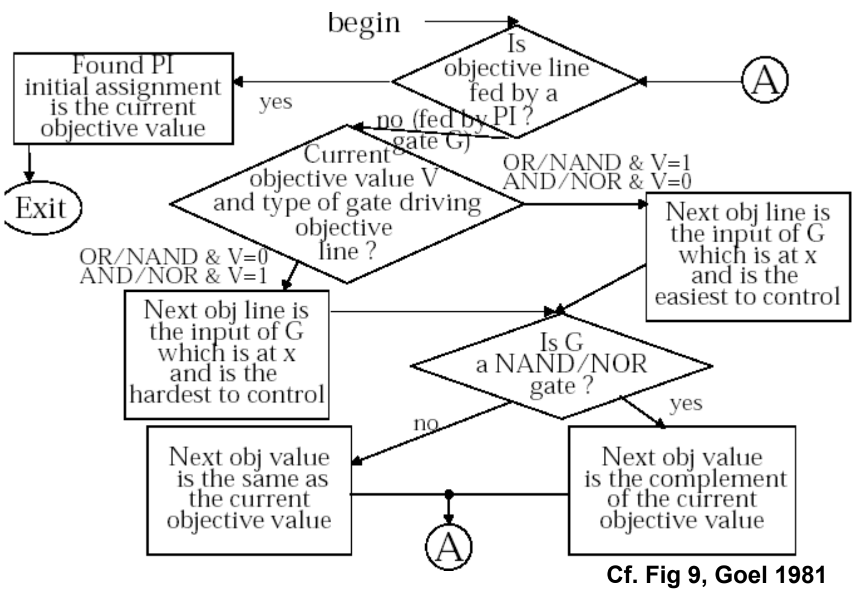
Assign new PIs

backtrack

Try a new value on existing PIs

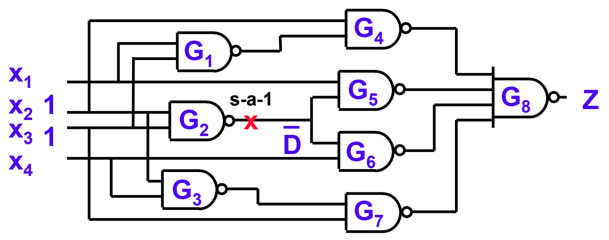
Cf. Fig 5, Goel 1981

Flowchart of Backtrace (not backtrack!)


begin
Is objective line fed by a PI ?
yes
Found PI initial assignment is the current objective value
Exit
A
no (fed by gate G)
Current objective value V and type of gate driving objective line ?
OR/NAND & V=1
AND/NOR & V=0
Next obj line is the input of G which is at x and is the easiest to control
OR/NAND & V=0
AND/NOR & V=1
Next obj line is the input of G which is at x and is the hardest to control
Is G a NAND/NOR gate ?
no
yes
Next obj value is the same as the current objective value
Next obj value is the complement of the current objective value
A

Cf. Fig 9, Goel 1981

PODEM Example



Initial objective: (0, G_2)

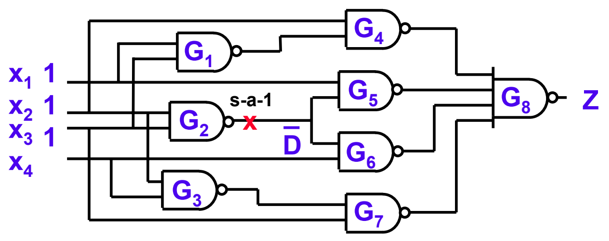
Backtrace to PIs: $x_2 = 1$

Objective: (0, G_2)

Backtrace: $x_3 = 1$

Implication: $G_2 = \overline{D}$

Podem Example – 2a



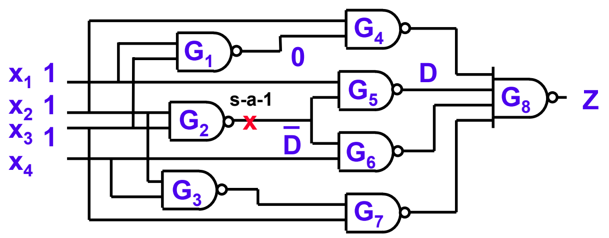
D-frontier is {G_5, G_6}

Attempt to propagate through G_5

Require $x_1 = 1$

Implication?

