CIS 4930 Digital System Testing
Logic Simulation

Dr. Hao Zheng
Comp Sci & Eng
U of South Florida

Overview

- What is Logic Simulation?
- Types of Simulation
  - Compiled Simulation
  - Event Driven Simulation
- Delay models
- Hazards

3.1 What is Logic Simulation?

- Based on the logic model, it is the process of evolving signals in time in response to input stimuli

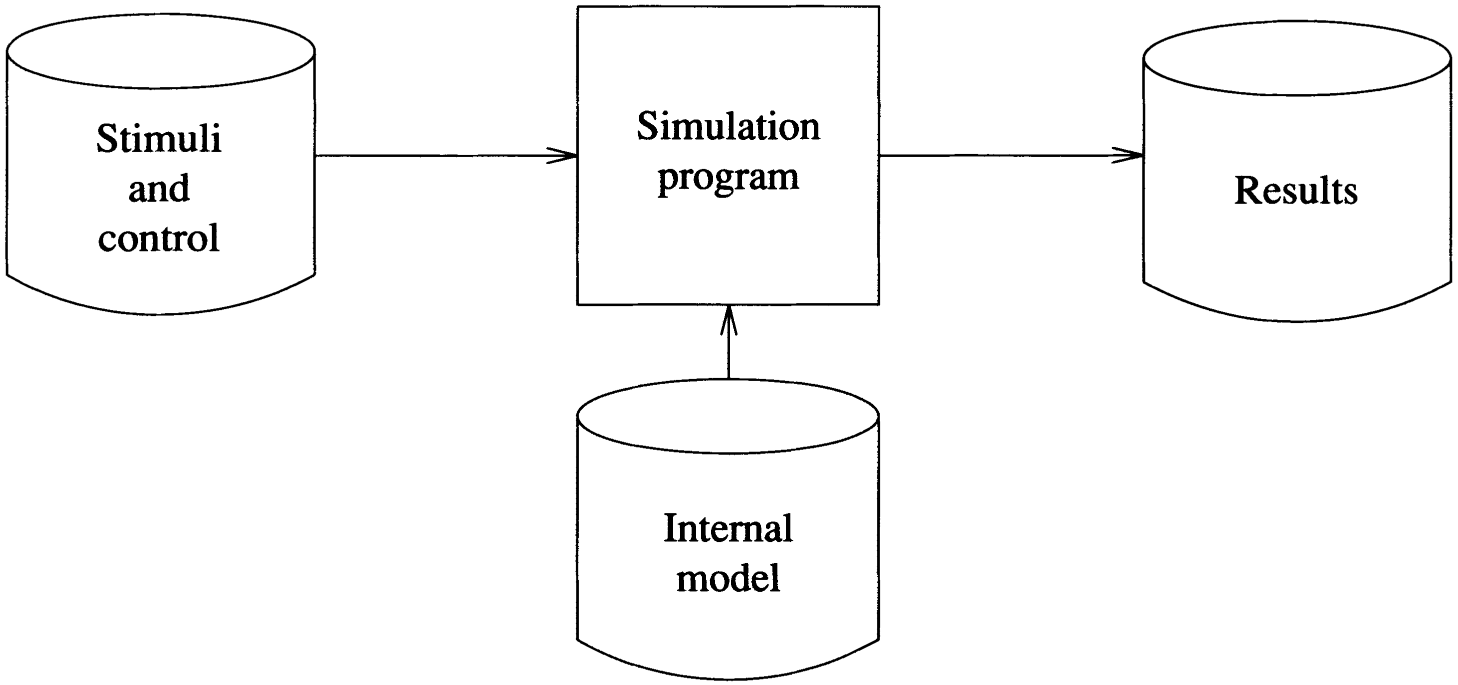


Figure 3.1 Simulation process

Why Logic Simulation?

- Verify design correctness i.e., design has desired behavior (function + timing)
- Verify against expected results
- Can be used to answer questions such as:
  - Correct independent of initial state?
  - Sensitive to delay variations of components?
  - Free of critical races, oscillations, etc.?

Why Logic Simulation?...contd.,

- We can also use it to:
  - Evaluate design alternatives
  - Evaluate proposed changes
  - Documentation (timing diagrams)
- Against a prototype, simulation has following advantages:
  - Checking various error conditions
  - Ability to change delays in the model
  - Start the simulated circuit in any desired state
  - Precise control of timing of async events (interrupts)
  - Automated testing

3.2 Problems in Logic Simulation

- How does one generate input stimuli?
  - Test generation
- How does one know the results are correct?
  - Output checking
- How "good" are the applied input stimuli, i.e., how "complete" is the testing experiment?
  - Test coverage

Your design is as good as your testing cases!

- Only the design behavior covered by the test cases is checked!

3.3 Types of Simulation

- **Compiled simulation**
  - Compiled code generated from RTL functional/structural models
  - Cannot deal with timing/delay
  - Applicable to synchronous functional testing.
- **Event-drive simulation**
  - Model interpreted from the data structures generated from RTL functional/structural models
  - Simulate only *active* part of the circuit.
  - Can simulate model with various delay specifications.
  - More general.

3.4 The Unknown Logic Value

- When a circuit is powered on, initial state of FFs and RAMs is unpredictable.
- Special logic value, ***u***, to indicate unknown logic value
  - need to extend Boolean operators to 3-valued logic
- ***u*** represents one value in set {0,1}
- 0 and 1 represented by sets {0}, {1} respectively

Boolean Operation with *u*

AND(0, **u**) = AND({0}, {0,1})
= {AND(0,0), AND(0,1)}
= {0, 0}
= {0} = 0

OR(0, **u**) = OR({0}, {0,1})
= {OR(0,0), OR(0,1)}
= {0, 1}
= **u**

NOT(**u**) = NOT({0,1}) = {NOT(0), NOT(1)} = {1, 0} = **u**

Truth Tables for 3-valued Logic

AND	0	1	u
0	0	0	0
1	0	1	u
u	0	u	u

OR	0	1	u
0	0	1	u
1	1	1	1
u	u	1	u

NOT	0	1	u
	1	0	u

Figure 3.2 Truth tables for 3-valued logic

$\cap$	0	1	x	u
0	0	$\varnothing$	0	$\varnothing$
1	$\varnothing$	1	1	$\varnothing$
x	0	1	x	u
u	$\varnothing$	$\varnothing$	u	u

Figure 3.3 Modified intersection operator

Evaluation of 3-valued Logic

- Evaluate $f(x_1, x_2, \ldots, x_n)$ for $(v_1, v_2, \ldots, v_n)$
  - Evaluate the vector with a primitive cube of f with the intersection operation
  - If the intersection is consistent, the output value of the primitive cube is the output of the vector on f. Otherwise, the vector leads to **u.**

Example: For a AND,

- What primitive cube is compatible with $(u,\ 0)$?
- What primitive cube is compatible with $(u,\ 1)$?

