ATG for SSFs in Combinatorial Circuits

Traian Pop
trapo@ida.liu.se

Sorin Manolache
sorma@ida.liu.se

Outline

- Introduction
- Deterministic Test Generation
  - Fault-Oriented ATG
  - Fault Independent ATG
- Random Test Generation
  - Combined Deterministic and Random Test Generation
- ATG Systems
- Conclusions

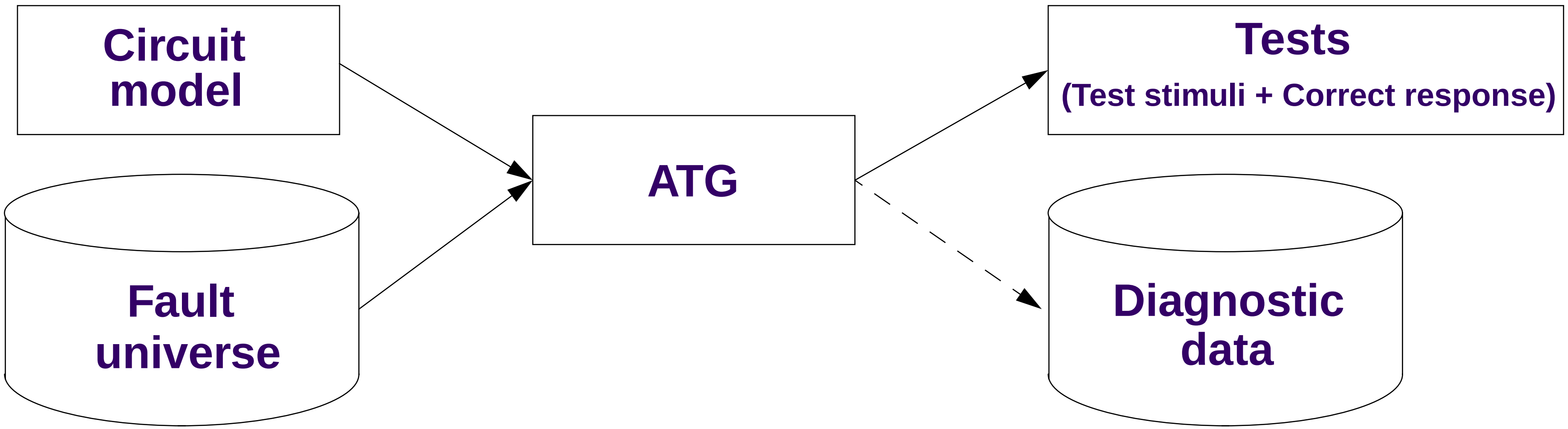
Testing

- off-line
- edge-pin
- stored-pattern
- full comparison of the output results

General problems for TG:

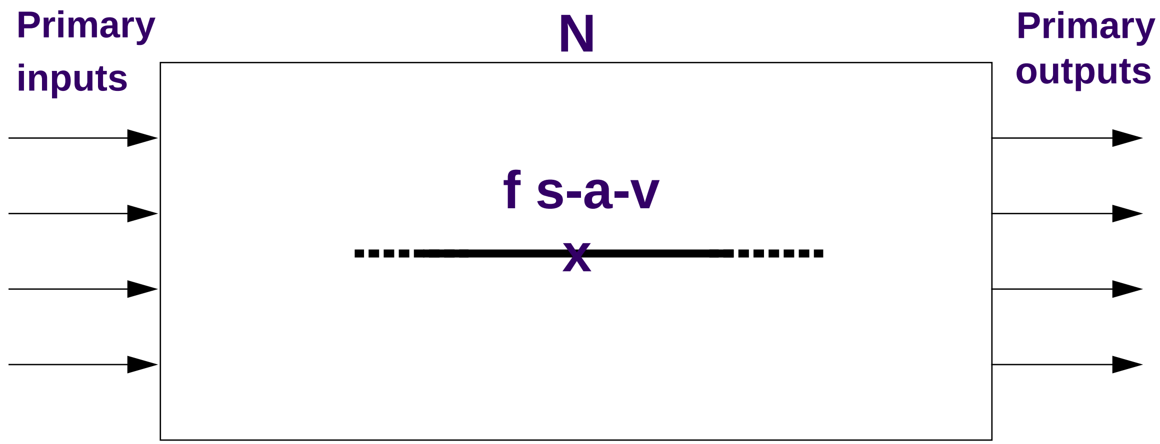
- the cost of generating the test
- the quality of the generated test
- the cost of applying the test