Podem Example – 2b



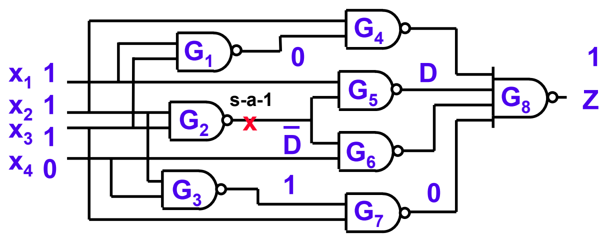
D-frontier is $\{G_5, G_6\}$

Attempt to propagate through G_5

Require $x_1 = 1$

Implication $G_1 = 0$, $G_4 = 1$, $G_5 = D$

Podem Example – 3a



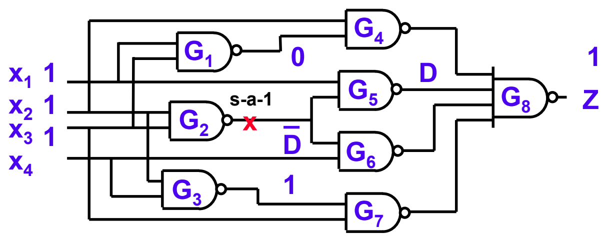
Attempt to propagate D through G_8.

Objective $(1, G_6)$

Backtrace to set $x_4 = 0$

Implication produces $G_3 = 1$ $G_7 = 0$ $G_8 = 1$
failed in propagating error

Podem Example – 3b

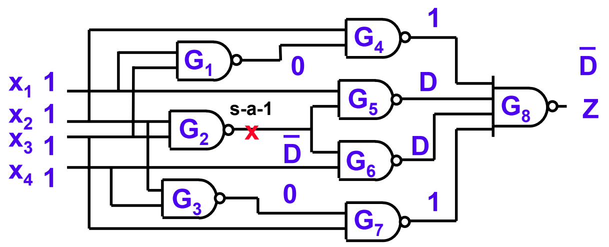


Attempt to propagate D through G_8.

Objective (1, G_6)

Backtrace

Podem Example - 4



BACKTRACK to most recent assignment x_4

Try alternative value $x_4 = 1$

Implication results in $G_3 = 0$, $G_6 = D$, $G_8 = \overline{D}$

Generated test 1111

Status on Podem

Podem approach very successful

At the core of most ATPG systems today

Spawned many additional innovations

- FAN – Fujiwara – sophisticated backtrace
- Socrates – Schulz – learning

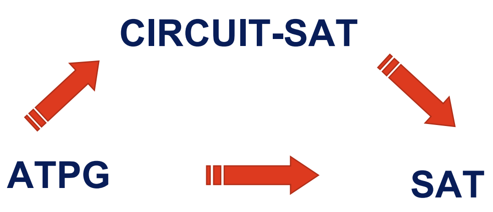
But if we had it all to do over …

Another approach to ATPG

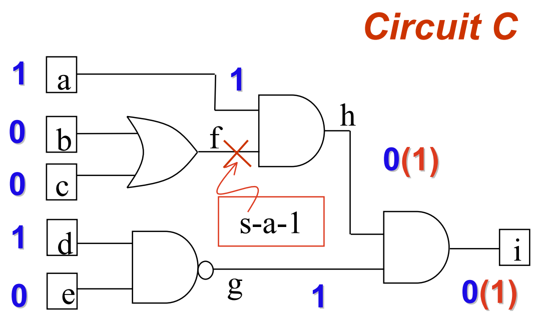
The ATPG problem

The CIRCUIT-SAT problem

The Boolean Satisfiability (SAT) problem



The ATPG problem

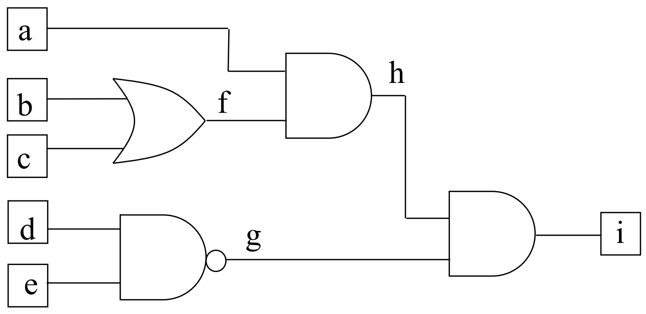


- A logic circuit
- A fault point
- A fault value

$\psi = f, s-a-1$

Does there exist a value assignment to the primary inputs which distinguishes the faulted and correct circuits ?