3-valued Logic – Pessimistic

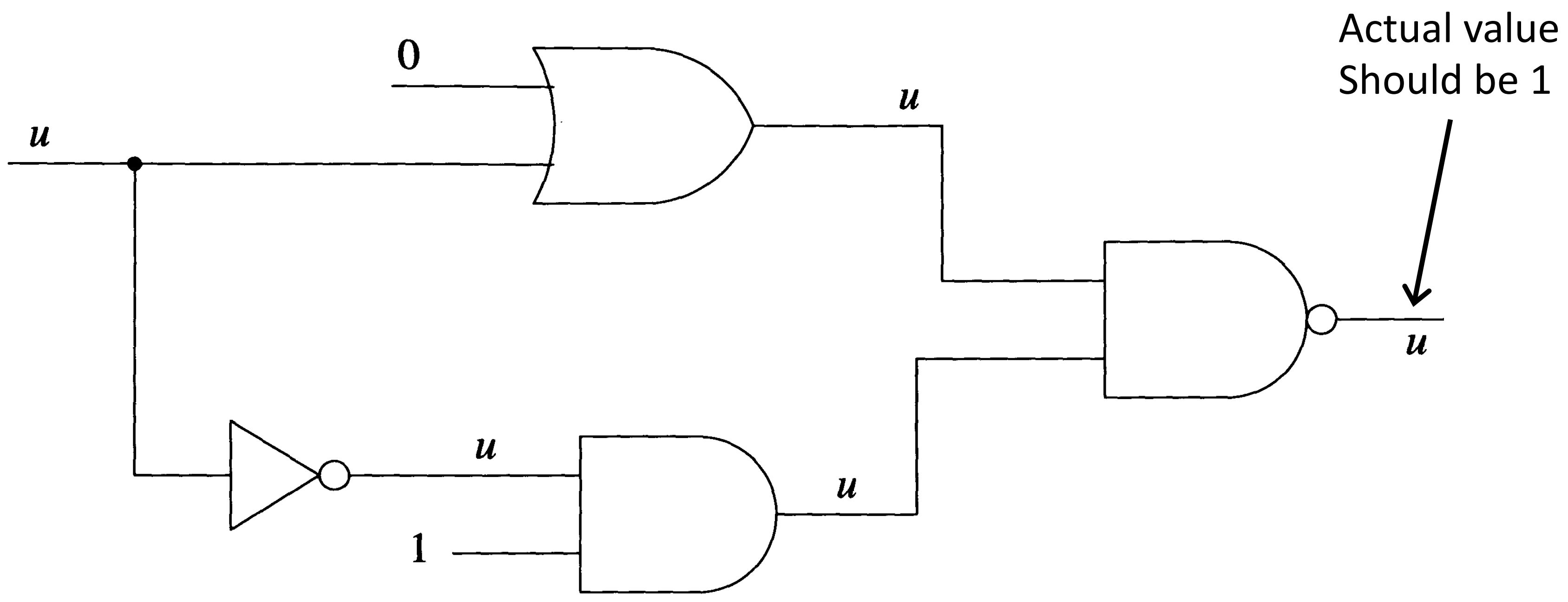


Figure 3.4 Pessimistic result in 3-valued simulation

Pessimistic result due to reconvergent fanout!!

3.5 Compiled Simulation

- Compiled model becomes part of the simulator
- In extreme cases, compiler model is the simulator!!
- Example: Assume data *F* to FF satisfies setup time constraint. Therefore, we can ignore gate delays

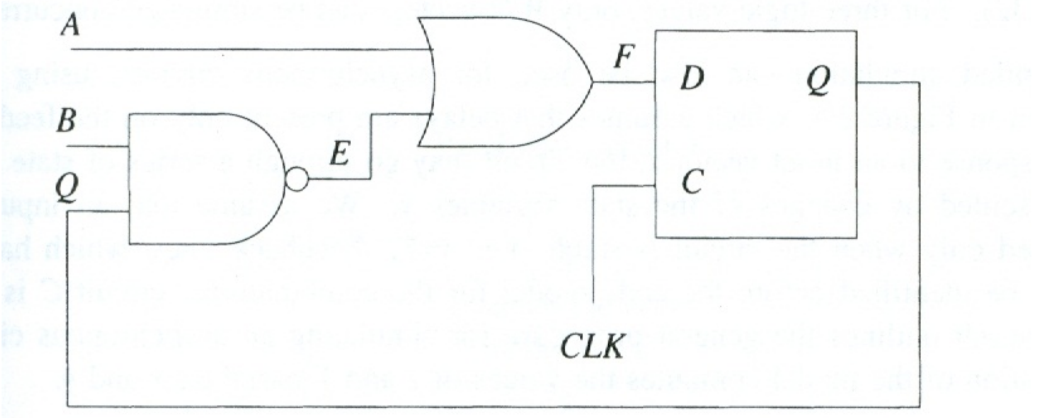


Figure 3.7 Synchronous circuit

Compiled Simulation - Example

- Simulate level by level

Note: for every input vector every element is evaluated

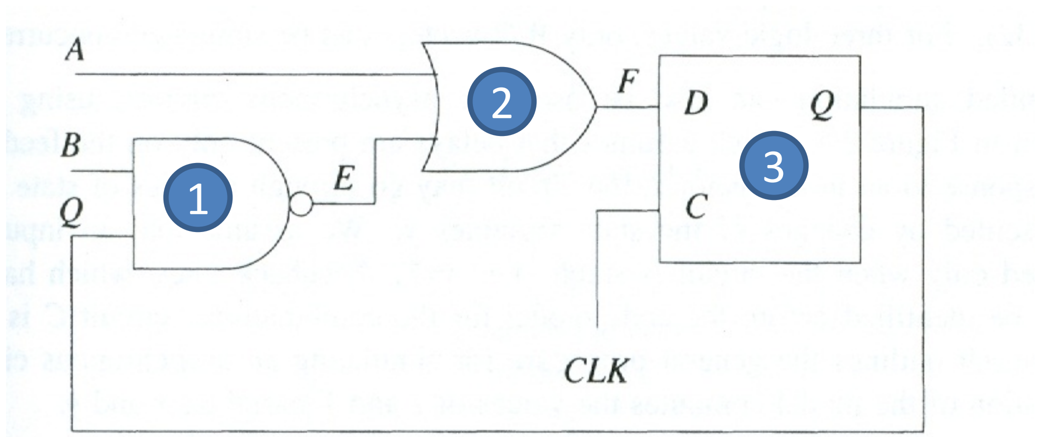


Figure 3.7 Synchronous circuit

Assembly Code

LDA	B
AND	Q
INV	
STA	E
OR	A
STA	F
STA	Q

Compiled Simulation: Asynchronous Circuits

- For asynchronous circuits, delay modeling in the feedback path is necessary

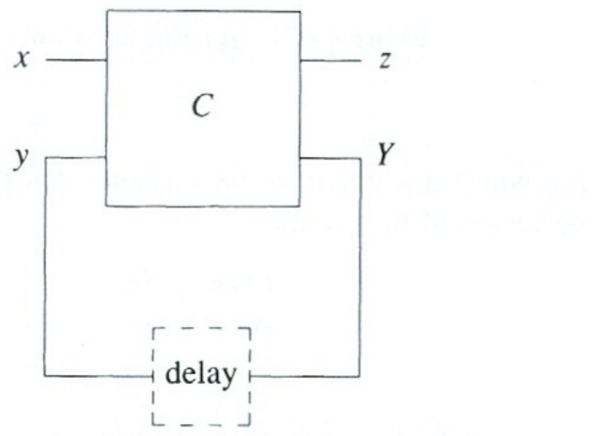


Figure 3.8 Asynchronous circuit model

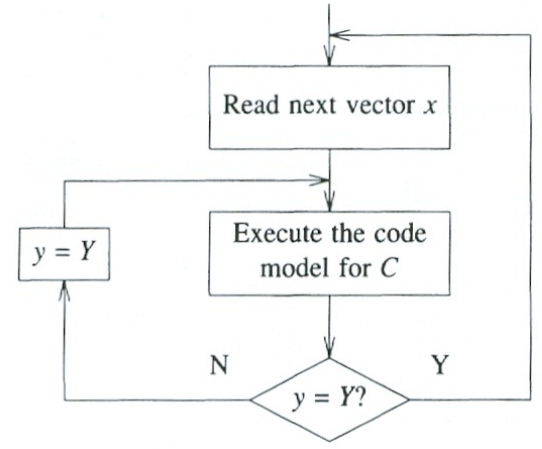
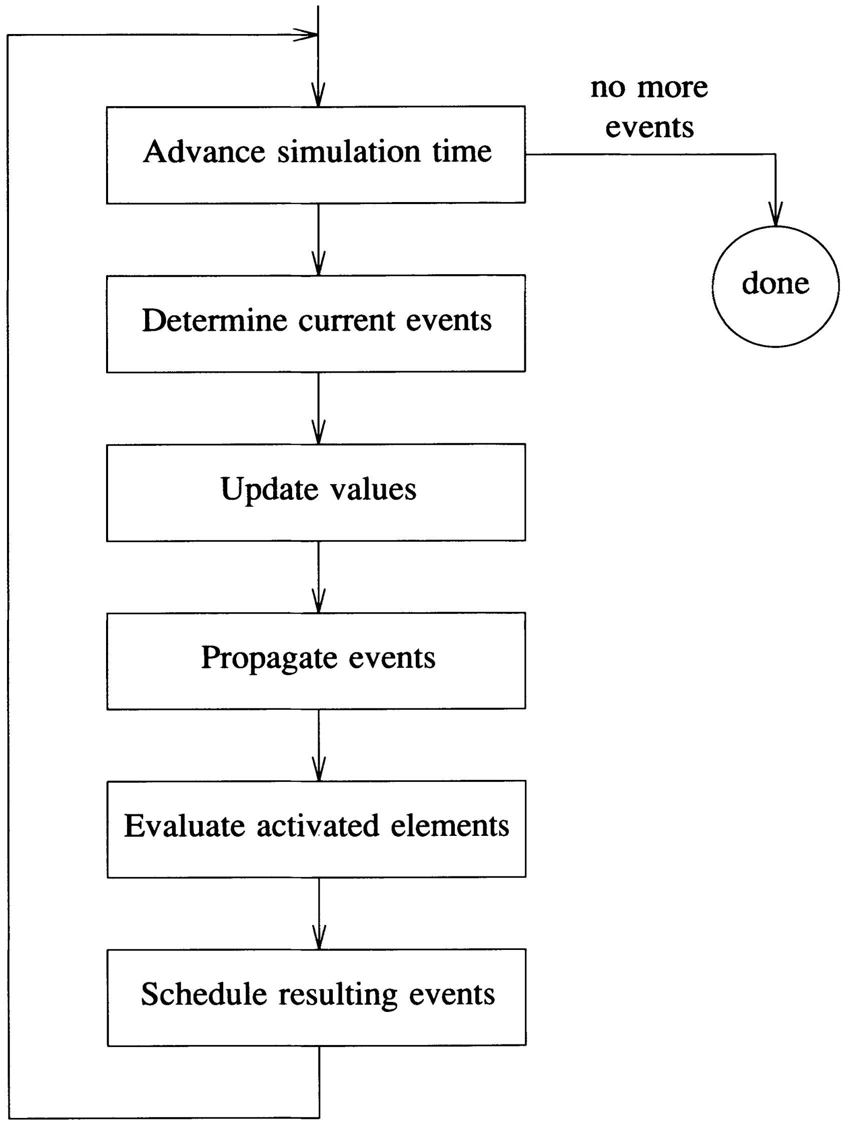


