Υπολογιστικά Συστήματα Υψηλής Αξιοπιστίας

Παραγωγή Δοκιμής

Δρ. Γκάμας Βασίλειος

Επιστημονικός Συνεργάτης
vgkamas@uniwa.gr

Πανεπιστήμιο Δυτικής Αττικής
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Σκοπός παρουσίασης

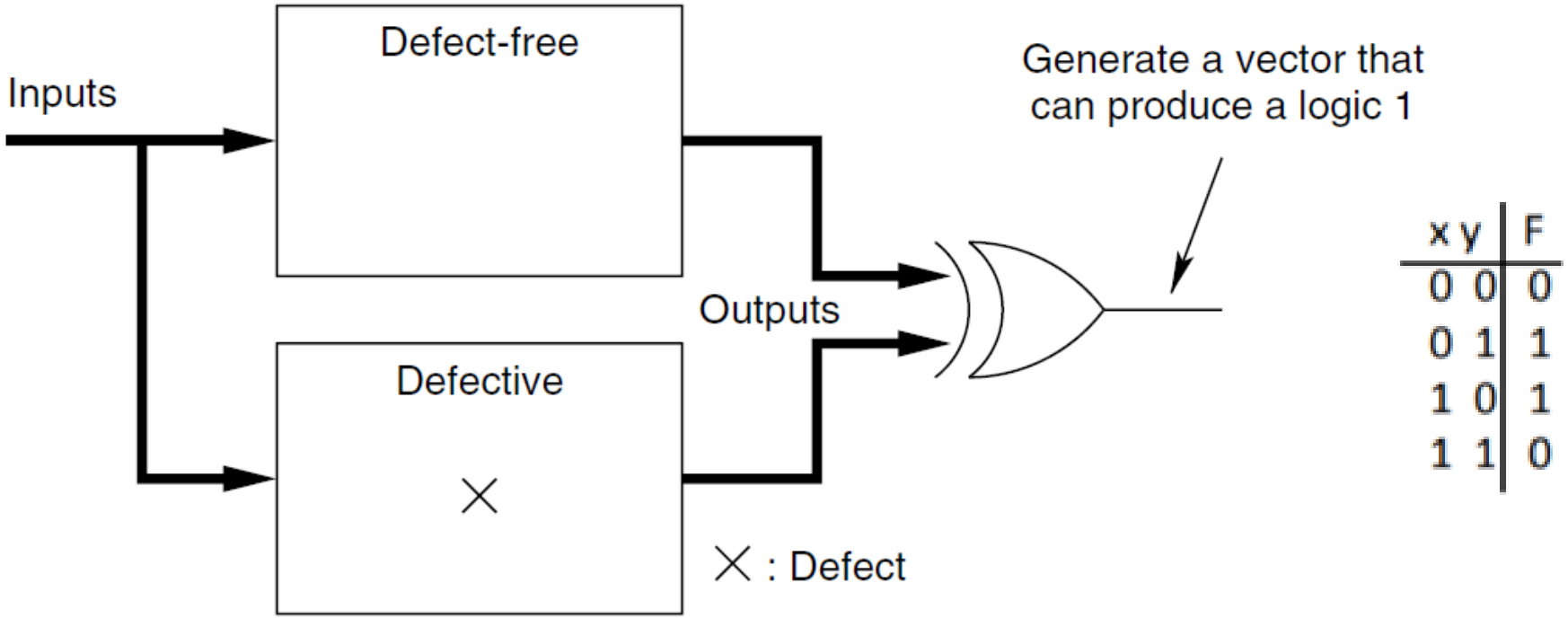
- Να εισάγει τους φοιτητές στις βασικές έννοιες και τεχνικές αυτόματης παραγωγής διανυσμάτων δοκιμής

Εισαγωγή

- Η παραγωγή δοκιμής (test generation) αποτελεί ουσιώδες στοιχείο του ελέγχου ορθής λειτουργίας κυκλωμάτων VLSI
  - Αποδοτικό ATPG μπορεί να μειώσει δραστικά το υψηλό κόστος του DFT
  - Στόχος της παραγωγής δοκιμής είναι ο προσδιορισμός ενός μικρού συνόλου αποδοτικών διανυσμάτων δοκιμής με μικρό υπολογιστικό κόστος
- Η διαδικασία ATPG είναι ιδιαίτερα απαιτητική
  - Εκθετική πολυπλοκότητα
  - Τα μεγέθη των κυκλωμάτων εξακολουθούν να αυξάνονται (Νόμος Moore)
  - Υψηλές συχνότητες ρολογιού
    - Ανάγκη για έλεγχο όχι μόνο δομικών ελαττωμάτων αλλά και σφαλμάτων καθυστέρησης

Αρχή λειτουργίας ATPG

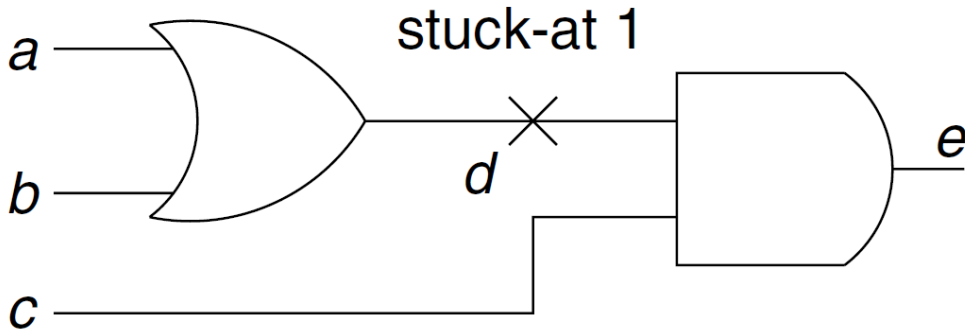
- Δημιουργία ενός διανύσματος εισόδου (δοκιμής) το οποίο είναι ικανό να ξεχωρίσει ένα κύκλωμα χωρίς ελαττώματα από ένα ελαττωματικό



x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Απλό παράδειγμα ATPG

- Ας υποθέσουμε το σφάλμα d/1 στο παρακάτω ελαττωματικό κύκλωμα
- Απαιτείται ο διαχωρισμός της εξόδου του κυκλώματος με ελαττώματα από το κύκλωμα χωρίς ελαττώματα
- Ανάγκες
  - Να θέσουμε το d ίσο με 0 στο κύκλωμα χωρίς ελαττώματα ορίζοντας τις κατάλληλες τιμές στις εισόδους της πύλης OR
  - Να διαδώσουμε το αποτέλεσμα του σφάλματος στην έξοδο
- Διάνυσμα: abc=001 (έξοδος fault-free κυκλώματος = 0, έξοδος faulty κυκλώματος = 1)



Ένα τυπικό ATPG σύστημα

- Με δεδομένο ένα κύκλωμα και ένα μοντέλο σφάλματος:

  Repeat
  Generate a test for each undetected fault
  Drop all other faults detected by the test using a fault simulator
  Until all faults have been considered

- Σημείωση 1: ενδέχεται ένα σφάλμα να μην μπορεί να ελεγχθεί (untestable) – να μην μπορεί να δημιουργηθεί κάποιο test
- Σημείωση 2: ένα σύστημα ATPG μπορεί να παρακάμψει τον έλεγχο ενός σφάλματος (aborted fault), εάν οι πόροι του συστήματος ξεπεράσουν ένα συγκεκριμένο όριο

Τυχαία παραγωγή δοκιμής

- Η πιο απλή τεχνική παραγωγής δοκιμής: Παράγονται N τυχαία διανύσματα δοκιμής
- Παράγονται τυχαία λογικές τιμές στις κύριες εισόδους ενός υπό εξέταση κυκλώματος με την πιθανότητα εκχώρησης ενός λογικού 0 ή 1 να είναι ισοδύναμη
- Η στάθμη εμπιστοσύνης (level of confidence) που κάποιος μπορεί να έχει σε ένα τυχαίο σύνολο διανυσμάτων δοκιμής T μετριέται ως η πιθανότητα το T να μπορεί να ανιχνεύσει όλα τα σφάλματα μόνιμης τιμής στο κύκλωμα
- Για N τυχαία διανύσματα δοκιμής, η ποιότητα της δοκιμής t_N εκφράζει την πιθανότητα ανίχνευσης όλων των ανιχνεύσιμων σφαλμάτων μόνιμης τιμής από τα N τυχαία διανύσματα δοκιμής

Τυχαία παραγωγή δοκιμής - Πιθανότητα ανίχνευσης σφαλμάτων (1/2)

- Έστω ένα κύκλωμα *n* κύριων εισόδων
- T_f: σύνολο διανυσμάτων δοκιμής που μπορούν να ανιχνεύσουν το σφάλμα *f*.
- Πιθανότητα το σφάλμα *f* να μπορεί να ανιχνευθεί από ένα τυχαίο διάνυσμα δοκιμής

$$d_f = \frac{T_f}{2^n}$$

- Πιθανότητα ένα τυχαίο διάνυσμα δοκιμής να μην ανιχνεύσει το σφάλμα *f*

$$e_f = 1 - d_f$$

Τυχαία παραγωγή δοκιμής - Πιθανότητα ανίχνευσης σφαλμάτων (2/2)

