EEDG/CE 6303: Testing and Testable Design

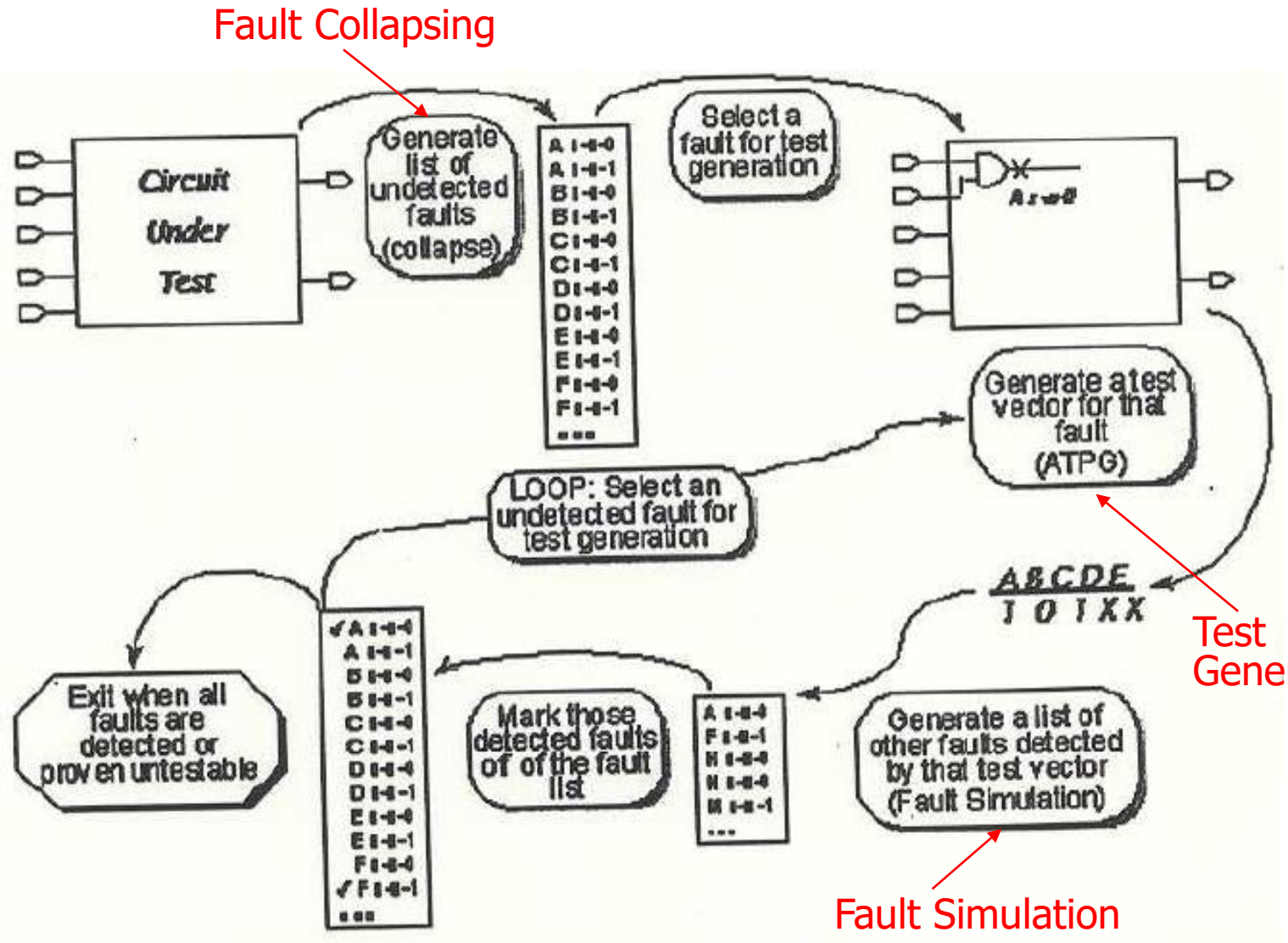
Mehrdad Nourani

Dept. of ECE
Univ. of Texas at Dallas

Session 04

Fault Simulation

Fault Analysis System (Review)