Figure 3.9 Asynchronous circuit simulation with compiled-code model

3.6 Event-driven simulation

- **Event** – Change in the value of a signal
- **Active Signal** – Signal with changing value at some arbitrary time
- **Activated element** – an element with at least one input signal with an event
- **Evaluation** – process of determining the output values of an element



Event-driven simulation: Terminology

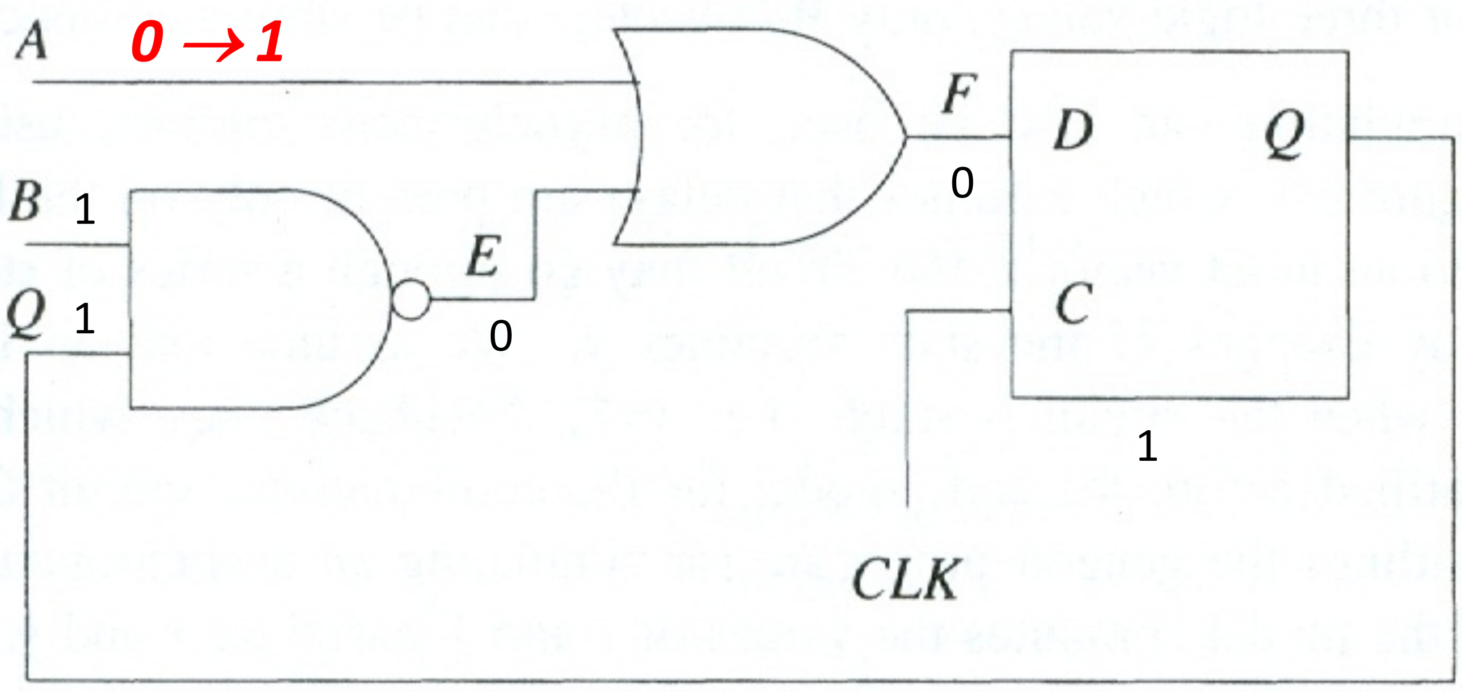
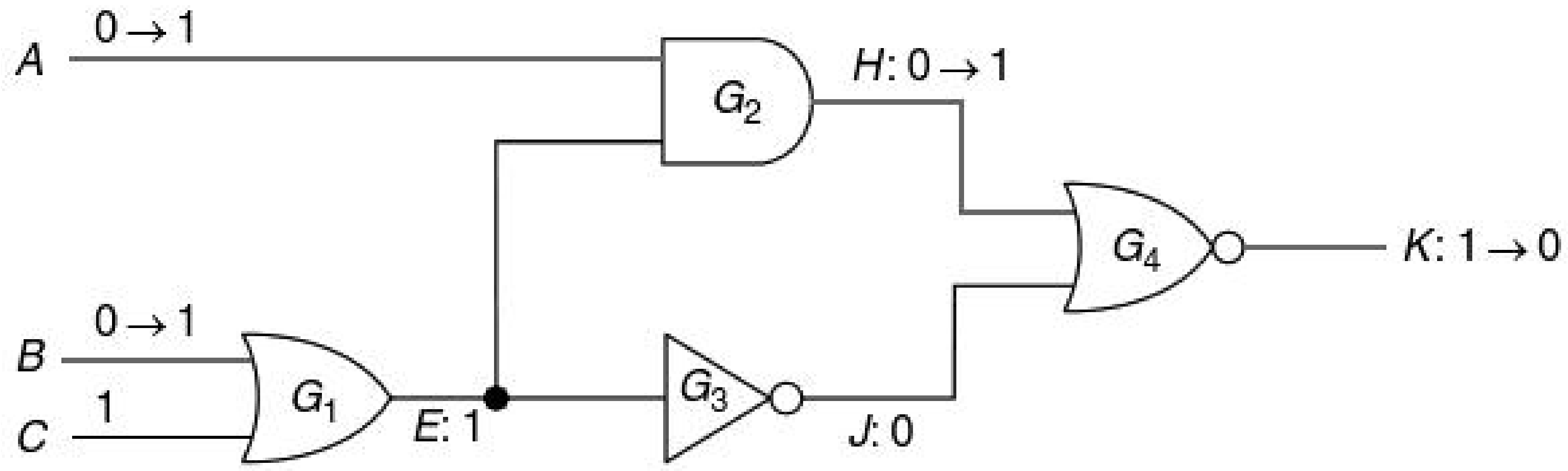


Figure 3.7 Synchronous circuit

Event-Driven Simulation: Example



Event-Driven Simulation

- Uses structural model to propagate events
- Stimulus file – changes in the values on primary inputs
- Events on other lines are produced by evaluations of activated elements
- Event occurs at a certain (simulated) time
- Applied stimuli – event sequences in time
- Can have events scheduled in future i.e., *pending* events
- Simulator maintains an *event list*

3.7. Delay Modeling

- Every gate introduces delay on signals propagating through it
- Behavior of a gate can be separated into functional and timing
- Delay Models
  - ❑ *Zero delay* – All gates have zero delay; used for functional verification
  - ❑ *Unit Delay* – All gates have delay = 1
  - ❑ *Real Delay* – All gates have different delays