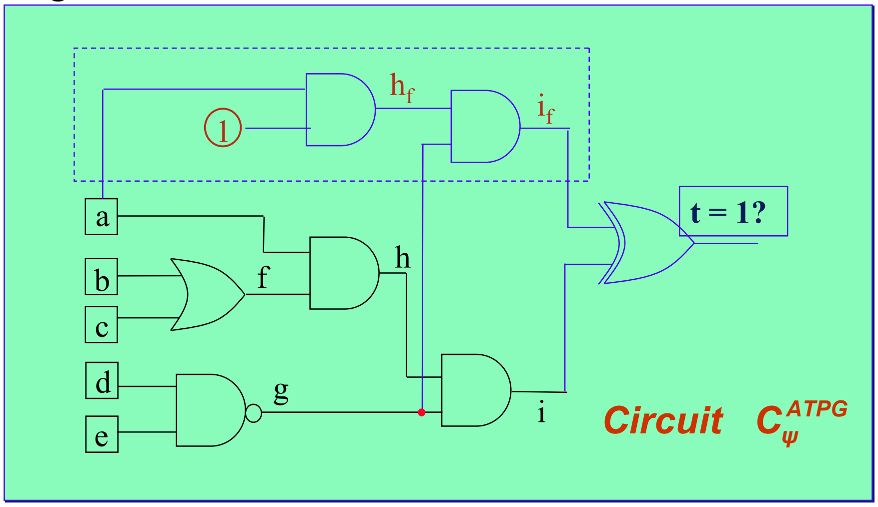
The CIRCUIT-SAT problem



Does there exist a value assignment to the primary inputs which causes at least one primary output to assume logic value '1' ?

ATPG as a CIRCUIT-SAT problem

Can we find an input value in which the faulty circuit and the good circuit differ?



The Boolean Satisfiability (SAT) problem

Given a formula, f :

- ❖ Defined over a set of variables, V *(a,b,c)*
- ❖ Comprised of a conjunction of clauses *(C_1, C_2, C_3)*
- ❖ Each clause is a disjunction of literals of the variables V

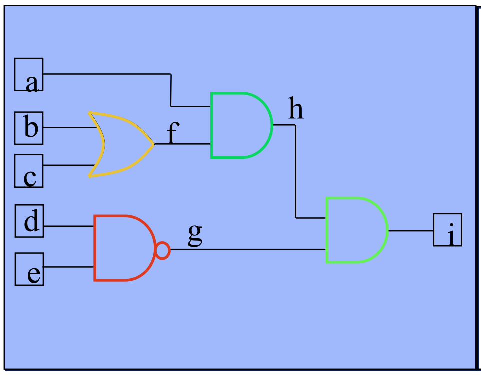
Does there exist an assignment of Boolean values to the variables, V which sets at least one literal in each clause to '1' ?

Example : $\underbrace{(a+b+\overline{c})}_{C_1}\underbrace{(\overline{a}+c)}_{C_2}\underbrace{(a+\overline{b}+c)}_{C_3}$ *a=b=c=1*

CIRCUIT-SAT as a SAT problem

A set of clauses representing the functionality of each gate

A unit literal *(i)* clause asserting the output to be ***'1'***



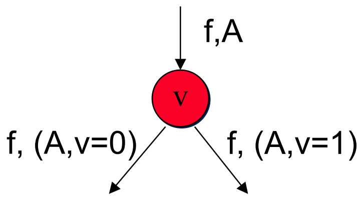
$$(\bar{b}+f)(\bar{c}+f)(b+c+\bar{f})$$
$$(d+g)(e+g)(\bar{d}+\bar{e}+\bar{g})$$
$$(a+\bar{h})(f+\bar{h})(\bar{a}+\bar{f}+h)$$
$$(h+\bar{i})(g+\bar{i})(\bar{h}+\bar{g}+i)$$
$$(i)$$

Algorithm for SAT [DPLL-62]

We have reduced ATPG to SAT- but then what?