- Πιθανότητα *N* τυχαία διανύσματα δοκιμής να μην ανιχνεύσουν το σφάλμα *f*

$$e_f^N = \left(1 - d_f\right)^N$$

- Πιθανότητα να ανιχνεύσει το σφάλμα *f* τουλάχιστον ένα από τα *N* τυχαία διανύσματα ελέγχου

$$1 - \left(1 - d_f\right)^N$$

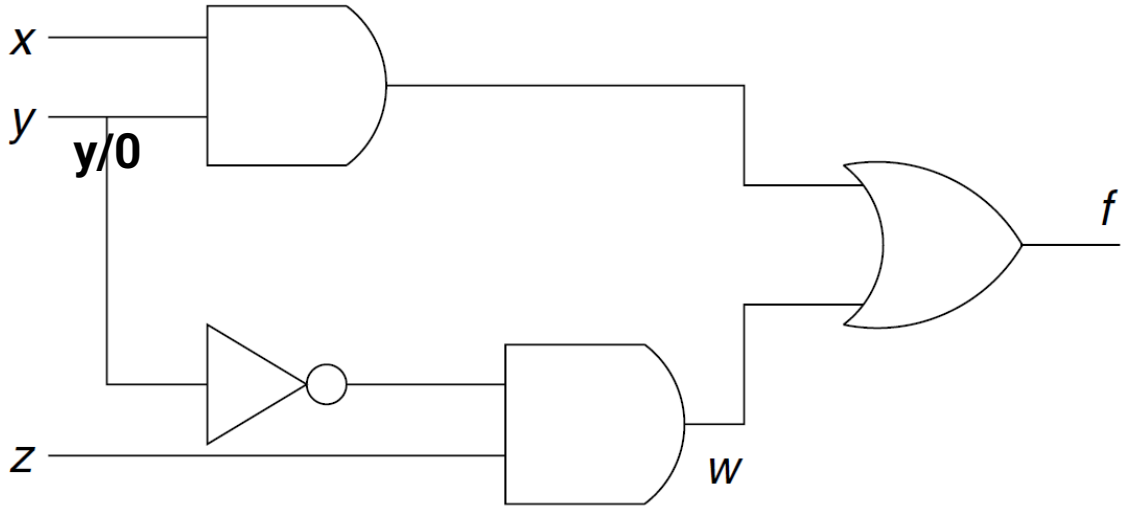
Εξαντλητική παραγωγή δοκιμής

- Εξαντλητική δοκιμή
  - Εφαρμογή 2^n διανυσμάτων δοκιμής σε ένα υπό εξέταση συνδυαστικό κύκλωμα n εισόδων
  - Εξασφαλίζει την ανίχνευση όλων των ανιχνεύσιμων σφαλμάτων στα συνδυαστικά κυκλώματα
  - Ο χρόνος δοκιμής ενδέχεται να είναι απαγορευτικά μεγάλος εάν το πλήθος των κύριων εισόδων του κυκλώματος είναι μεγάλο
  - Εφικτή μόνο σε μικρά κυκλώματα
- Ψευδό-εξαντλητική δοκιμή
  - Κατάτμηση του κυκλώματος επιμέρους τμήματα (κώνους)
  - Εφαρμογή εξαντλητικής δοκιμής σε κάθε τμήμα

Διαφορά Boole (1/3)

- Διαφορά Boole: Τεχνική παραγωγής διανυσμάτων δοκιμής

- Έστω το παρακάτω κύκλωμα με συνάρτηση εξόδου $f = xy + \bar{y}z$
- Έστω το σφάλμα μόνιμης τιμής y/0 και $f' = f(y = 0)$ η έξοδος του ελαττωματικού κυκλώματος.
- Σκοπός της παραγωγής δοκιμής είναι η εύρεση ενός διανύσματος δοκιμής για το οποίο να ισχύει η σχέση $f' \oplus f = 1$



Διαφορά Boole (2/3)

- Η σχέση $f' \oplus f = 1$ ισχύει όταν το f και f' έχουν αντίθετες λογικές τιμές.
- Οποιοδήποτε διάνυσμα ελέγχου μπορεί να θέσει $f' \oplus f = 1$ μπορεί να παράξει αντίθετες τιμές ανάμεσα στις εξόδους του κυκλώματος με σφάλμα και του κυκλώματος χωρίς σφάλμα.
- Ορισμός διαφοράς Boole

$$\frac{df}{dy} = f(y = 1) \quad f(y = 0)$$

  - Αναφέρεται σε μια έξοδο f ενός λογικού κυκλώματος ως προς μια είσοδο y

Διαφορά Boole (3/3)

- Για να μπορέσει να ανιχνευθεί το σφάλμα μόνιμης τιμής y/ν (δηλαδή *y stuck-at-0* ή *a stuck-at-1*) θα πρέπει
  - Να διεγερθεί το σφάλμα μόνιμης τιμής θέτοντας $y = \bar{\nu}$,
  - Το σύνολο όλων των διανυσμάτων δοκιμής που μπορούν να μεταδώσουν το αποτέλεσμα του σφάλματος στην έξοδο f να ικανοποιούν την σχέση $\frac{df}{dy} = 1$
- Η επίλυση της παραπάνω εξίσωσης δίνει τις τιμές των κυρίων εισόδων πού επιτρέπουν την παρατηρησημότητα ενός απλού σφάλματος μόνιμης τιμής που έχει συμβεί σε μία γραμμή *y* του κυκλώματος
- Τα διανύσματα δοκιμής τότε δίνονται από την επίλυση των σχέσεων:

$$y\frac{df}{dy} = 1 \text{ για σφάλμα stuck at 0}$$

$$\bar{y}\frac{df}{dy} = 1 \text{ για σφάλμα stuck at 1}$$

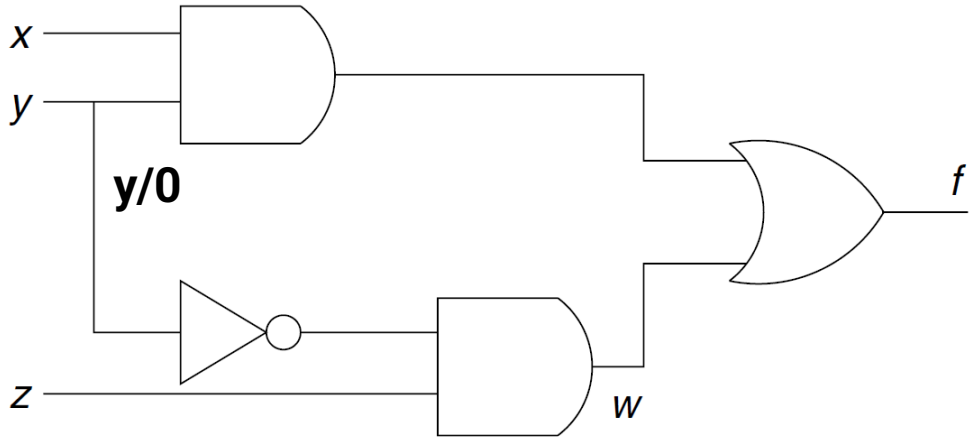
Διαφορά Boole – Παράδειγμα #1

- Έστω το σφάλμα μόνιμης τιμής y/0. Για να διεγερθεί το σφάλμα θέτουμε y=1, οπότε έχουμε:

$$y \cdot \frac{df}{dy} = 1 \Rightarrow y \cdot [f(y=1) \oplus f(y=0)] = 1 \Rightarrow = y \cdot (x \oplus z) = 1 \Rightarrow$$

$$y \cdot (x\bar{z} + \bar{x}z) = 1 \Rightarrow xy\bar{z} + \bar{x}yz = 1$$

- Το σφάλμα μπορεί να ανιχνευθεί από τα διανύσματα xyz = 110 ή xyz = 011



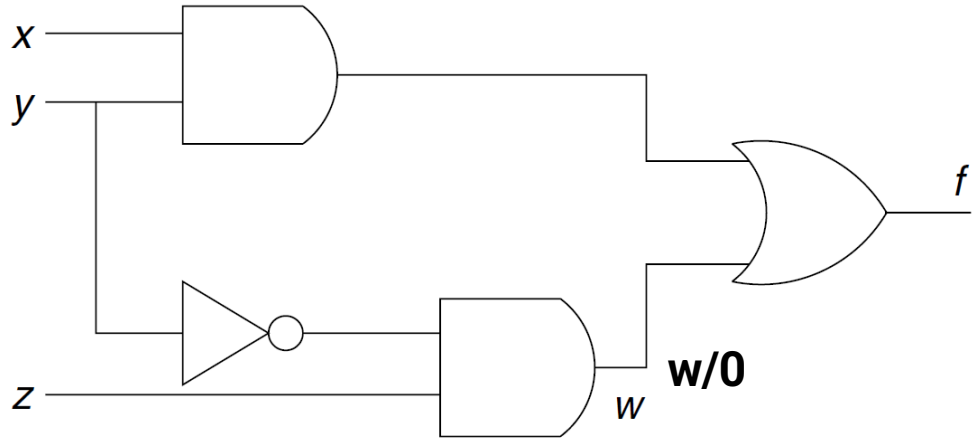
Διαφορά Boole – Παράδειγμα #2

- Έστω το σφάλμα μόνιμης τιμής w/0. Για να διεγερθεί το σφάλμα θέτουμε w=1, οπότε έχουμε:

$$w \cdot \frac{df}{dw} = 1 \Rightarrow w \cdot [f(w=1) \oplus f(w=0)] = 1 \Rightarrow w \cdot [1 \oplus xy] = 1 \Rightarrow$$

$$w \cdot \overline{xy} = 1 \Rightarrow w \cdot (\bar{x} + \bar{y}) = 1 \Rightarrow w\bar{x} + w\bar{y} = 1 \Rightarrow \bar{y}z\bar{x} + \bar{y}z\bar{y} = 1 \Rightarrow \bar{x}\bar{y}z + \bar{y}z = 1 \Rightarrow \bar{y}z = 1$$