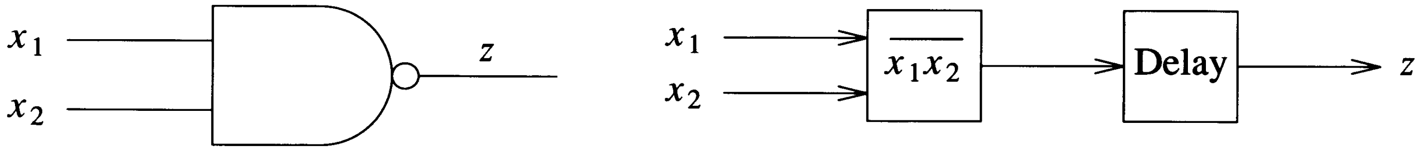
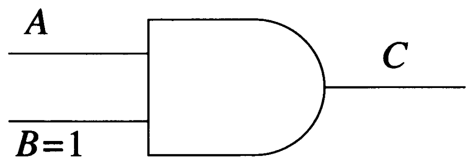


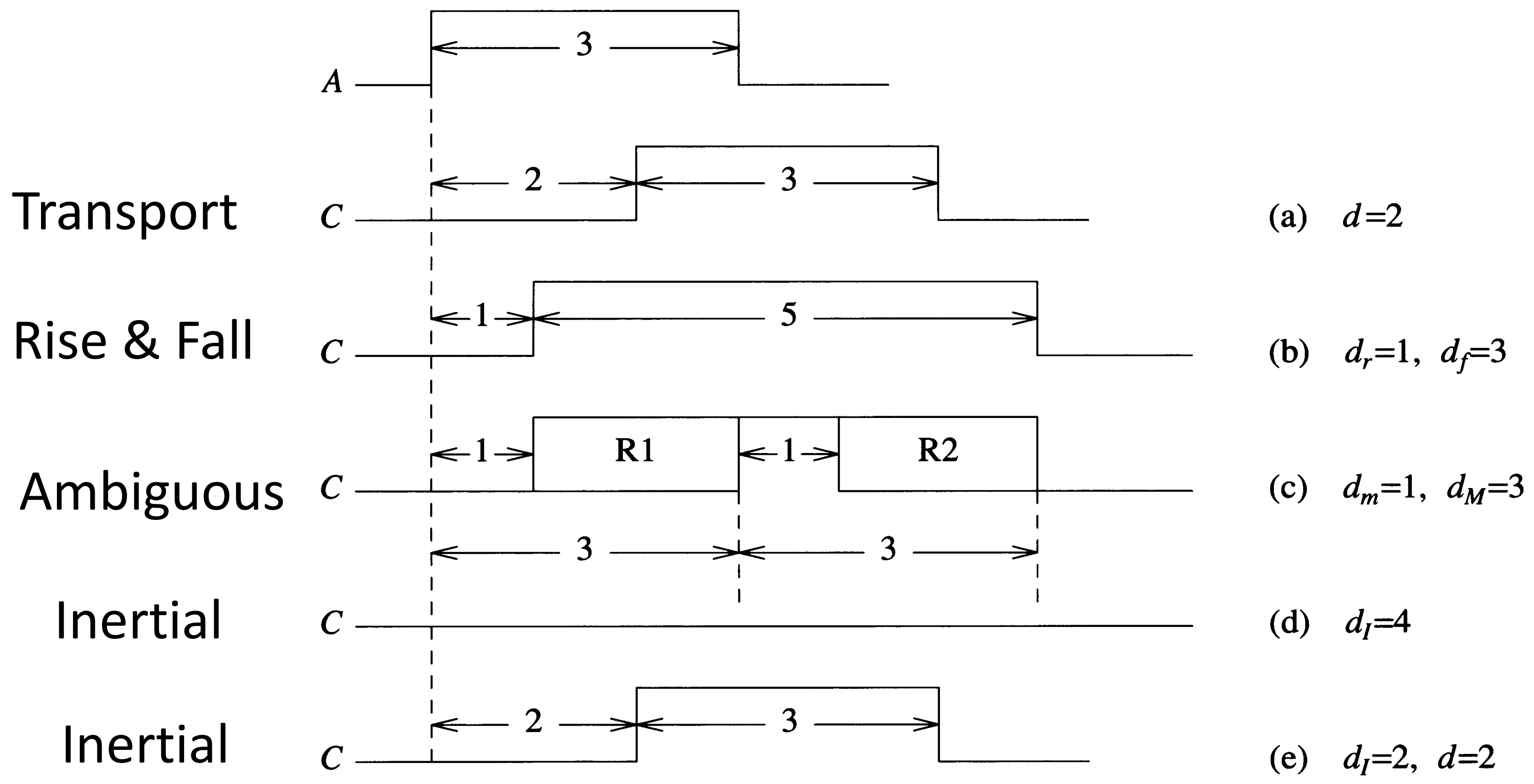
Figure 3.13 Separation between function and delay in modeling a gate

Delay Models

- **Transport Delay**
  - Interval separating output change from the associated input change
- **Inertial Delay**
  - All circuits require energy to switch states
  - Energy in a signal is a function of its amplitude and duration.
  - Therefore, an input signal of minimum duration is required to cause a change on the output.
  - Minimum duration is known as input inertial delay d_I

Example: Transport & Inertial Delay



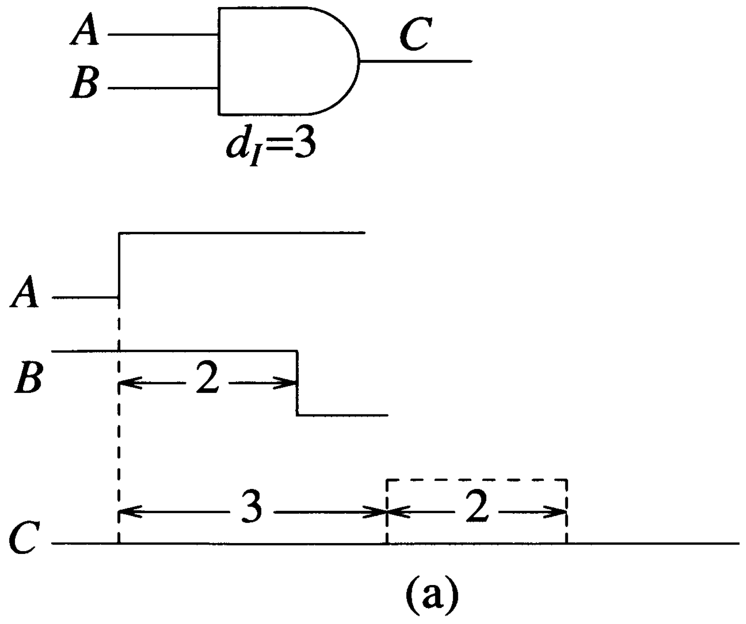


Inertial Delay

- Inertial delay can be modeled at the input or output of a gate
- Input inertial delay
  - No input < d_I can propagate through circuit
- Output inertial delay
  - No output < d_I can be generated

Output Inertial Delay

An output pulse caused by close input transitions



Example for which input and output inertial delay models will give different results