Logic Simulation

Logic Simulation

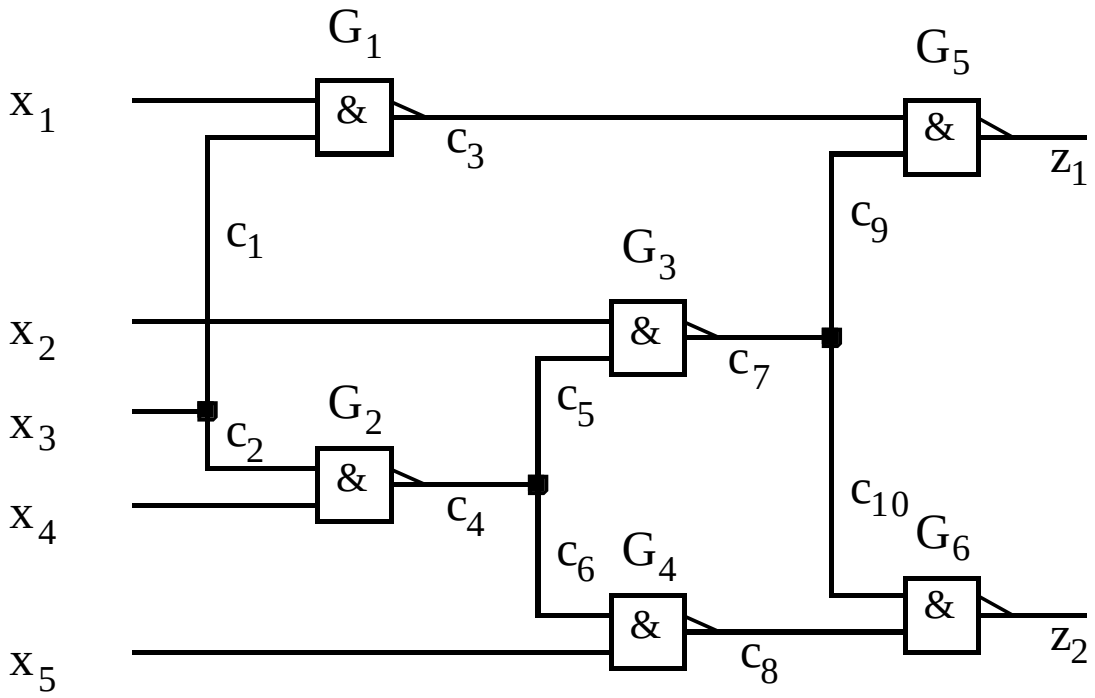
- **Logic simulation** for a combinational logic circuit is the determination of steady-state logic values implied at each circuit line by the vector applied to its primary inputs
  - **Two-valued** logic simulation: When each component of the vector applied $P = (p_1, p_2, \ldots p_n)$ is fully-specified, i.e., can take only two values – 0 and 1
  - **Three-valued** logic simulation: When one or more component of vector applied is incompletely specified, i.e., has the value x
  - There are 5, 7, 9, even 41-level logic simulation where more detailed information is processed and provided.

Logic Simulation (cont'd)

- Such simulators implicitly assume that the details of delays are unimportant in determining the steady state values implied by this vector
  - Are hence called **cycle-based** or **zero delay** simulators
  - If used to verify correctness of a circuit design, such simulation must be complemented by **timing analysis**
- A simple logic simulator must
  - Read vector to be simulated and assign the values to corresponding inputs
  - Compute the value implied by the vector at each circuit line

Logic Simulation (cont'd)

- However, the value at a line must be computed **after** the values at all its fanins are computed
  - E.g., value at c_7 must be computed after those at both its fan-ins, x_2 and c_5, are computed



Level-Based Logic Simulation

- For combinational circuits, lines can be ordered a priori to ensure correct logic simulation
  - **Input level** of line c_i in a combinational circuit, $\eta_{inp}(c_i)$, is the maximum number of circuit elements traversed along any path from any of the circuit inputs to the line
  - Circuit lines can then be considered during simulation in non-decreasing order in terms of the values of their input levels

Level-Based Logic Simulation (cont'd)