Given : CNF formula $f(v_1,v_2,..,v_k)$, and an ordering function Next_Variable

```
Is_SAT(f, A)
{
   if Check_SAT(f, A) return SAT
   if Check_UNSAT(f,A) return UNSAT
   v = Next_Variable(f, A)
   if Is_SAT(f, (A,v=0)) return SAT
   if Is_SAT(f, (A,v=1)) return SAT
   return UNSAT
}
```



DPLL Algorithm – Unit Clause Rule

- Unit Literal Propagation rule (Boolean Constraint Propagation, BCP)

$$(\underset{\underset{0}{\|}}{a} + \underset{\underset{0}{\|}}{b} + c) \Rightarrow \boxed{c = 1}$$

DPLL Algorithm – Pure Literal Rule

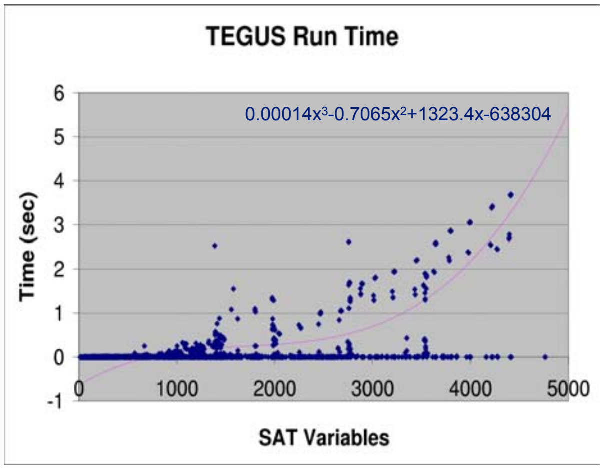
- Pure-Literal rule: a

$$\begin{array}{ll} (a + \ldots) & ☑\ (\overline{a} + \ldots) \\ (a + \ldots) & ☑\ (\overline{a} + \ldots) \\ \vdots & \quad \vdots \\ (a + \ldots) & ☑\ (\overline{a} + \ldots) \end{array} \Rightarrow \boxed{\begin{array}{c} \text{ASSIGN a = 1} \\ \text{SKIP a = 0} \end{array}}$$

Tegus Performance on Real Circuits

Results :

- Of the 11,000 instances generated, 90% were solved in less than 1/100th of a second
- The remaining exhibited roughly a cubic growth in execution time



Why is ATPG Easy (although NP-Complete)?

Current Status on Manufacture Test

- Practical approach to test: use scan – achieve 99%+ stuck-at coverage
- Single stuck-at-fault testing for combinational logic is a ``solved problem''
  - Despite the fact that it is NP-complete
  - After 20+ years of research
  - Results applied to combinational-equivalence checking
- Single stuck-at-fault testing for sequential circuits is an intractable problem
  - Despite the fact that it is also (only) NP-complete
  - Even after 20+ years of research
  - Time-frame expansion used in state-space search
- Principal research focus is on ATPG for enhanced fault models
  - Delay fault testing
- Other approaches
  - BIST

PODEM

```
PODEM(po, lvalue) {

        jlist = po with logical value lvalue;

        status = SEARCH_1(jlist);
        return(status);
}
```

SEARCH_1(jlist)

```
SEARCH_1(jlist)
{
        if (length of jlist is zero) return SUCCEED;

        if (BACKTRACE(po, po_value, &pi, &pi_value) == FALSE)
                return(FAILED);
        if(IMPLY(pi, pi_value, jlist) != IMPLY_CONFLICT) {
                search_status = SEARCH_1(JLIST);
                if (search_status == FAILED) {
                        restore the state of the network to what it was
                        prior to the most recent primary input assignment;
                        search_status = SEARCH_2(jlist, pi, 1 – pi_value);
                }
        } else {
                restore the state of the network;
                search_ status = SEARCH_2(jlist, pi, 1 – pi value);
        }
        return(search_status);
}
```

SEARCH_2(jlist)

```
SEARCH_2(jlist, pi, pi_value)
{
        backtracks = backtracks + 1;
        if (backtracks > BACKTRACK_LIMIT) return(ABORTED);
                if(IMPLY(pi, pi_value, jlist) != IMPLY_CONFLICT) {
                search_status = SEARCH_1(jlist);
                if (search_status == FAILED) {
                        restore the state of the network;
        } else {
                search_status = FAILED;
                restore the state of the network;
        }
        return(search_status);
}
```