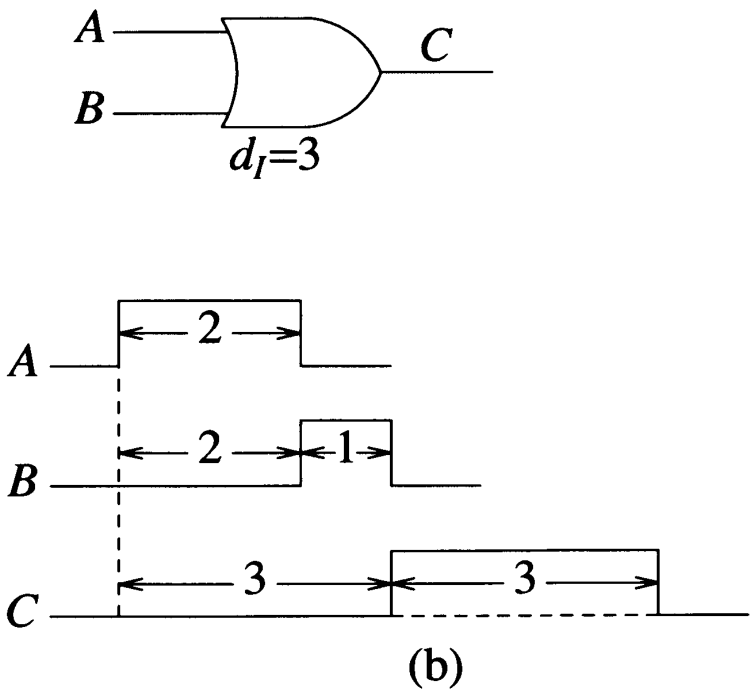
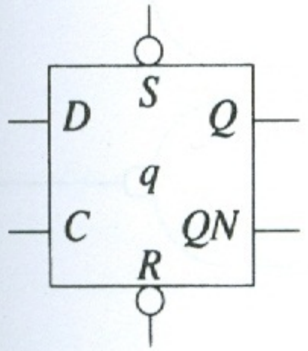
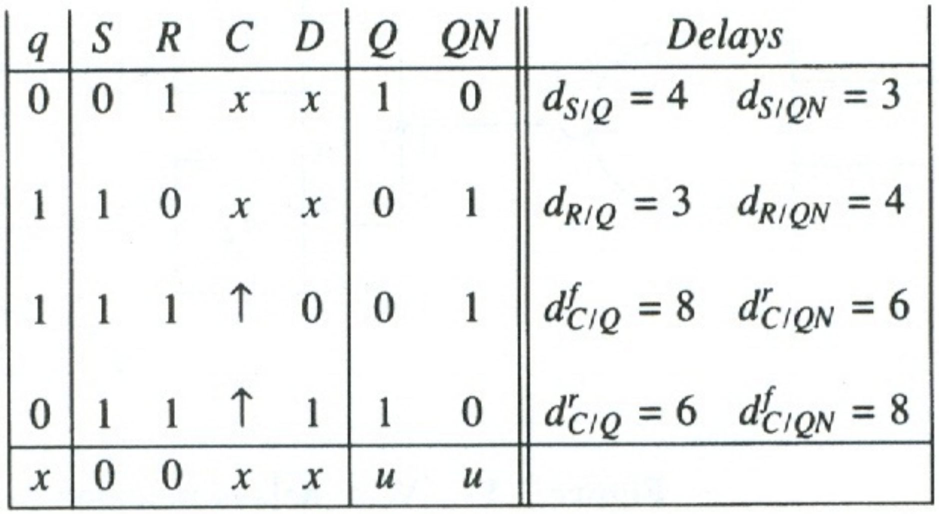


Figure 3.15 Output inertial delay

Delay Modeling - FF

- Delays in FF more complex
- Rise, Fall, input-to-output (I/O) delays



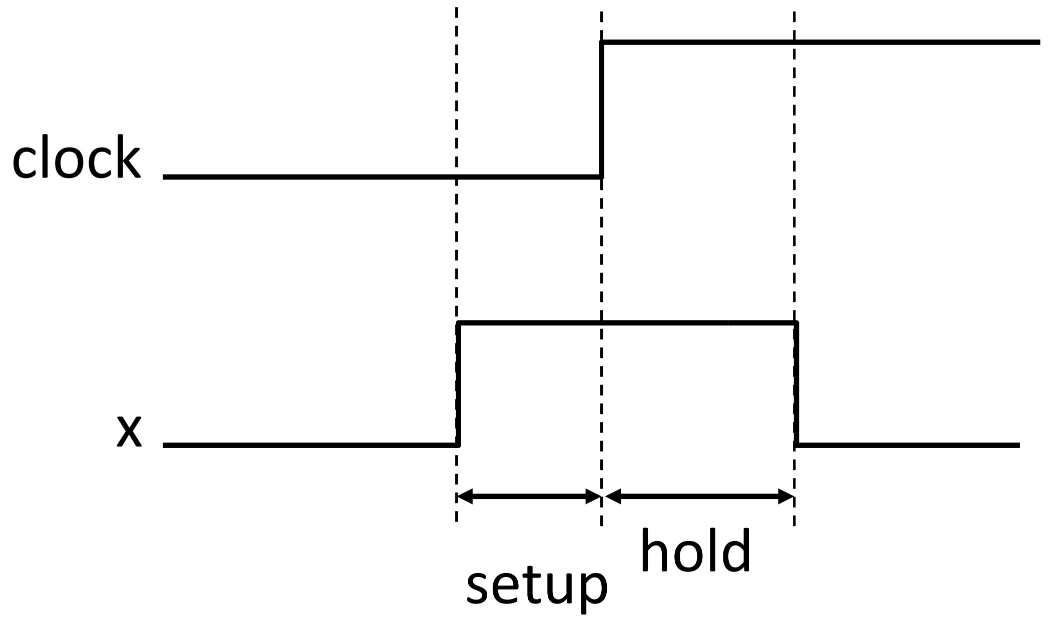


q	S	R	C	D	Q	QN	Delays	
0	0	1	x	x	1	0	$d_{S/Q} = 4$	$d_{S/QN} = 3$
1	1	0	x	x	0	1	$d_{R/Q} = 3$	$d_{R/QN} = 4$
1	1	1	$\uparrow$	0	0	1	$d^f_{C/Q} = 8$	$d^r_{C/QN} = 6$
0	1	1	$\uparrow$	1	1	0	$d^r_{C/Q} = 6$	$d^f_{C/QN} = 8$
x	0	0	x	x	u	u		

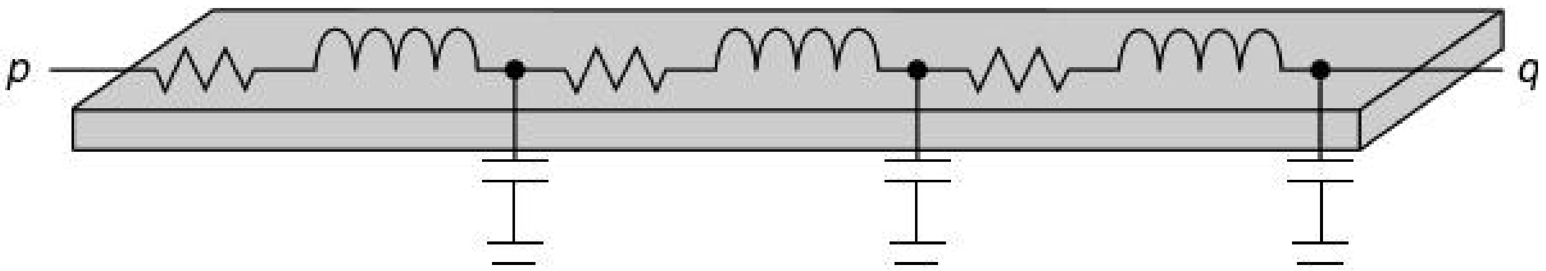
Figure 3.16 I/O delays for a *D* F/F

Race Condition

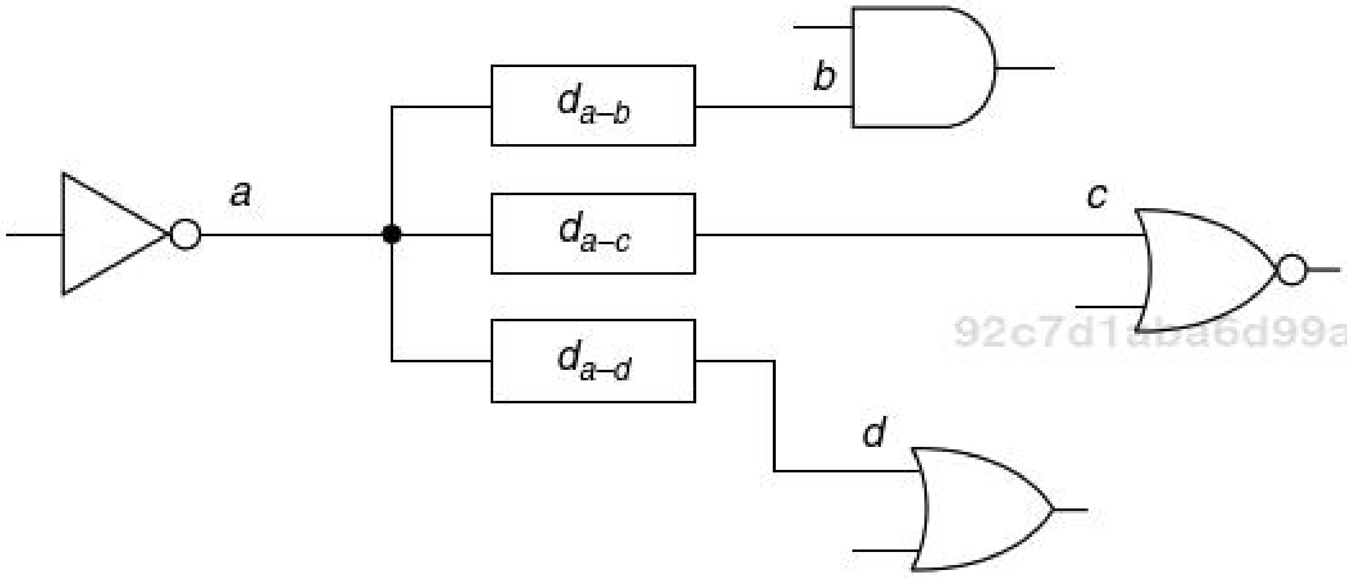
- Setup time
  - Minimum interval preceding an active clock edge.
- Hold time
  - Minimum interval following an active clock edge.