- Procedure **InputLevelize()** can compute $\eta_{inp}(c_i)$ for all lines

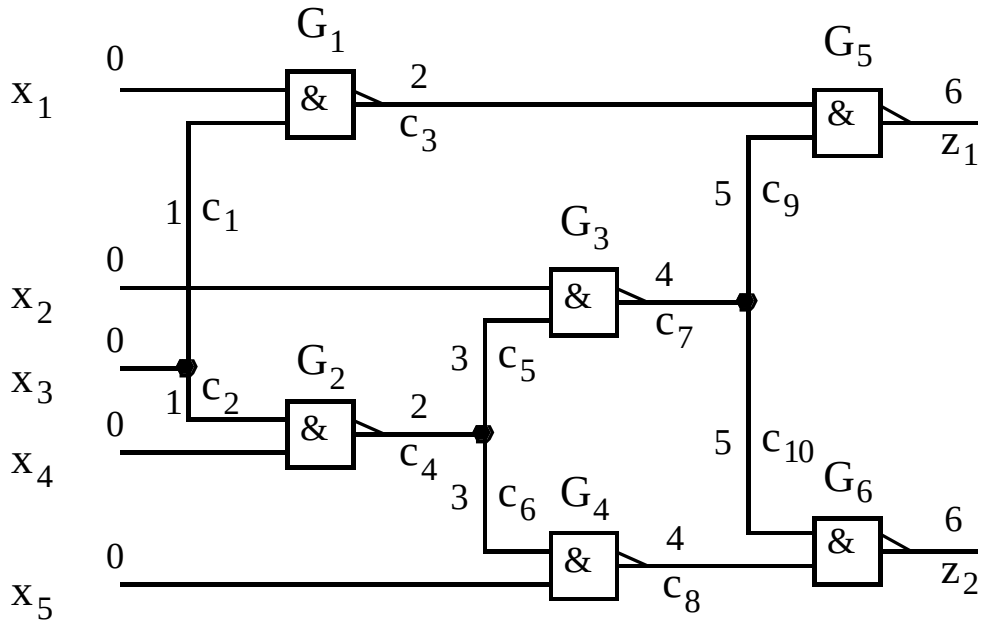
Procedure 3.2: InputLevelize()

1. *Initialization* : For each circuit line c, η_{inp} = undefined.
2. For each primary input x_i, $\eta_{inp} = 0$.
3. While there exists one or more logic elements such that (i) η_{inp} is defined for each of the element's inputs, and (ii) η_{inp} is undefined for any of its outputs, select one such element.
   (a) If the selected element is a gate with inputs c_{i_1}, c_{i_1},..., c_{i_α} and output c_j, then assign $\eta_{inp}(c_j) = \max[\eta_{inp}(c_1), \eta_{inp}(c_2), \ldots, \eta_{inp}(c_\alpha)] + 1$.
   (b) If the selected element is a fanout system with stem c_i and branches c_{j_1}, c_{j_2},..., c_{j_β}, then for each output c_{j_l}, where $l = 1, 2, \ldots, \beta$, assign $\eta_{inp}(c_{j_l}) = \eta_{inp}(c_i) + 1$.

- Note that Step 3 of above procedure can be implemented more efficiently

Level-Based Logic Simulation (cont'd)

- Once η_{inp} is computed for each line, the lines can be sorted in ascending order of their η_{inp} values in an **ordered list** $Q_{\eta_{inp}}$.
- Example of computing η_{inp} :