- Το σφάλμα μπορεί να ανιχνευθεί από τα διανύσματα xyz = 001 ή xyz = 101



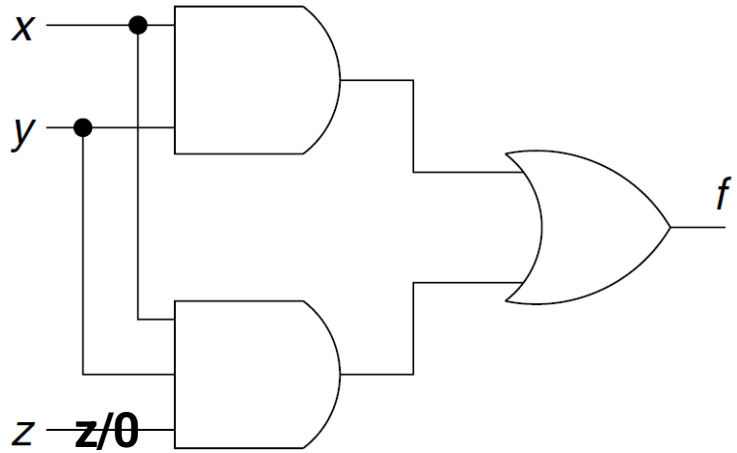
Διαφορά Boole – Παράδειγμα #3

- Έστω το σφάλμα μόνιμης τιμής z/0. Για να διεγερθεί το σφάλμα θέτουμε z=1, οπότε έχουμε:

$$z \cdot \frac{df}{dz} = 1 \Rightarrow z \cdot [f(z=1) \oplus f(z=0)] = 1 \Rightarrow z \cdot [xy \oplus xy] = 1 \Rightarrow$$

$$z \cdot 0 = 1 \rightarrow \text{ΑΔΥΝΑΤΟ}$$

- **Το σφάλμα z/0 δεν μπορεί να δοκιμαστεί (untestable)!!!**

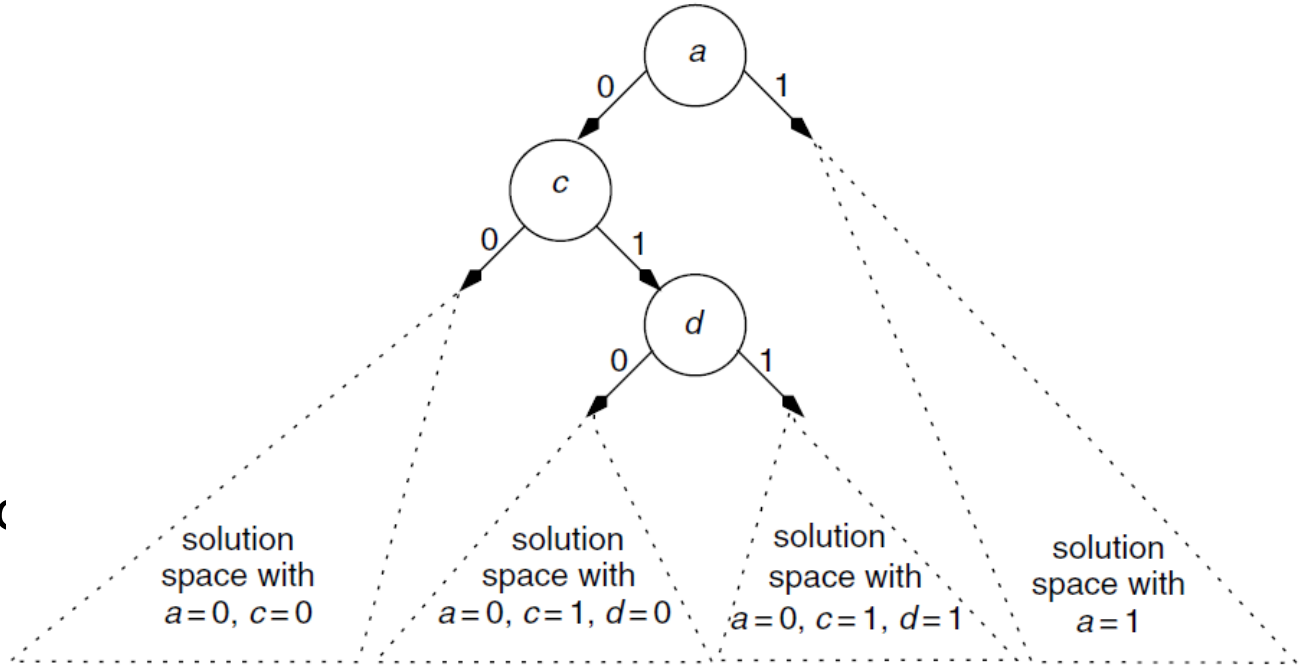


Ντετερμινιστικό ATPG

- Σε γενικές γραμμές δεν χρειαζόμαστε ένα πλήρες σύνολο διανυσμάτων δοκιμής τα οποία μπορούν να ανιχνεύσουν ένα σφάλμα.
- Αντίθετα θέλουμε να υπολογίσουμε ένα διάνυσμα δομικής γρήγορα.
- Αντί να χρησιμοποιήσουμε την διαφορά Boole για να υπολογίσουμε όλα τα διανύσματα δοκιμής, χρησιμοποιούμε την τεχνική αναζήτησης branch-and-bound για να βρούμε ένα διάνυσμα δοκιμής γρήγορα
- Το ντετερμινιστικό ATPG έχει δύο στόχους
  - Να διεγείρει ένα σφάλμα
  - Να μεταδώσει το αποτέλεσμα του σφάλματος σε μία έξοδο του κυκλώματος

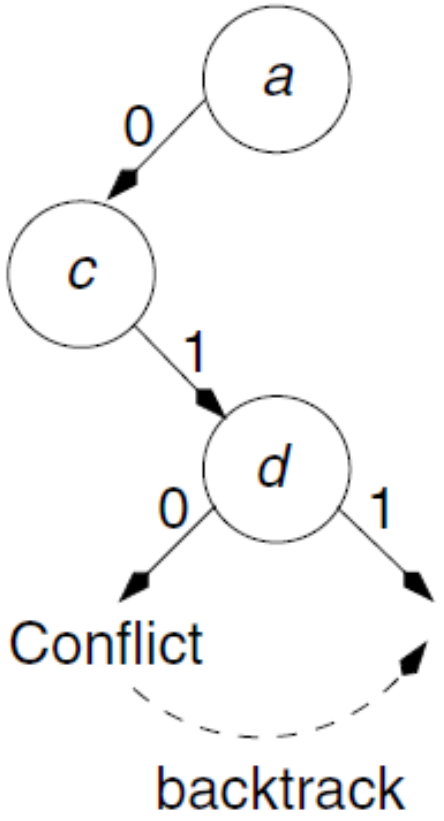
Δέντρο απόφασης στην τεχνική branch-and-bound

- Η τεχνική ATPG πραγματοποιεί συστηματικά αναζήτηση σε έναν χώρο αναζήτησης.
- Branching - Σε κάθε επίπεδο του δέντρου επιλέγεται ποια είσοδος θα τεθεί σε ποια τιμή
- Bounding – Αποφεύγεται η αναζήτηση μεγάλων τμημάτων του δέντρου απόφασης περιορίζοντας τον χώρο αναζήτησης
- a, c, d: κύριοι είσοδοι ενός λογικού κυκλώματος
- Ένα σφάλμα μπορεί να ανιχνευθεί χωρίς να απαιτηθεί να εξεταστούν όλες οι αποφάσεις.



Οπισθοδρόμηση στο δέντρο απόφασης

- Η τεχνική ATPG πραγματοποιεί κάθε φορά αναζήτηση σε ένα branch του δέντρου
- Στην περίπτωση που εμφανιστεί κάποιο conflict (το συγκεκριμένο διάνυσμα δεν μπορεί να ανιχνεύσει το σφάλμα) γίνεται backtrack στην προηγούμενη απόφαση



- If d=0 causes a conflict, backtrack to d=1
- If d=1 also causes a conflict, backtrack to c=0

ATPG για fanout-free κυκλώματα (1/7)

- Έστω ένα fanout-free συνδυαστικό κύκλωμα c με σφάλμα μόνιμης τιμής g/v (g-stuck-at-v)

Algorithm 2 Basic Fanout Free ATPG (C, g/v)

```
1: initialize circuit by setting all values to X;
2: JustifyFanoutFree(C, g, v̄); /* excite the fault by justifying line g to v̄ */
3: PropagateFanoutFree(C, g); /* propagate fault-effect from g to a PO */
```

- Οι συναρτήσεις JustifyFanoutFree( ) και PropagateFanoutFree ( ) είναι επανηληπτικές

ATPG για fanout-free κυκλώματα (2/7)

Η συνάρτηση JustifyFanoutFree( )

Algorithm 3 JustifyFanoutFree(C, g, v)

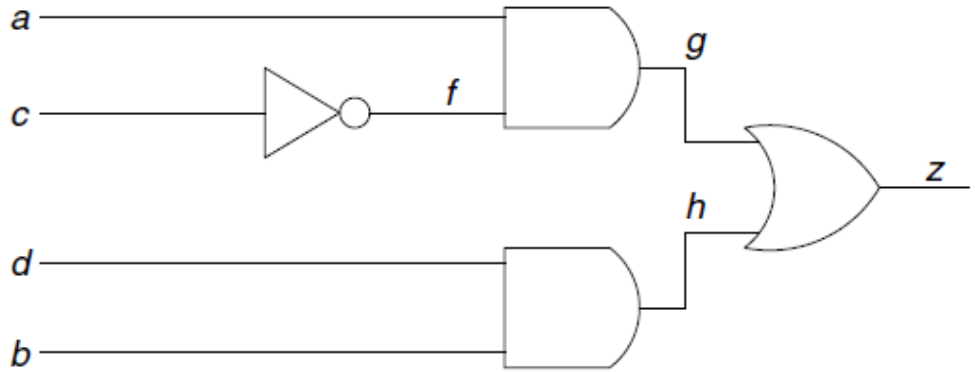
```
 1: g = v;
 2: if gate type of g == primary input then
 3:    return;
 4: else if gate type of g == AND gate then
 5:    if v == 1 then
 6:       for all inputs h of g do
 7:          JustifyFanoutFree(C, h, 1);
 8:       end for
 9:    else {v == 0}
10:       h = pick one input of g whose value == X;
11:       JustifyFanoutFree(C, h, 0);
12:    end if
13: else if gate type of g == OR gate then
14:    ...
15: end if
```