Deterministic Test Generation



- manual / **automatic**
- **fault-oriented / fault-independent**

- targeted at certain fault within a fault universe given by a fault model

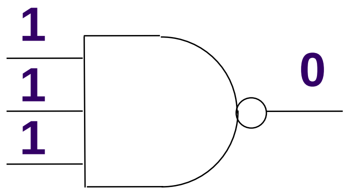


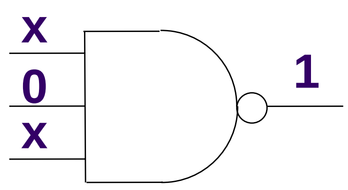
Problems:

- fault activation
- error propagation

Fault-Oriented ATG (cont’d)

Fault activation: line-justification (recursively justifying the value of a gate output by values of the gate inputs until primary inputs are reached)

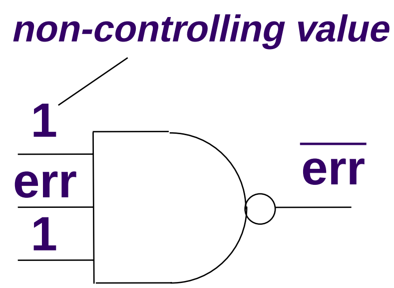




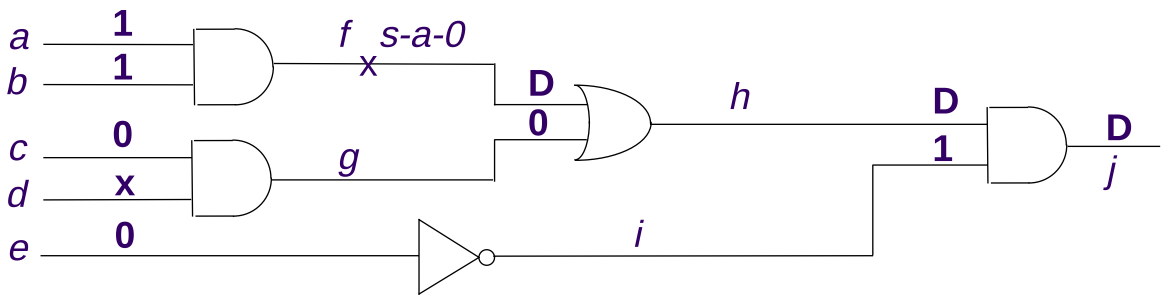
Error propagation:

- **composite logic values**
- **reduced to a set of line-justification problems**

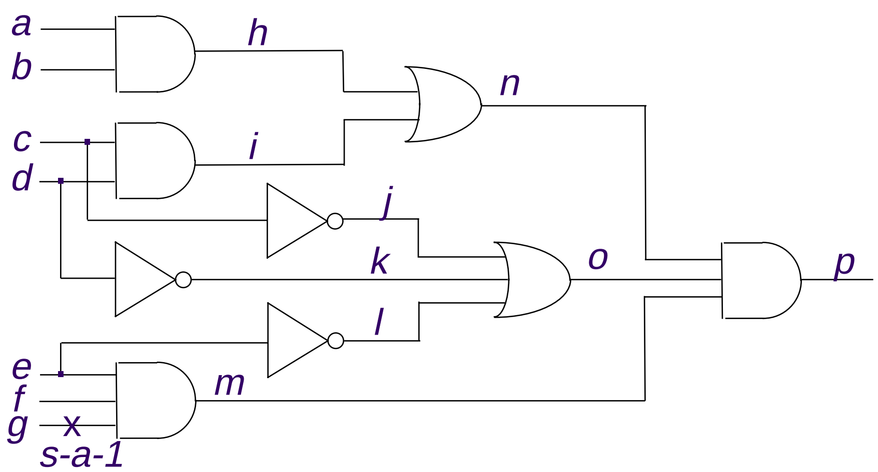
v/v_f	
0/0	0
1/1	1
0/1	$\overline{D}$
1/0	D



Fanout-free circuits:



Circuits with fanout:



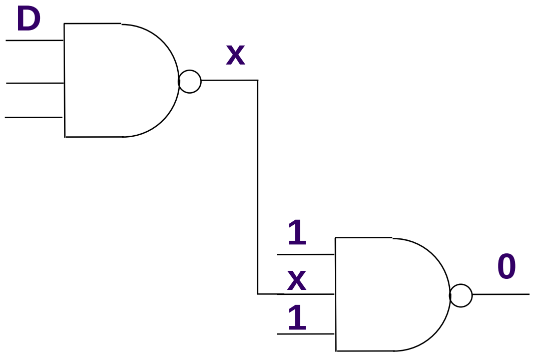
Decisions	Implications	
	$g = \overline{D}$ $e = 1$ $f = 1$ $m = \overline{D}$ $n = 1$ $o = 1$ $l = 0$ $p = \overline{D}$	Initial implications
$i = 1$	$c = 1$ $d = 1$ $j = 0$ $k = 0$ $o = 0$	To justify $n = 1$ Contradiction
$h = 1$	$a = 1$ $b = 1$	To justify $n = 1$
$o = 1$	$c = 0$ $k = 0$	To justify $o = 1$