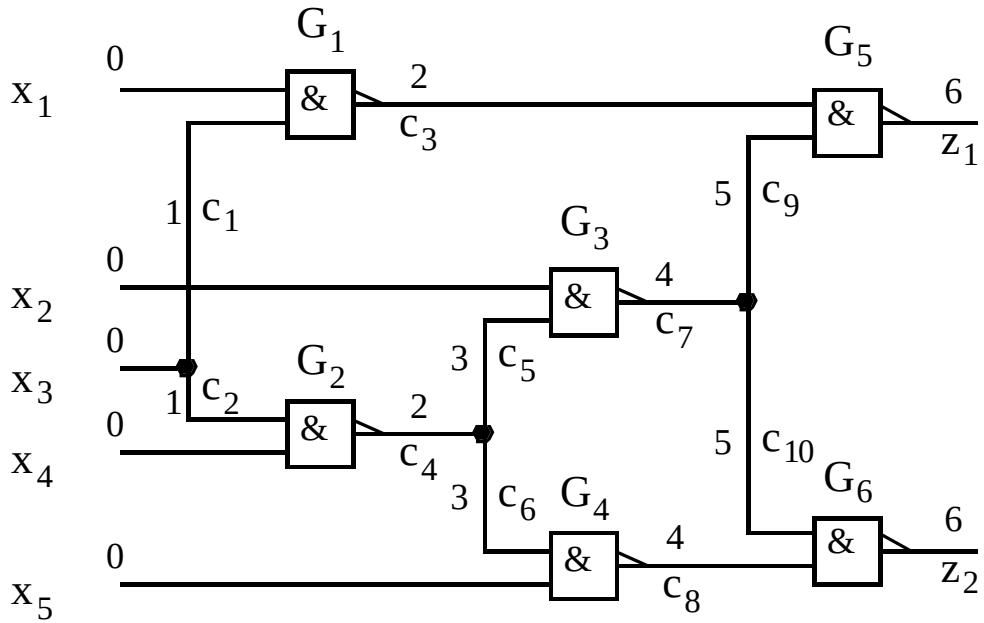


Level-Based Logic Simulation (cont'd)

- Once the η_{inp} values are computed, the lines can be ordered as:

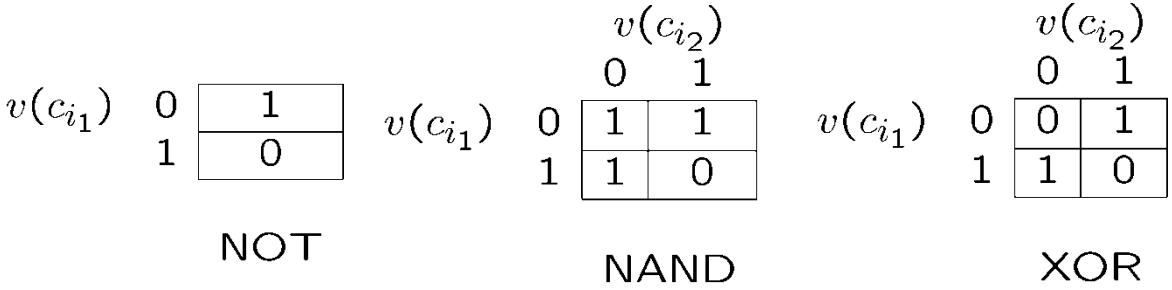
$Q_{\eta_{inp}}=(x_1, x_2, x_3, x_4, x_5, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, z_1, z_2)$

  - Note that $Q_{\eta_{inp}}$ is not unique in this case
  - Circuit lines can be considered in the sequence given by $Q_{\eta_{inp}}$ during logic simulation



Level-Based Logic Simulation (cont'd)

- 2-level logic simulation of primitive gates:



$v(c_{i_1})$	0	1
	1	0

NOT

$v(c_{i_1})$ \ $v(c_{i_2})$	0	1
0	1	1
1	1	0

NAND

$v(c_{i_1})$ \ $v(c_{i_2})$	0	1
0	0	1
1	1	0

XOR

$v(c_i)$	$v(c_{j_1})$	$\cdots$	$v(c_{j_l})$	$\cdots$
0	0	$\cdots$	0	$\cdots$
1	1	$\cdots$	1	$\cdots$

Gate driving line c_j	Logic expression $\mathcal{G}_j^2()$
NOT	$v(c_j) =$ not $v(c_{i_1})$
AND	$v(c_j) = v(c_{i_1})$ and $v(c_{i_2})$
NAND	$v(c_j) =$ not $[v(c_{i_1})$ and $v(c_{i_2})]$
OR	$v(c_j) = v(c_{i_1})$ or $v(c_{i_2})$
NOR	$v(c_j) =$ not $[v(c_{i_1})$ or $v(c_{i_2})]$
XOR	$v(c_j) = \{v(c_{i_1})$ and [not $v(c_{i_2})]\}$ or $\{[$ not $v(c_{i_1})]$ and $v(c_{i_2})\}$

Level-Based Logic Simulation (cont'd)

- 3-level logic simulation of primitive gates:

NOT

$v(c_{i_1})$	
0	1
1	0
x	x

NAND

$v(c_{i_1})$ \ $v(c_{i_2})$	0	1	x
0	1	1	1
1	1	0	x
x	1	x	x

XOR

$v(c_{i_1})$ \ $v(c_{i_2})$	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