ATPG για fanout-free κυκλώματα (3/7)

- Έστω το κύκλωμα C με το σφάλμα g/0
- Κλήσεις στην συνάρτηση JustifyFanoutFree ( )

Algorithm 3 JustifyFanoutFree(C, g, v)

```
 1: g = v;
 2: if gate type of g == primary input then
 3:    return;
 4: else if gate type of g == AND gate then
 5:    if v == 1 then
 6:       for all inputs h of g do
 7:          JustifyFanoutFree(C, h, 1);
 8:       end for
 9:    else {v == 0}
10:       h = pick one input of g whose value == X;
11:       JustifyFanoutFree(C, h, 0);
12:    end if
13: else if gate type of g == OR gate then
14:    ...
15: end if
```

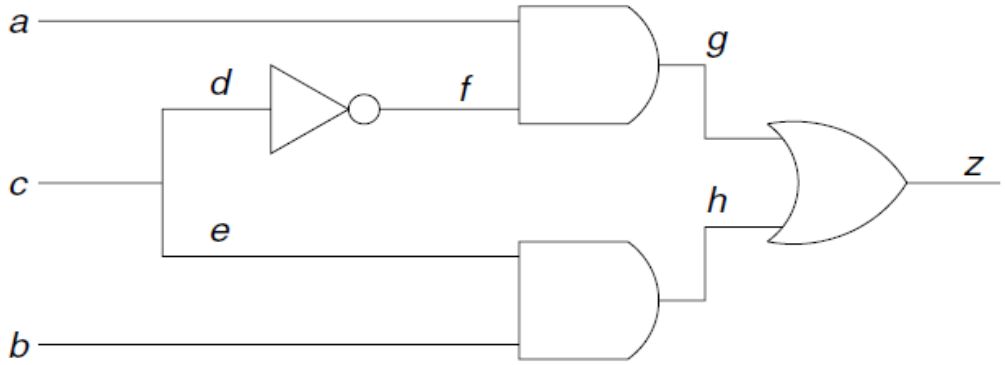


call #1: JustifyFanoutFree($C, g, 1$)
call #2: JustifyFanoutFree($C, a, 1$)
call #3: JustifyFanoutFree($C, f, 1$)
call #5: JustifyFanoutFree($C, c, 0$)

Μετά από αυτές τις κλήσεις στην συνάρτηση JustifyFanoutFree(), το acdb = 10XX αποτελεί ένα διάνυσμα εισόδου το οποίο μπορεί να θέσει την γραμμή g στην τιμή 1.

ATPG για fanout-free κυκλώματα (4/7)

- Ας υποθέσουμε ότι θέλουμε να θέσουμε το z=0.
- Σύμφωνα με τον αλγόριθμο JustifyFanoutFree() θα πρέπει να θέσουμε g = 0 and h = 0.
- Για να θέσουμε g = 0, υποθέτουμε ότι χρησιμοποιούμε την τιμή της εισόδου f οπότε θα πρέπει να θέσουμε c = 1
- Για να θέσουμε το h = 0, έστω ότι θέτουμε e=0.
- Ωστόσο δεν μπορούμε να χρησιμοποιήσουμε αυτήν την επιλογή καθώς θα δημιουργηθεί σύγκρουση (conflict) στο σήμα στην γραμμή c
- Οπότε έπρεπε να ληφθεί κάποια διαφορετική απόφαση για να θέσουμε το h = 0.
- Σε αυτήν την περίπτωση η επιλογή b = 0 θα έπρεπε να επιλεχθεί.
- Το συγκεκριμένο παράδειγμα δείχνει την δυνατότητα λήψης λάθος αποφάσεων οι οποίες μπορεί να οδηγήσουν σε πιθανά **backtracks** στην αναζήτηση.



ATPG για fanout-free κυκλώματα (5/7)

Algorithm 4 PropagateFanoutFree(C, g)

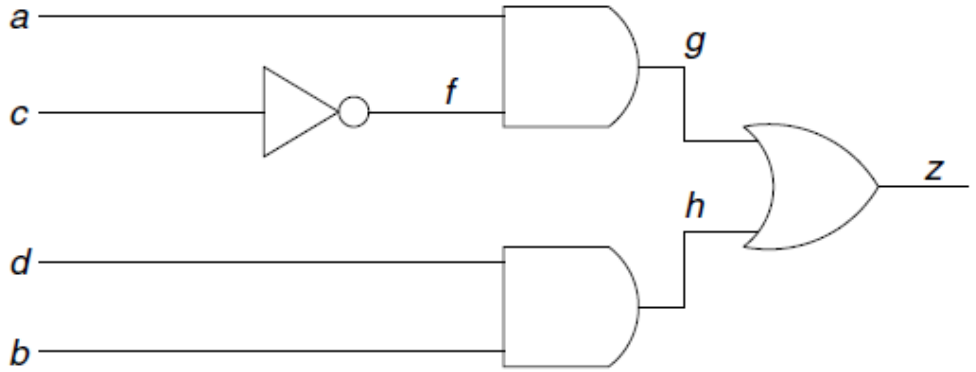
```
 1: if g has exactly one fanout then
 2:     h = fanout gate of g;
 3:     if none of the inputs of h has the value of X then
 4:         backtrack;
 5:     end if
 6: else {g has more than one fanout}
 7:     h = pick one fanout gate of g that is unjustified;
 8: end if
 9: if gate type of h == AND gate then
10:     for all inputs, j, of h, such that j ≠ g do
11:         if the value on j == X then
12:             JustifyFanoutFree(C, j, 1);
13:         end if
14:     end for
15: else if gate type of h == OR gate then
16:     for all inputs, j, of h, such that j ≠ g do
17:         if the value on j == X then
18:             JustifyFanoutFree(C, j, 0);
19:         end if
20:     end for
21: else if gate type of h == . . . gate then
22:     . . .
23: end if
24: PropagateFanoutFree(C, h);
```

ATPG για fanout-free κυκλώματα (6/7)

Algorithm 4 PropagateFanoutFree(C, g)

```
 1: if g has exactly one fanout then
 2:     h = fanout gate of g;
 3:     if none of the inputs of h has the value of X then
 4:         backtrack;
 5:     end if
 6: else {g has more than one fanout}
 7:     h = pick one fanout gate of g that is unjustified;
 8: end if
 9: if gate type of h == AND gate then
10:     for all inputs, j, of h, such that j ≠ g do
11:         if the value on j == X then
12:             JustifyFanoutFree(C, j, 1);
13:         end if
14:     end for
15: else if gate type of h == OR gate then
16:     for all inputs, j, of h, such that j ≠ g do
17:         if the value on j == X then
18:             JustifyFanoutFree(C, j, 0);
19:         end if
20:     end for
21: else if gate type of h == . . . gate then
22:     . . .
23: end if
24: PropagateFanoutFree(C, h);
```

Διάδοση του αποτελέσματος του σφάλματος g/0 από το g στο z



call #1: PropagateFanoutFree(C, g)
call #2: JustifyFanoutFree(C, h, 0)
call #3: JustifyFanoutFree(C, b, 0)
call #4: PropagateFanoutFree(C, z)

Διάνυσμα ελέγχου acb = 100

ATPG για fanout-free κυκλώματα (7/7)

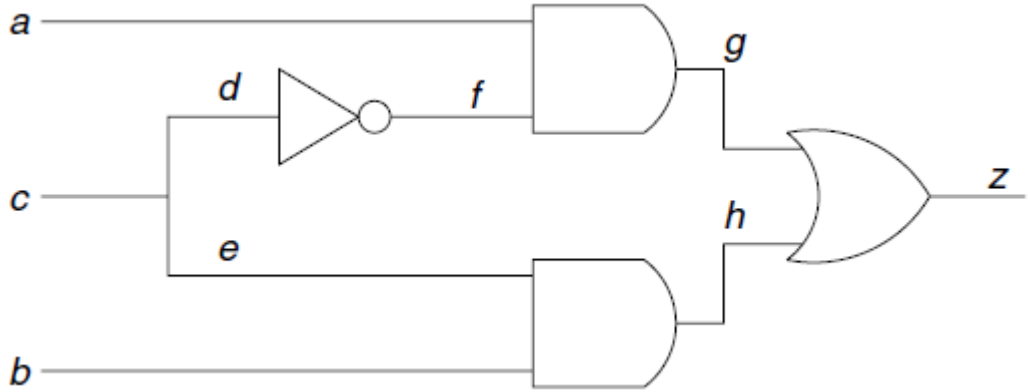
- Για την δημιουργία διανυσμάτων ελέγχου για συνδυαστικά κυκλώματα θα πρέπει να υπάρχει ένας μηχανισμός ο οποίος θα επιτρέπει το ATPG να αποφεύγει συγκρούσεις καθώς και να μπορεί να διαχειριστεί τις συγκρούσεις όταν εμφανίζονται.
- Για να συμβεί αυτό το αντίστοιχο δέντρο απόφασης θα πρέπει να κατασκευάζεται κατά την διάρκεια της αναζήτησης ενός διανύσματος ελέγχου και θα πρέπει να γίνονται backtracks σε περίπτωση εμφάνισης μιας σύγκρουσης.