Wire Delay



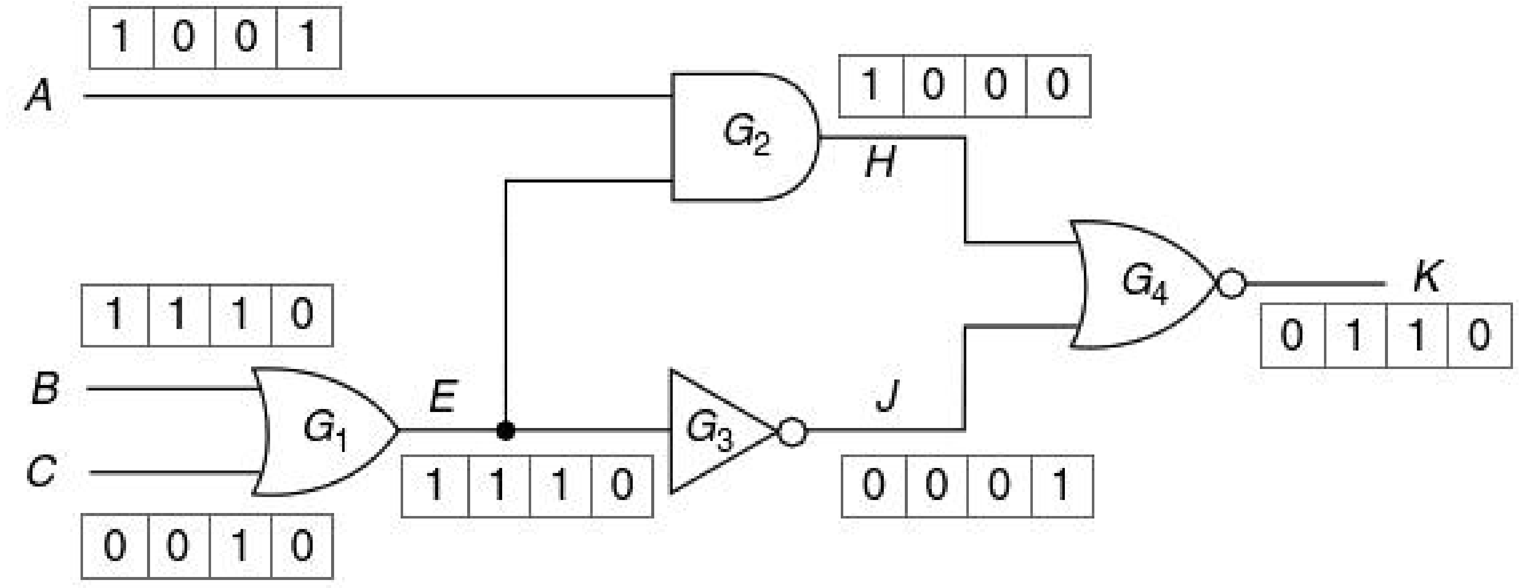
(a) Distributed wire delay model



(b) Fanout delay modeling

3.8 Evaluation of Logic Elements

- A process computing output of a logic element based on its inputs and current state.
  - Truth table is an obvious choice.
  - More advanced techniques to speed up simulation: *input scanning*, *input counting*, **parallel simulation**.



Zoom Tables

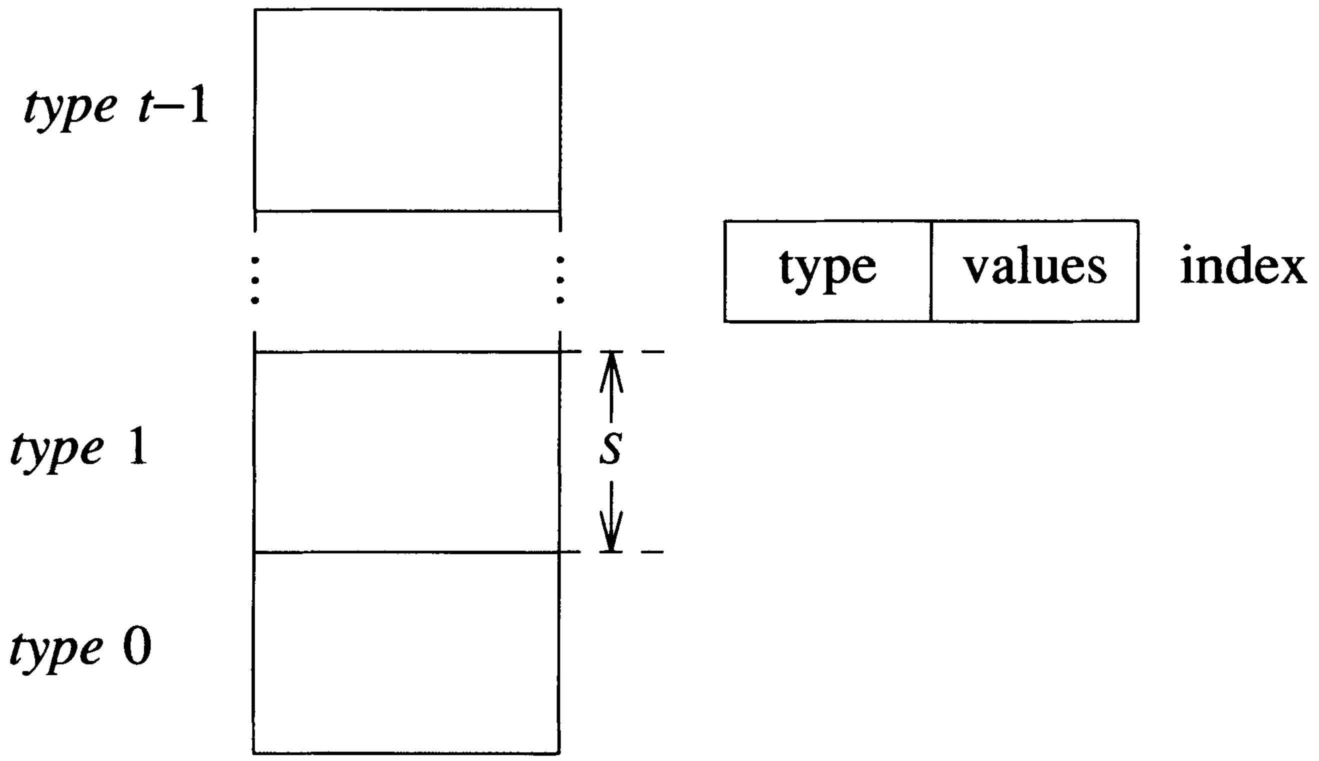


Figure 3.19 Zoom table structure

Logic Gate Characterization

c	x	x	$c \oplus i$
x	c	x	$c \oplus i$
x	x	c	$c \oplus i$
$\bar{c}$	$\bar{c}$	$\bar{c}$	$\bar{c} \oplus i$

	c	i
AND	0	0
OR	1	0
NAND	0	1
NOR	1	1

Figure 3.20 Primitive cubes for a gate with controlling value c and inversion i

- Controlling value ***c*** – uniquely determines output
- Inversion ***i***

Input Scanning

```
$evaluate\ (G, c, i)$
begin
    $u\_values$ = FALSE
    for every input value $v$ of $G$
        begin
            if $v = c$ then return $c \oplus i$
            if $v = u$ then $u\_values$ = TRUE
        end
    if $u\_values$ return $u$
    return $\bar{c} \oplus i$
end
```

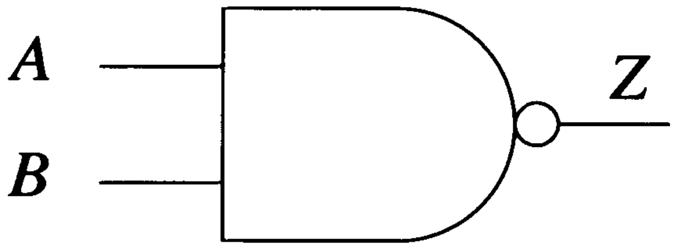
Figure 3.21 Gate evaluation by scanning input values

Input Counting

```
$evaluate\ (G, c, i)$
begin
    if $c\_count > 0$ then return $c \oplus i$
    if $u\_count > 0$ then return $u$
    return $\overline{c} \oplus i$
end
```

Figure 3.22 Gate evaluation based on input counting

3.9 Hazard Detection



Ideally,

A: 0 -> 1, B: 1 -> 0, Z: 1-> 1

Possibly;

A: 0 -> 1 -> 1, B: 1 -> 1 -> 0,
Z: 1 -> 0 -> 1

Hazard Detection

- If wire *S* changes at times *t* and *t*+1, assume its value between *t* and *t*+1 is unknown.
- Evaluate the logic element, check if ***u*** appears on output between *t* and *t*+1.
- Ex: A: 0->***u***->1, B: 1->***u***->0, Z: 1->***u***->1, hazard!