Symbol	v	u
0	0	0
1	1	1
x	1	0
–	0	1

Gate driving line c_j	Logic expressions $\mathcal{G}_j^3()$
NOT	$v(c_j) =$ not $u(c_{i_1})$ $u(c_j) =$ not $v(c_{i_1})$
AND	$v(c_j) = v(c_{i_1})$ and $v(c_{i_2})$ $u(c_j) = u(c_{i_1})$ and $u(c_{i_2})$
NAND	$v(c_j) =$ not $[u(c_{i_1})$ and $u(c_{i_2})]$ $u(c_j) =$ not $[v(c_{i_1})$ and $v(c_{i_2})]$
OR	$v(c_j) = v(c_{i_1})$ or $v(c_{i_2})$ $u(c_j) = u(c_{i_1})$ or $u(c_{i_2})$
NOR	$v(c_j) =$ not $[u(c_{i_1})$ or $u(c_{i_2})]$ $u(c_j) =$ not $[v(c_{i_1})$ or $v(c_{i_2})]$
XOR	$v(c_j) = \{v(c_{i_1})$ and [not $u(c_{i_2})]\}$ or $\{[$ not $u(c_{i_1})]$ and $v(c_{i_2})\}$ $u(c_j) = \{u(c_{i_1})$ and [not $v(c_{i_2})]\}$ or $\{[$ not $v(c_{i_1})]$ and $u(c_{i_2})\}$

Level-Based Logic Simulation (cont'd)

- Simple logic simulation:

Procedure 3.3: SimpleLogicSimulation()

1. *Preprocessing* : Use Procedure InputLevelize() to compute the input level $\eta_{inp}(c_i)$ of each line and to obtain $\mathcal{Q}_{\eta_{inp}}$, the ordered list of circuit lines.

2. While there exists a vector to be simulated, read a vector $P = (p_1, p_2, \ldots, p_n)$.

   (a) For each primary input x_i, assign $v(x_i) = p_i$.

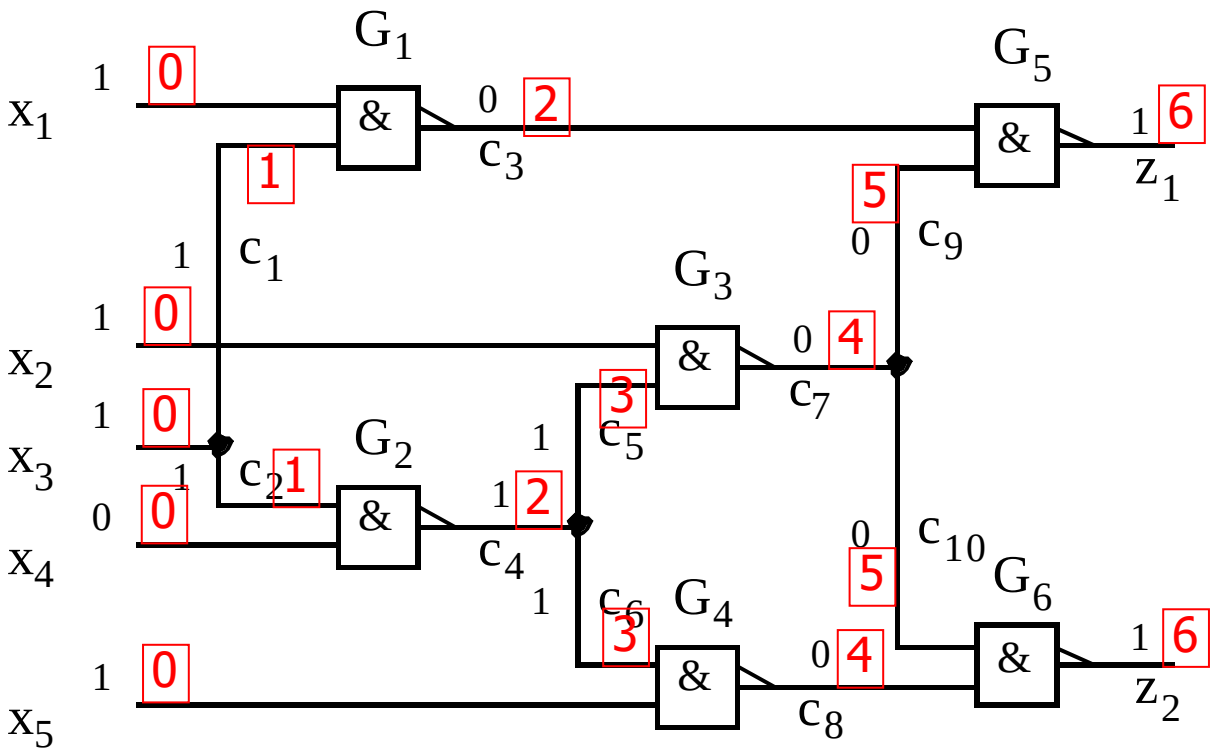
   (b) In the order in which lines appear in $\mathcal{Q}_{\eta_{inp}}$, for each line c_i in $\mathcal{Q}_{\eta_{inp}}$, compute the value $v(c_i)$.