ATPG για κυκλώματα με fanout

- Έστω το κύκλωμα C με το σφάλμα g/0
- Η συνάρτηση JustifyFanoutFree( ) λειτουργεί και στην περίπτωση κυκλωμάτων με fanout

Algorithm 3 JustifyFanoutFree(C, g, v)

```
1: g = v;
2: if gate type of g == primary input then
3:    return;
4: else if gate type of g == AND gate then
5:    if v == 1 then
6:       for all inputs h of g do
7:          JustifyFanoutFree(C, h, 1);
8:       end for
9:    else {v == 0}
10:      h = pick one input of g whose value == X;
11:      JustifyFanoutFree(C, h, 0);
12:   end if
13: else if gate type of g == OR gate then
14:   ...
15: end if
```



call #1: JustifyFanoutFree($C, g, 1$)
call #2: JustifyFanoutFree($C, a, 1$)
call #3: JustifyFanoutFree($C, f, 1$)
call #4: JustifyFanoutFree($C, d, 0$)
call #5: JustifyFanoutFree($C, c, 0$)

Μετά από αυτές τις κλήσεις στην συνάρτηση JustifyFanoutFree(), το abc = 1X0 αποτελεί ένα διάνυσμα εισόδου το οποίο μπορεί να θέσει την γραμμή g στην τιμή 1.

Αλγοριθμική εξαγωγή διανυσμάτων δοκιμής

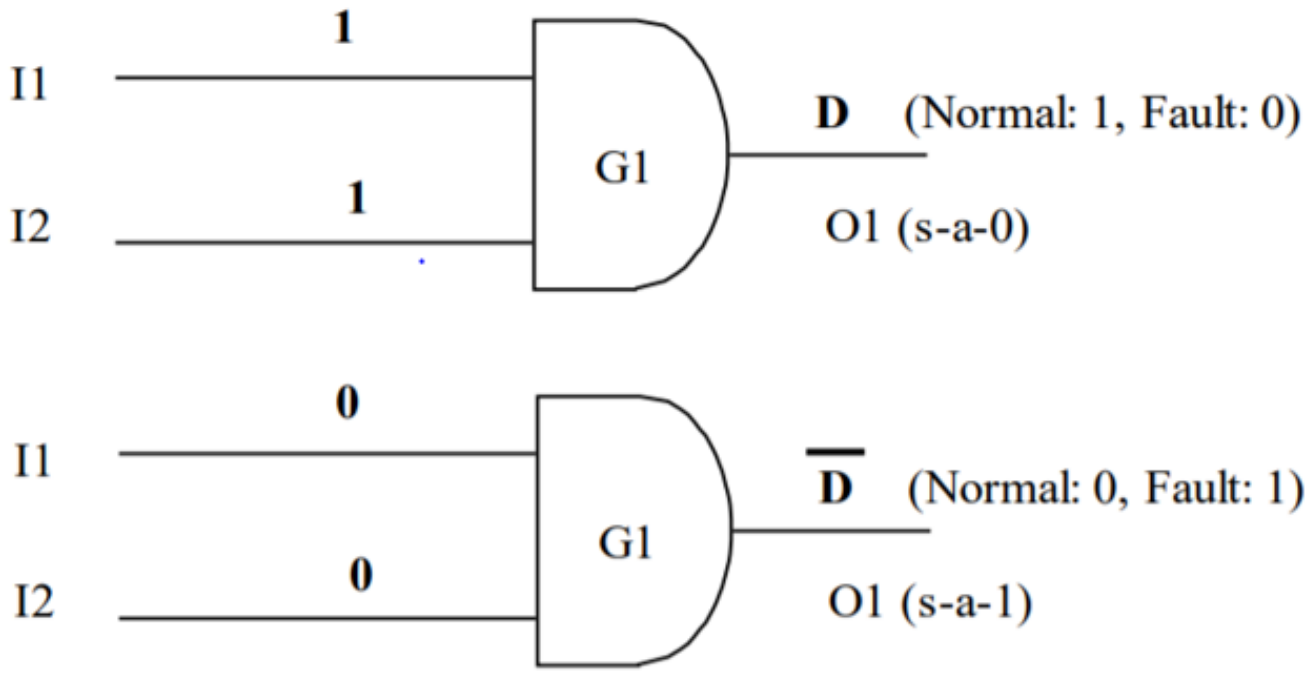
- D-Algorithm
- PODEM
- FAN

5-valued άλγεβρα για συνδυαστικά κυκλώματα

- Αντί να χρησιμοποιήσουμε δύο κυκλώματα (fault-free και faulty), επιλύουμε το ATPG πρόβλημα σε ένα κύκλωμα.
- Για να το πετύχουμε, κάθε τιμή του σήματος θα πρέπει να μπορεί να συλλέξει ταυτόχρονα τις fault-free και faulty τιμές
- 5-Value Algebra: *0, 1, X, D,* $\overline{D}$

Symbol	Implication	Fault-free circuit	Faulty circuit
0	(0/0)	0	0
1	(1/1)	1	1
X	(X/X)	X	X
D	(1/0)	1	0
$\overline{D}$	(0/1)	0	1

5-valued άλγεβρα για συνδυαστικά κυκλώματα



Boolean τελεστές στην 5-valued άλγεβρα

AND	0	1	D	$\overline{D}$	X
0	0	0	0	0	0
1	0	1	D	$\overline{D}$	X
D	0	D	D	0	X
$\overline{D}$	0	$\overline{D}$	0	$\overline{D}$	X
X	0	X	X	X	X

OR	0	1	D	$\overline{D}$	X
0	0	1	D	$\overline{D}$	X
1	1	1	1	1	1
D	D	1	D	1	X
$\overline{D}$	$\overline{D}$	1	1	$\overline{D}$	X
X	X	1	X	X	X

NOT	
0	1
1	0
D	$\overline{D}$
$\overline{D}$	D
X	X

Παραδείγματα

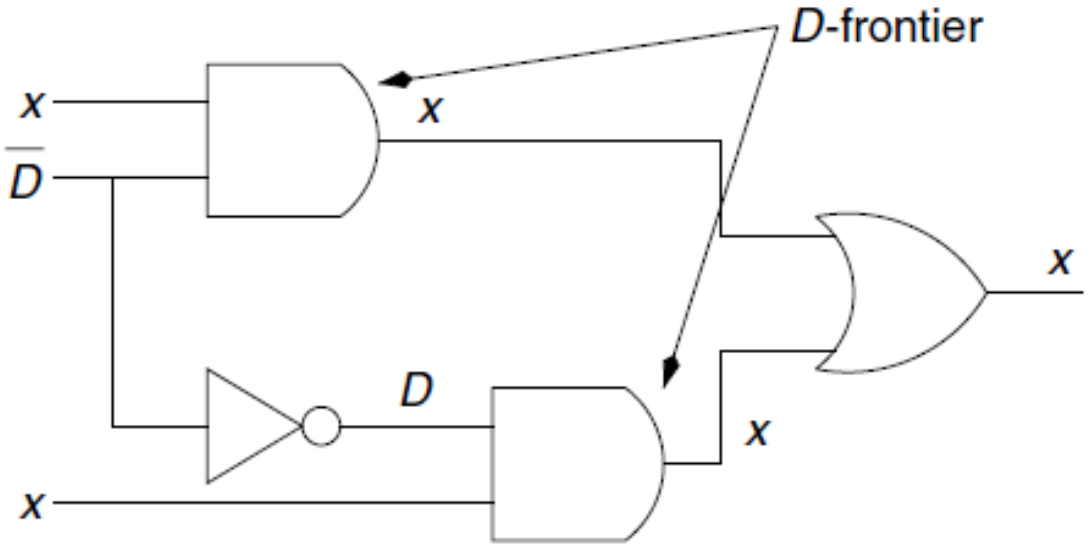
- 1 AND D = D
  - Το 1 AND 1/0 περιλαμβάνει 2 υπολογισμούς: (i) 1 AND 1=1 και (ii) 1 AND 0=0.
- 0 OR $\overline{D}$= $\overline{D}$
  - Το 0 OR 0/1 περιλαμβάνει 2 υπολογισμούς: (i) 0 OR 0=0, (ii) 0 OR 1=1.

D-Algorithm

- Αλγόριθμος ATPG για συνδυαστικά κυκλώματα με εσωτερικές διακλαδώσεις (fanout structure)
- Βασική ιδέα του αλγορίθμου: μετάδοση του αποτελέσματος ενός σφάλματος στην κύρια έξοδο ενός κυκλώματος διατηρώντας πάντα ένα μη-κενό D-frontier
- Αρχικά εκχωρείται η τιμή X σε όλες οι γραμμές του κυκλώματος, εκτός από την γραμμή στην οποία εμφανίζεται το σφάλμα στην οποία εκχωρείται η τιμή D ή $\overline{D}$

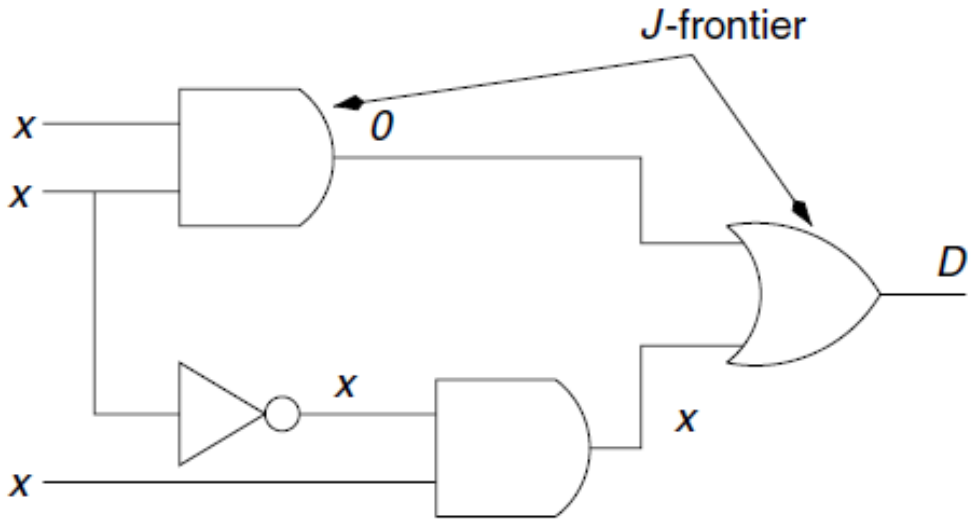
D-Frontier στον D-Algorithm

- D-Frontier: Περιλαμβάνει όλες τις πύλες των οποίων η έξοδος έχει την τιμή X και τουλάχιστον ένα D *ή* $\overline{D}$ στις εισόδους τους
- Αρχικά το D-Frontier αποτελείται από 1 πύλη (έξοδος του fault-site)