Fault-Oriented ATG (cont'd)

```
Solve()
begin
    if(Imply_and_check() = FAILURE) then return FAILURE
    if(error at PO and all lines are justified)
        then return SUCCESS
    if(error can't be propagated to a PO)
        then return FAILURE
    select an unsolved problem
    repeat
        begin
            select one untried way to solve it
            if(Solve() = SUCCESS) then return SUCCESS
        end
    until all ways to solve it have been tried
    return FAILURE
end
```

D-frontier:

- used during error propagation process (D-drive)
- becomes void if no error can be propagated to a PO



J-frontier:

- keeps track of unjustified lines

Minimizing the number of incorrect decisions:

- **maximum implications principle**
- global implications
- reversing incorrect decisions
- error-propagation look-ahead

Fault-Oriented ATG (cont'd)

Algorithms:

- the D algorithm
- the 9-V algorithm
- single-path sensitization
- PODEM
- FAN
- etc.

```
D-alg()
begin
    if(Imply_and_check() = FAILURE) then return FAILURE
    if(error not at PO) then begin
        if(D-frontier = void) then return FAILURE
        repeat
            begin
                select an untried gate (G) from D-frontier
                c = controlling value of G
                assign c̄ to every input of G with value x
                if(D-alg() = SUCCESS) then return SUCCESS
            end
        until all gates from D-frontier have been tried
        return FAILURE
    end
    if(J-frontier = void) then return SUCCESS
    select a gate (G) from J-frontier
    c = controlling value of G
    repeat
        begin
            select an input (j) of G with value x, assign c to j
            if(Solve() = SUCCESS) then return SUCCESS
        end
    until all inputs of G are specified
    return FAILURE
end
```

Fault-Oriented ATG (cont'd)

Principles used for decisions:

- attack the most difficult problem first
- try the easiest solution first

Measures:

- controllability: the relative difficulty of setting a line to a value
- observability: the relative difficulty of propagating an error from a line to a PO

Cost functions:

- distance-based functions
- recursive cost functions
- fanout-based cost functions

$$C0(l) = min\{C0(i)\} + f_l - 1$$
$$C1(l) = \sum\{C1(i)\} + f_l - 1$$

Fault-Independent ATG

Goal: to produce a set of tests that detect a large set of SSFs without targeting individual faults

Critical-path TG algorithm:

- select a PO and assign it a critical value
- recursively justify any critical value on a gate output by critical values on the gate inputs

! critical values are used instead of primitive values (complete specification)

Fault-Independent ATG (cont'd)

Critical-path TG for circuits with reconvergent fanout:

- conflicts
- self-masking
- multiple-path sensitization
- overlap among PO cones

Fault-oriented TG vs. Fault-independent TG:

- fault-oriented TG needs an additional simulation step to detect SSFs
- using critical-path TG, new tests can be generated starting from already existing ones
- fault-independent algorithms cannot identify undetectable faults

Random Test Generation

- do not target a particular fault

- random test vectors are applied and one hopes to detect as many faults as possible

Comparison

- advantages:
    - low test generation cost
    - test vectors usually generated on-the-fly (no need for test vector storage)
- disadvantages:
    - long test sequence (approx. 10 times longer than deterministically generated tests)
    - high test application cost

Quality Measures

- ***test quality*** – t_N – the probability to detect *all* possible faults after applying N random tests
- ***N-step detection probability of f*** – d^f_N – the probability to detect fault *f* after applying N random tests
- ***detection quality*** – d_N – the probability to detect the most difficult to detect fault after applying N random tests

$$d_N = \min_f d^f_N$$

$$t_N < d_N$$

Test Length (1)

- usually, one is interested in how long a test sequence should be in order to achieve a detection quality (test quality) of c
- simulation too expensive
- if N tests detect the most difficult to detect fault with probability c, then they will detect another fault with a probability $\geq$ c

Test Length (2)

- $d_N \geq c$, N = ?