      i. If c_j is the output of a gate, then use following equation.
      $v(c_j) = \mathcal{G}^2{}_j(v(c_{i_1}), v(c_{i_1}), \ldots, v(c_{i_\alpha}))$

      ii. If c_j is a branch of a fanout system, then use the following equation.
      $v(c_j) = \mathcal{G}^2{}_j(v(c_i))$

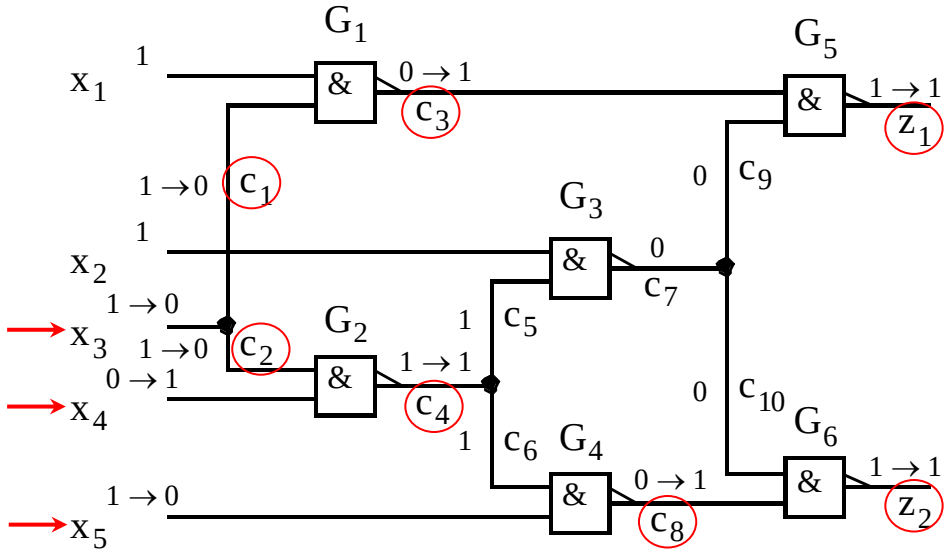
Level-Based Logic Simulation (cont'd)

- Example of simulation:
  - Vector P = (1, 1, 1, 0, 1)
  - $Q_{\eta_{inp}}$=(x_1, x_2, x_3, x_4, x_5, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}, z_1, z_2)



Event-Driven Logic Simulation

- **Event-driven** logic simulation takes advantage of the fact that subsequent vectors simulated may imply the identical values at many circuit lines.
- P = (x_1, x_2, x_3, x_4, x_5) =(1, 1, 1, 0, 1) → P = (1, 1, 0, 1, 0)



- When the second vector is simulated, values are recomputed at only 7 (out of 12) lines