3.9 Hazard Detection

- Static hazards under different delay models.

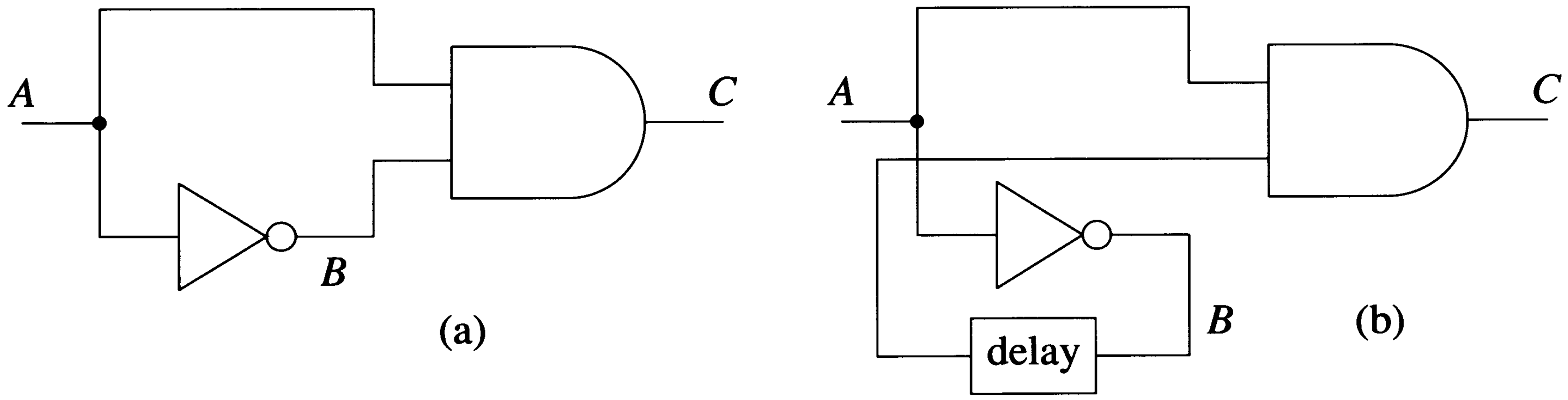


Figure 3.10 (a) Circuit used as pulse generator (b) Correct model for compiled simulation

Consider input sequence A = 010 for (a).

- For 0-delay model, B = 101, and C = 000 – no static hazard
- For unit-delay model, B = 1101, and C = 0010 – static hazard

3.10 Gate-level Event-Driven Simulation

Assume transition independent nominal transport delays

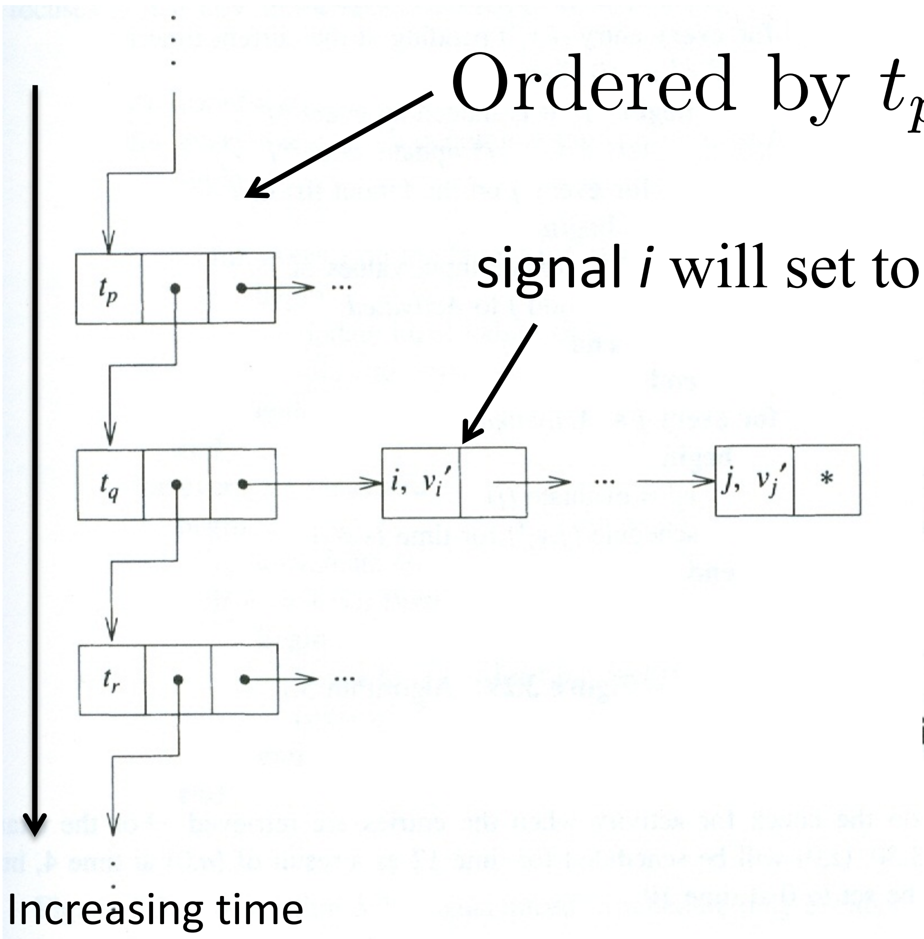

Ordered by $t_p < t_q < t_r < \ldots$
signal *i* will set to value v_i'
t_p
t_q
$i,\ v_i'$
$j,\ v_j'$
*
t_r
Increasing time

Figure 3.27 Event list implemented as a linked list structure

```
while (event list not empty)
    begin
        t = next time in list
        process entries for time t
    end
```

igure 3.28 General structure for event-driven simulation

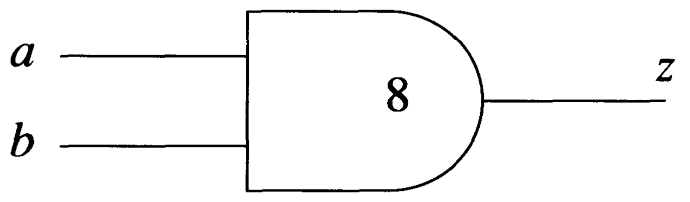
Algorithm 3.1 - Basic Algorithm

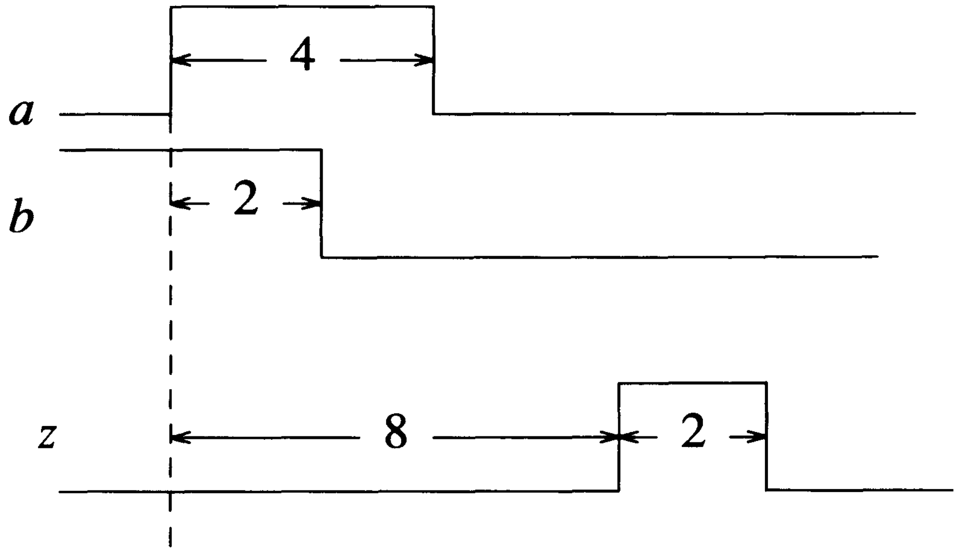
Pass 1:
Find all activated gates for all events at current time