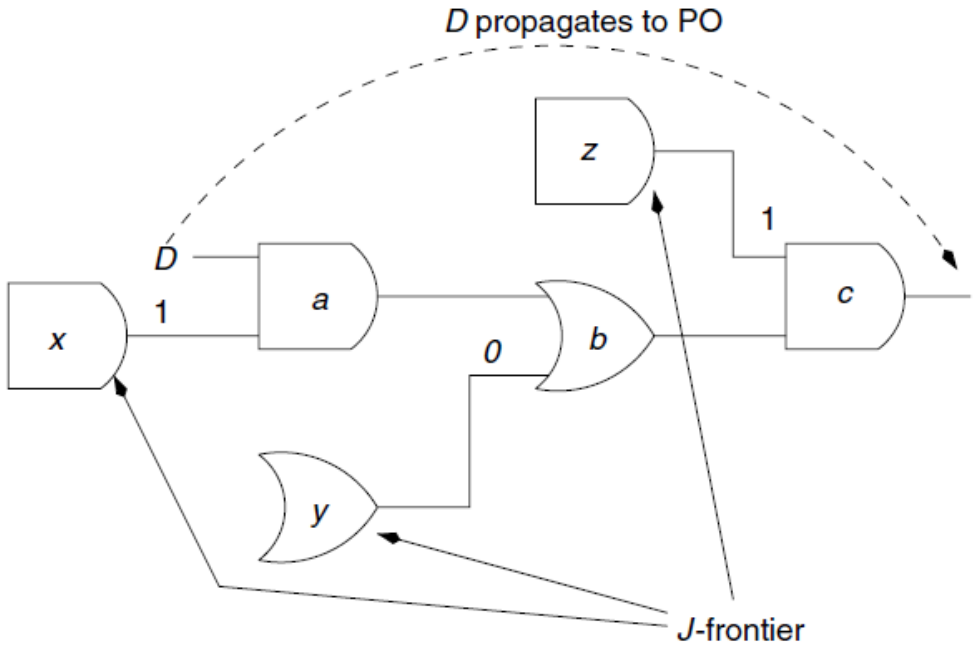
J-Frontier στον D-Algorithm

- J-Frontier: Περιλαμβάνει όλες τις πύλες των οποίων οι έξοδοι έχουν καθορισμένη τιμή (δεν είναι X) αλλά δεν έχουν γίνει justify από τις εισόδους



Λειτουργία D-Algorithm

- Έστω το σφάλμα stack at 0 στην είσοδο της πύλης a.
  - Στην συγκεκριμένη είσοδο εκχωρείται η τιμή D
- Για να μεταδοθεί το D στην κύρια έξοδο του κυκλώματος προσθέτουμε στο J-frontier τις πύλες που πρέπει να ελεγχθούν σε μία συγκεκριμένη τιμή
  - Στο συγκεκριμένο παράδειγμα θα πρέπει να θέσουμε σε non-controlling τιμές τις εξόδους των πυλών x, y, z

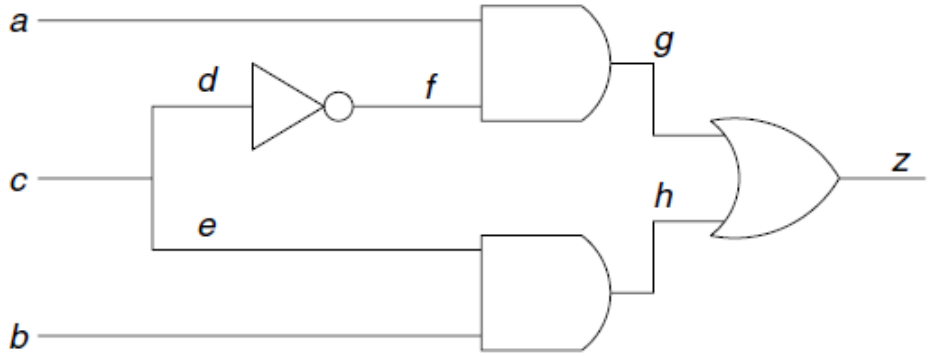


Λειτουργία D-Algorithm

- Όταν το D *ή* $\overline{D}$ φτάσει σε μία κύρια έξοδο, όλες οι πύλες που ανήκουν στο J-frontier θα πρέπει να γίνουν justify (να ελεγχθούν οι τιμές των εισόδων τους) χρησιμοποιώντας την τεχνική αναζήτησης branch-and-bound
- Αυτό γίνεται ενημερώνοντας το J-frontier backward.
- Κάθε φορά που εμφανίζεται μια σύγκρουση γίνεται οπισθοχώρηση.
- Σε κάθε βήμα θα πρέπει το D-frontier να ελέγχεται έτσι ώστε το D *ή* $\overline{D}$ να έχει μεταδοθεί σε μία κύρια έξοδο

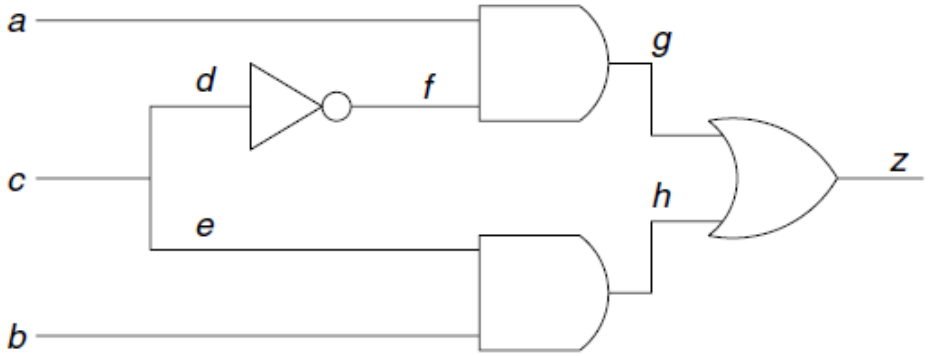
1ο παράδειγμα λειτουργίας D-Algorithm

- Έστω ένα stuck-at-1 σφάλμα στην γραμμή f
- Σε όλες τις γραμμές του κυκλώματος εκτός της γραμμής f, εκχωρείται η τιμή X
- Στην γραμμή f εκχωρείται η τιμή $\overline{D}$
- D-frontier={g}, J-frontier={f= $\overline{D}$ }
- Για να μεταδοθεί το αποτέλεσμα του σφάλματος από την γραμμή f στην γραμμή εξόδου z θέτουμε a=1 και h=0
- J-frontier={a=1, h=0, f= $\overline{D}$ }
- Θα πρέπει να γίνουν justify όλες οι γραμμές στο J-frontier



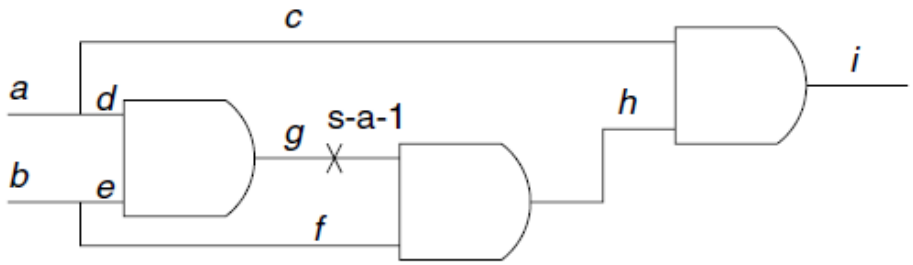
1ο παράδειγμα λειτουργίας D-Algorithm

- Επειδή το a αποτελεί κύρια είσοδο έχει γίνει ήδη justify
- Για να εκχωρηθεί στο f η τιμή $\overline{D}$ (f= $\overline{D}$) πρέπει να θέσουμε d=1
- Για να εκχωρηθεί στο h η τιμή 0 (h=0) υπάρχουν 2 επιλογές
  - e=0. Δεν είναι δυνατή επιλογή (d=1, e=0). Ο αλγόριθμος κάνει backtrack
  - b=0. Δυνατή επιλογή
- Προκύπτει το διάνυσμα δοκιμής abc={1,0,1}



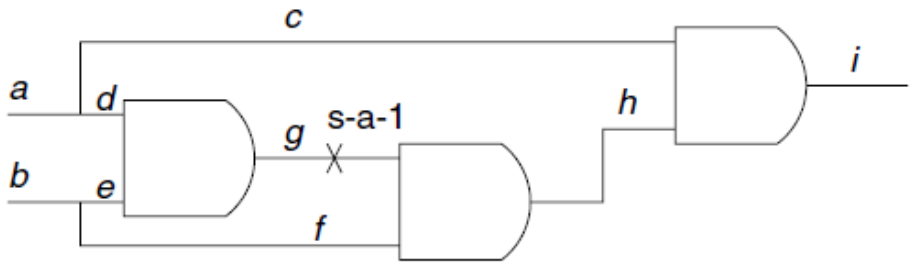
2ο παράδειγμα λειτουργίας D-Algorithm

- Έστω ένα stuck-at-1 σφάλμα στην γραμμή g
- Σε όλες τις γραμμές του κυκλώματος εκτός της γραμμής g, εκχωρείται η τιμή X
- Στην γραμμή g εκχωρείται η τιμή $\overline{D}$
- D-frontier={h}, J-frontier={g= $\overline{D}$ }
- Για να μεταδοθεί το αποτέλεσμα του σφάλματος από την γραμμή g στην γραμμή εξόδου i θέτουμε c=1 και f=1
- J-frontier={c=1, f=1, g= $\overline{D}$ }
- Θα πρέπει να γίνουν justify όλες οι γραμμές στο J-frontier