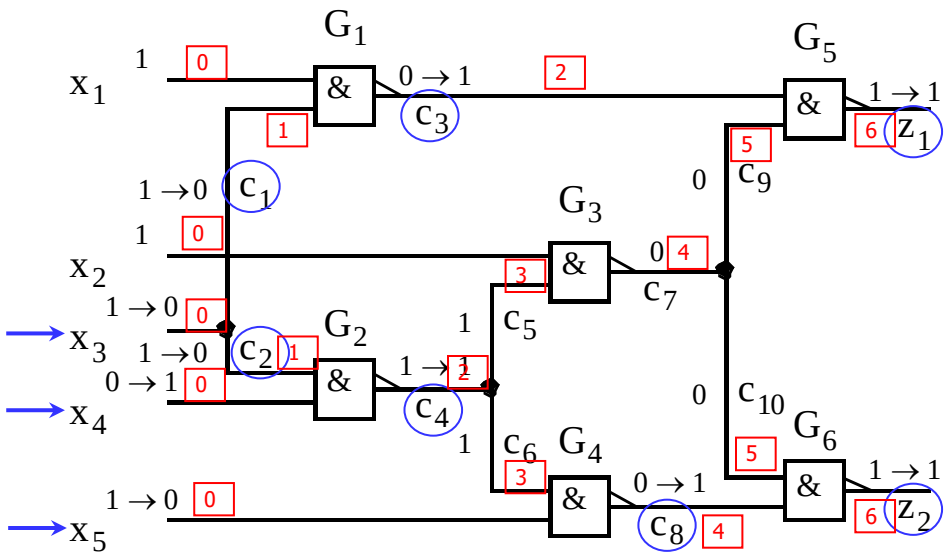
Event-Driven Logic Simulation (cont'd)

- An **event** is said to occur at a circuit line during the simulation of the current vector if the value implied at the line by the current vector is different from that implied by the previous vector simulated.
- Value at line c_j is recomputed only if an event occurs at one or more lines in its fanin
  - —Sometimes the value recomputed for the current vector may turn out to be identical to that implied by the previous vector

Event-Driven Logic Simulation (cont'd)

- Identification, recording, and use of events
  - —The value implied at a line by the current vector compared with that for the previous vector
    - If the two are identical, no event
    - Otherwise, an event is identified
    - If necessary, the value at the line is updated
  - —If an event is identified at c_i, then each line c_j in fanout of c_i is added to a list $Q_{n_{inp}}(c_j)$ *if it does not already belong to the list*
    - At each step of the algorithm, the first non-empty list Q_i is identified
    - A line c_j is removed from Q_i
    - Values implied at c_i by current vector is computed, presence or absence of event at c_j identified, value at c_i is updated, and any event at c_i is handled as above
    - Simulation for current vector stops when all Q_i are empty

Event-Driven Logic Simulation (cont'd)



- If an event is identified at c_i, then each line c_j in fanout of c_i is added to a list $Q_{n_{inp}}(c_j)$ *if it does not already belong to the list*
  - At each step of the algorithm, the first non-empty list Q_i is identified
  - A line c_j is removed from Q_i
  - Values implied at c_i by current vector is computed, presence or absence of event at c_j identified, value at c_i is updated, and any event at c_i is handled as above
  - Simulation for current vector stops when all Q_i are empty

List	Lines
Q0	**x3**, x4, x5
Q1	
Q2	
Q3	
Q4	
Q5	
Q6	

List	Lines
Q0	**x4**, x5
Q1	c1, c2
Q2	
Q3	
Q4	
Q5	
Q6	

List	Lines
Q0	**x5**
Q1	c1, c2
Q2	c4
Q3	
Q4	
Q5	
Q6	

List	Lines
Q0	
Q1	**c1**, c2
Q2	c4
Q3	
Q4	c8
Q5	
Q6	

List	Lines
Q0	
Q1	**c2**
Q2	c4, c3
Q3	
Q4	c8
Q5	
Q6	

List	Lines
Q0	
Q1	
Q2	**c4**, c3
Q3	
Q4	c8
Q5	
Q6	

List	Lines
Q0	
Q1	
Q2	**c3**
Q3	
Q4	c8
Q5	
Q6	

List	Lines
Q0	
Q1	
Q2	
Q3	
Q4	**c8**
Q5	
Q6	z1

List	Lines
Q0	
Q1	
Q2	
Q3	
Q4	
Q5	
Q6	**z1, z2**

List	Lines
Q0	
Q1	
Q2	
Q3	
Q4	
Q5	
Q6	

- The table is filled and values are assessed in breath-first (first-in first out) fashion.
  - **BLUE:** An event (a transition is assessed; one at a time)
  - **RED:** No event (transition stops; no need to traverse further)

Fault Simulation

- **Goal**: to determine the list of faults in a CUT (Circuit Under Test) and to simulate the fault-free and faulty circuits efficiently to determine if their outputs are different
- Five tasks involved:
  1. Fault-free (good) circuit simulation
  2. Fault specification (fault list generation and collapsing)
  3. Fault insertion (fault selection to be simulated and tracing the presence of the faults)
  4. Fault effect generation and propagation
  5. Fault detection and discarding