  $d^f_1 = |T_f| / 2^n$, for uniformly distributed input test vectors

  T_f – the set of test vectors that detect the fault *f*

  $d_{min} = \min_f d^f_1$, the lowest detection probability among the SSFs in the circuit

  $1 - (1 - d_{min})^N \geq c$

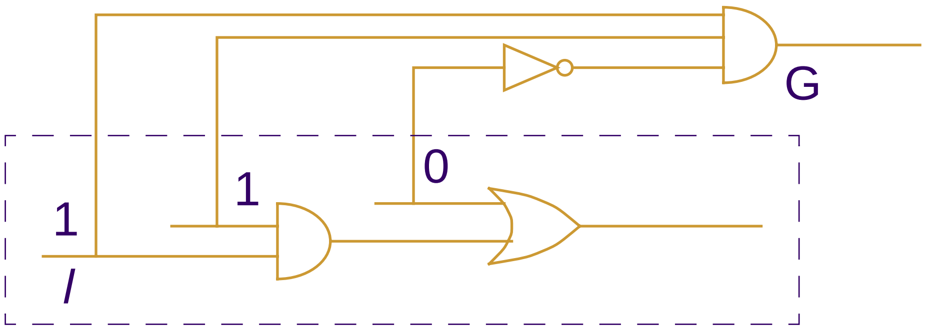
  $N \geq \text{ceil}(\ln(1 - c) / \ln(1 - d_{min}))$

- $t_N \geq c$, N = ?

  $N \geq \text{ceil}((\ln(1 - c) - \ln(k)) / \ln(1 - d_{min}))$, k – number of faults with detection probability $d_{min} \leq d \leq 2d_{min}$

- ***d_{min} has to be determined***

- $d_{min} \geq 1 / 2^n \Rightarrow$ exhaustive testing
- the probability to detect *l* s-a-0 = the probability of G being 1



- there may be several paths to propagate the error

$$d_{min} \geq P(G_k = 1) \quad (max?)$$

- P(l = 1) is computed in linear time for fanout-free circuits, otherwise exponential
- probability intervals and cuts can be used instead of fixed probabilities $\Rightarrow$ linear time

Difficult Faults

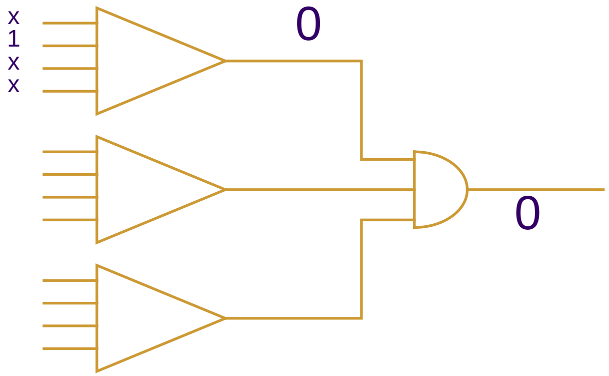
- sometimes N_{max} is bounded (fixed) (testing time)
- having a desired detection quality c, one can deduce a lower bound, d_L
- then the circuit has to be checked if there are faults with $d_f \leq d_L$, the *difficult faults*
- among checkpoint faults (checkpoint = primary input or fanout branch)
- if found, then modify the circuit in such a way that the difficult fault becomes easier to detect (design for testability)

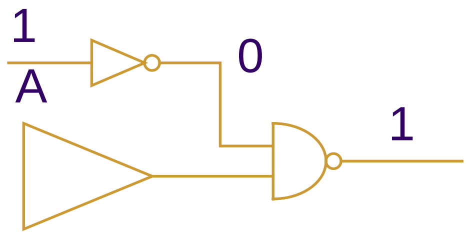
Non-Uniformly Distributed Test Vectors

- turned out to be better for some circuits for various optimization goals (coverage, testing cost, cost of DFT modifications)
- research for computing the pdf of the test vectors
- *adaptive RTG*, monitors the test generation process, gathers statistical data about the most successful test vectors and adjusts then the pdf of future test vectors

Combined Deterministic/Random TG

- **RAPS (Random Path Sensitization)**





half of generated tests will have A = 1

- **SMART (Sensitizing Method for Algorithmic Random Testing)**

- requirements:
  - fault coverage ↑
  - test generation cost ↓
  - test set size ↓
- should it collapse faults?

ATG Systems Structure

```
repeat
   Generate_test(t);
   fault simulate t
   v = value(t)
   if acceptable(v) then add t to the test
until endphase1();
                     /* what about test scheduling? */
                     /* redundancy elimination is important */
repeat
   select a new target fault f /* better one close to the PI */
   try to generate a new test t for f
   if successful
      add f to the test
      fault simulate f
      discard the faults detected by f
   fi
until endphase2();
```

- static compaction

01x	011	010
0x1		
0x0	0x0	001
x01	x01	

- dynamic compaction
  - try to set the unspecified PIs such that additional faults are detected
  - biggest problem is the selection of the additional fault to be tested

Other TG Methods

- algebraic – impractical
- extensions for tristate logic – problem: how to avoid the simultaneous enabling of multiple bus drivers
- TG for module-level circuits
  - modules assumed to be fault free
  - propagation and justification procedures more complicated, derived from the module's function
  - if the modules are not fault free, either they are tested before (design for testability) or after by replacing one module at a time with its gate level model