2ο παράδειγμα λειτουργίας D-Algorithm

- Για να εκχωρηθεί στο g η τιμή $\overline{D}$ (g= $\overline{D}$) υπάρχουν 2 επιλογές
  - a=0. Σε αυτή την περίπτωση c=0. Δεν είναι δυνατόν, γίνεται οπισθοδρόμηση οπότε a=1
  - b=0. Υπάρχει σύγκρουση καθώς το f=1
- Το σφάλμα g/1 δεν μπορεί να δοκιμαστεί (untestable)

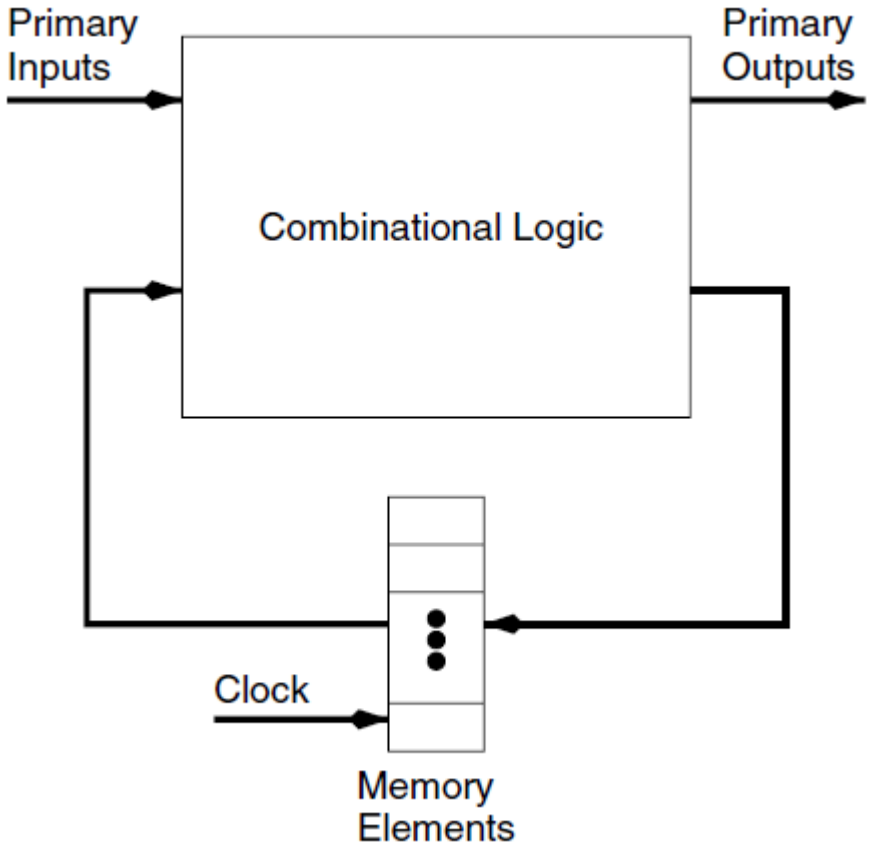


PODEM και FAN

- PODEM
  - Στηρίζεται στην τεχνική αναζήτησης branch-and-bound
  - Αποφάσεις λαμβάνονται μόνο για τις κύριες εισόδους του κυκλώματος
  - Χρησιμοποιείται μόνο το D-frontier (μπορεί να είναι κενό)
- FAN
  - Παραλλαγή του PODEM για βελτιωμένο ATPG
  - Χρησιμοποιεί headlines για την μείωση του αριθμού των αποφάσεων που πρέπει να ληφθούν

ATPG σε ακολουθιακά κυκλώματα

- Η παραγωγή δοκιμής για ακολουθιακά κυκλώματα παρουσιάζει αρκετές ομοιότητες με αυτήν για συνδυαστικά κυκλώματα
- Ωστόσο ένα διάνυσμα δοκιμής μπορεί να μην είναι αρκετό για να ανιχνεύσει ένα σφάλμα, καθώς η ενεργοποίηση και η διάδοση του σφάλματος μπορεί να απαιτήσει τα flip flop να λάβουν κάποιες συγκεκριμένες τιμές



Ακολουθιακό ATPG framework

- Στηρίζεται στο συνδυαστικό ATPG
- Στοχεύει 1 time frame κάθε φορά
- Το ίδιο σφάλμα εμφανίζεται σε όλα τα time frames
- Διέγερση σφάλματος στο time frame 0 και διάδοσή του σε μία κύρια έξοδο μέσω πολλαπλών time frames
- Justification της κατάστασης που απαιτείται στο time frame 0 μέσω πολλαπλών time frames
- Το ακολουθιακό ATPG είναι αρκετά πολύπλοκο καθώς οι οπισθοχωρήσεις μπορεί να αφορούν την αλλαγή αποφάσεων που λήφθηκαν για διαφορετικά time frames

Καθορισμός σφαλμάτων που δεν μπορούν να δοκιμαστούν (untestable faults)

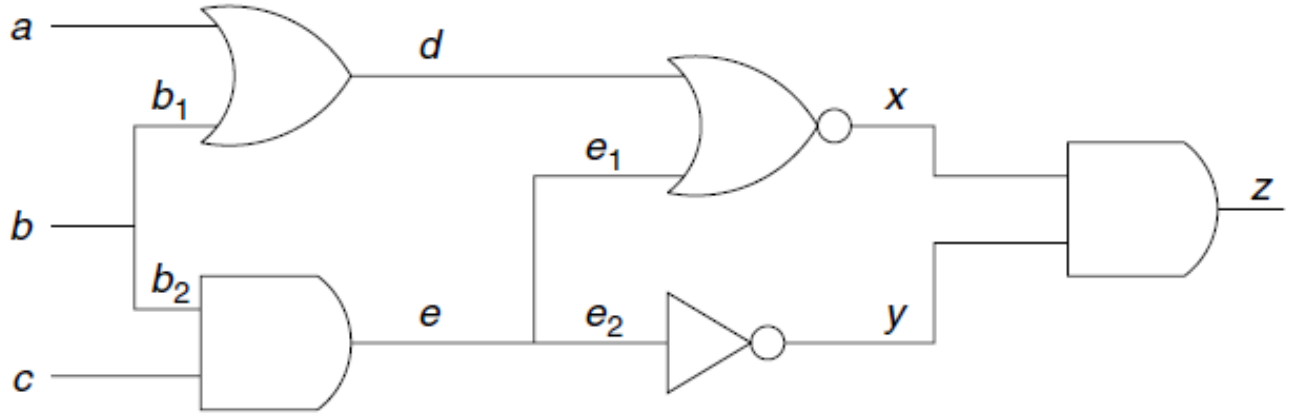
- Untestable faults
  - Εκείνα τα οποία δεν μπορούν να διεγερθούν
  - Εκείνα τα οποία δεν μπορούν να μεταδοθούν
  - Εκείνα τα οποία δεν μπορούν ταυτόχρονα να διεγερθούν ή να μεταδοθούν
- Το ATPG μπορεί να αφιερώσει πολύ χρόνο για την δημιουργία μιας δοκιμής για ένα untestable σφάλμα
- Ο γρήγορος καθορισμός των untestable σφαλμάτων επιτρέπει στο ATPG να παρακάμψει αυτά τα σφάλματα
- Στα συνδυαστικά κυκλώματα μπορεί να εμφανιστούν untestable faults λόγω πλεονασμού στο κύκλωμα
- Στα ακολουθιακά κυκλώματα μπορεί να εμφανιστούν untestable faults λόγω αδυναμίας προσέγγισης συγκεκριμένων καταστάσεων ή αδυναμίας μεταβάσεων μεταξύ καταστάσεων

Τεχνική FIRE

- Στηρίζεται στην ανάλυση των συγκρούσεων μεταξύ των λογικών τιμών που μπορεί να πάρει μια γραμμή του κυκλώματος
- S0 = σύνολο σφαλμάτων που δεν μπορούν να δοκιμαστούν όταν το σήμα σε μια γραμμή είναι s=0
  - Αυτά τα σφάλματα απαιτούν το s=1 για να ανιχνευθούν
- S1 = σύνολο σφαλμάτων που δεν μπορούν να δοκιμαστούν όταν το σήμα σε μια γραμμή είναι s=1
  - Αυτά τα σφάλματα απαιτούν το s=0 για να ανιχνευθούν
- Η τομή των συνόλων S0 και S1 περιλαμβάνει τα untestable σφάλματα
  - Απαιτούν ταυτόχρονα s=1 και s=0 για να ανιχνευθούν τα σφάλματα
- Η αναζήτηση περιορίζεται μόνο στις διακλαδώσεις (fanout stem) του κυκλωματος

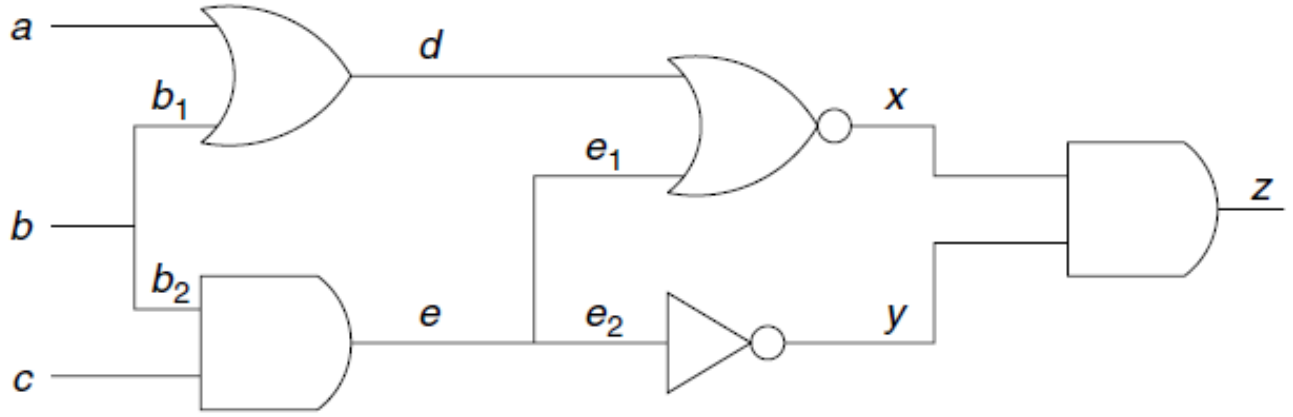
Παράδειγμα FIRE (1/3)

- Όταν το b=1 → b_1=1, b_2=1, d=1, x=0, z=0
  - Σφάλματα που δεν μπορούν να διεγερθούν: {b/1, b_1/1, b_2/1, d/1, x/0, z/0}
  - Σφάλματα που δεν μπορούν να παρατηρηθούν: {a/0, a/1, e1/0, e1/1, y/0, y/1, e2/0, e2/1, e/0, e/1, c/0, c/1, b2/0, b2/1}
  - Σφάλματα που δεν μπορούν να ανιχνευθούν: {b/1, b1/1, b2/1, d/1, x/0, z/0, a/0, a/1, e1/0, e1/1, y/0, y/1, e2/0, e2/1, e/0, e/1, c/0, c/1, b2/0, b2/1}



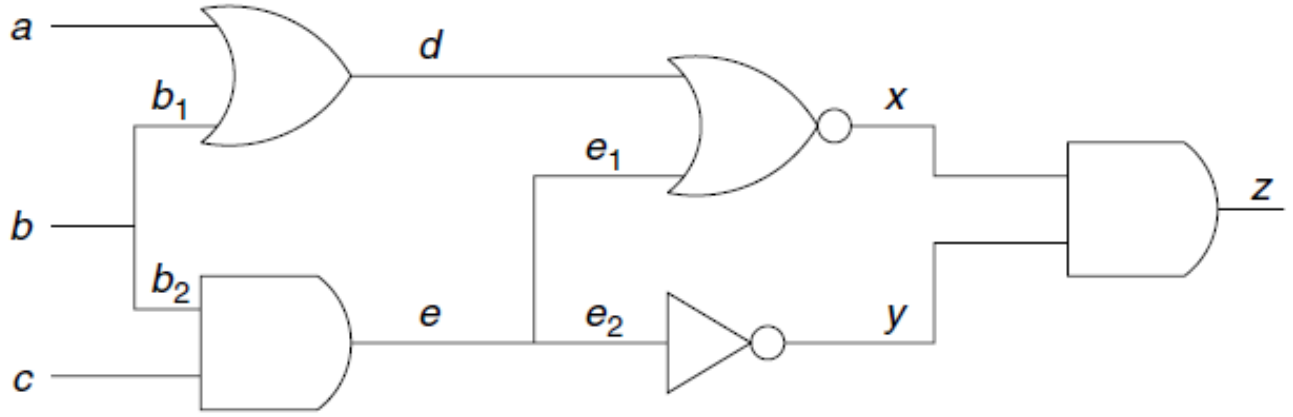
Παράδειγμα FIRE (2/3)

- Όταν το b=0 → b_1=0, b_2=0, e=0, e_1=0, e_2=0, y=1
  - Σφάλματα που δεν μπορούν να διεγερθούν: {b/0, b_1/0, b_2/0, e/0, e_1/0, e_2/0, y/1}
  - Σφάλματα που δεν μπορούν να παρατηρηθούν: {c/0, c/1}
  - Σφάλματα που δεν μπορούν να ανιχνευθούν: {b/0, b_1/0, b_2/0, c/0, c/1,e/0, e_1/0, e_2/0, y/1}



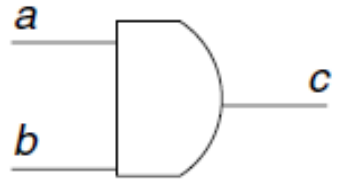
Παράδειγμα FIRE (3/3)

- Η τομή ανάμεσα στα δύο σύνολα περιλαμβάνει τα σφάλματα τα οποία για να ανιχνευθούν απαιτούν b=1 και b=0
- Λίστα untestable σφαλμάτων: {b_2/0, c/0, c/1, e/0, e_1/0, e_2/0, y/1}



Συγκρούσεις σε πολλαπλές γραμμές

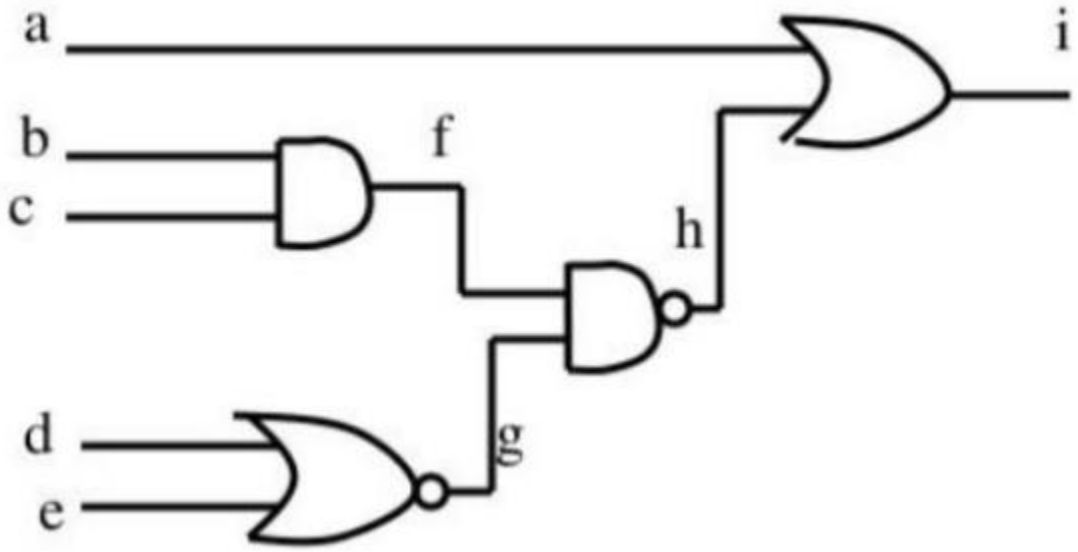
- Συγκρούσεις μπορεί να εμφανιστο ύν σε πολλαπλές γραμμές ενός κυκλώματος
- Το {a=0, c=1} δεν είναι δυνατό
- Το {b=0, c=1} δεν είναι δυνατό
- Το {a=1, b=1, c=0} αποτελεί μία σύγκρουση πολλαπλών γραμμών η οποία δεν μπορεί να διαχειριστεί με το μοντέλο συγκρούσεων μιας γραμμής (που χρησιμοποιεί το FIRE)



- 3 σύνολα
  - S_0: Σύνολο σφαλμάτων που δεν ανιχνεύονται όταν a = 0.
  - S_1: Σύνολο σφαλμάτων που δεν ανιχνεύονται όταν b = 0.
  - S_2: Σύνολο σφαλμάτων που δεν ανιχνεύονται όταν c = 1.
- Η τομή ανάμεσα στα 3 σύνολα δίνει τα σφάλματα που δεν ανιχνεύονται όταν εμφανιστεί η παραπάνω σύγκρουση πολλαπλών γραμμών

Άσκηση

- Κάνοντας χρήση της διαφοράς Boole, βρείτε ένα διάνυσμα δοκιμής το οποίο ανιχνεύει το σφάλμα “a stuck at 0” και άλλο ένα διάνυσμα δοκιμής το οποίο ανιχνεύει το σφάλμα “a stuck at 1” στο παρακάτω κύκλωμα



Λύση

- Για να διεγερθεί το σφάλμα a stuck at 0 θέτουμε α=1, οπότε έχουμε:

$$a \cdot \frac{di}{da} = 1 \Rightarrow a \cdot [i(a=1) \oplus i(a=0)] = 1 \Rightarrow a \cdot \left[1 \oplus \overline{bc(\overline{d+e})}\right] = 1 \Rightarrow$$
$$= a \cdot \left[bc\left(\overline{d+e}\right)\right] = 1 \Rightarrow abc(\overline{d+e}) = 1$$

* **Ισχύει η σχέση $1 \oplus X = \overline{X}$**

- Το σφάλμα a stuck at 0 ανιχνεύεται από το {a,b,c,d,e}={1,1,1,0,0}
- Για να διεγερθεί το σφάλμα a stuck at 1 θέτουμε $\bar{a} = 1$, οπότε έχουμε:

$$\bar{a} \cdot \frac{di}{da} = 1 \Rightarrow \bar{a} \cdot [i(a=1) \oplus i(a=0)] = 1 \Rightarrow \bar{a} \cdot \left[1 \oplus \overline{bc(\overline{d+e})}\right] = 1 \Rightarrow$$
$$\bar{a}bc(\overline{d+e}) = 1$$

- Το σφάλμα a stuck at 1 ανιχνεύεται από το {a,b,c,d,e}={0,1,1,0,0}

Πηγές

- Fay Talan, “VLSI Test Principles and Architectures”.

Ερωτήσεις