Pass 2:
For each activated gate, evaluated, schedule new events

```
$Activated = \varnothing$   /* set of activated gates */
**for every** entry $(i, v_i')$ pending at the current time $t$
    **if** $v_i' \neq v(i)$ **then**
        **begin**   /* it is indeed an event */
            $v(i) = v_i'$   /* update value */
            **for every** $j$ on the fanout list of $i$
                **begin**
                    update input values of $j$
                    add $j$ to *Activated*
                **end**
        **end**
**for every** $j \in Activated$
    **begin**
        $v_j'$ = evaluate $(j)$
        schedule $(j, v_j')$ for time $t+d(j)$
    **end**
```

Example





- Algorithm 3.1 will schedule two (z,0) events at times t=10 and t=12 (wasteful)
- Does not guarantee scheduling of *only events*

Algorithm 3.3 - One Pass Strategy

```
for every event $(i, v_i')$ pending at the current time $t$
    begin
        $v(i) = v_i'$
        for every $j$ on the fanout list of $i$
            begin
                update input values of $j$
                $v_j' =$ evaluate $(j)$
                if $v_j' \neq lsv(j)$ then
                    begin
                        schedule $(j, v_j')$ for time $t+d(j)$
                        $lsv(j) = v_j'$
                    end
            end
    end
```

Avoids building Activated element list

Avoid redundant events

"*lsv*" is last scheduled value

Problem with Algorithm 3.3

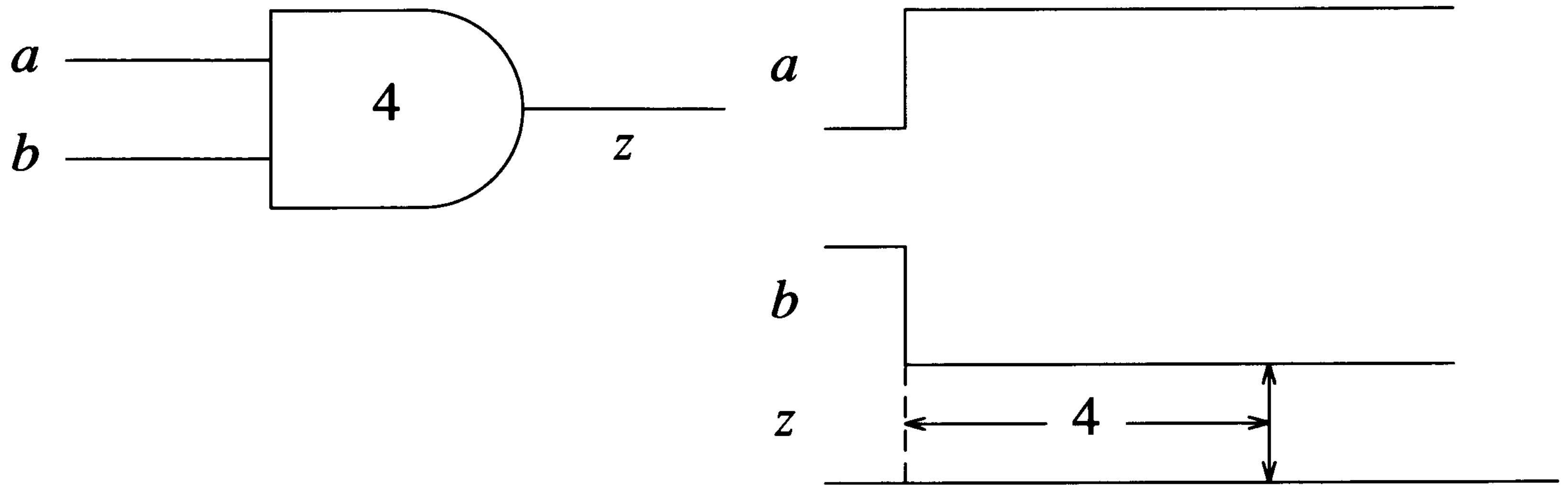


Figure 3.33 Processing of multiple input changes by Algorithm 3.3

When *a* changes before *b*, a zero-spike is generated at time 4!

Algorithm 3.4 – Suppression of zero-width spikes

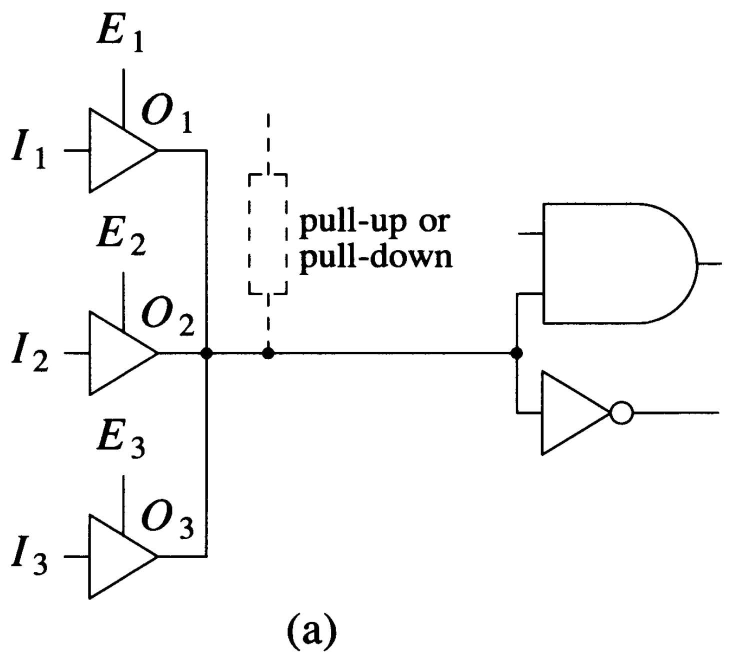
```
for every event $(i, v_i')$ pending at the current time $t$
    begin
        $v(i) = v_i'$
        for every $j$ on the fanout list of $i$
            begin
                update input values of $j$
                $v_j' = evaluate(j)$
                if $v_j' \neq lsv(j)$ then
                    begin
                        $t' = t + d(j)$
                        if $t' = lst(j)$
                            then cancel event $(j, lsv(j))$ at time $t'$
                        schedule $(j, v_j')$ for time $t'$
                        $lsv(j) = v_j'$
                        $lst(j) = t'$
                    end
            end
    end
```

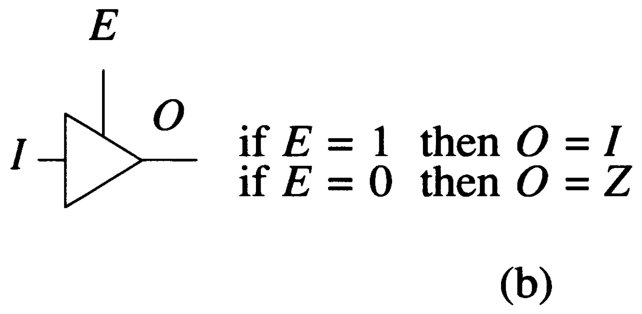
lst: Last Scheduled Time

If two events are scheduled at the same time, cancel the previously schedule one.

Simulating Tri-state Logic

- Only one driver should drive the bus
- Multiple drivers $\rightarrow$ bus conflict (can damage the bus
- If all drivers in Z state $\rightarrow$ bus is either pulled up or down

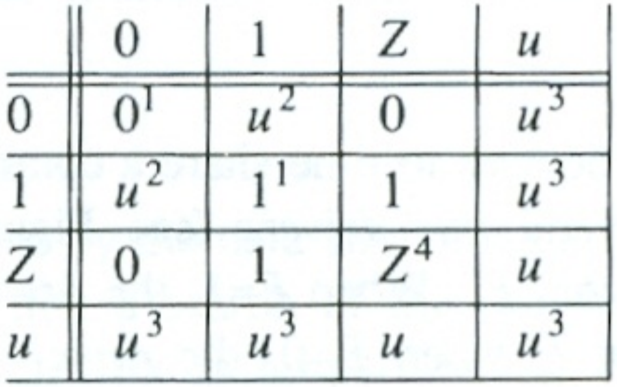




Z: High Impedance

Figure 3.36 (a) Bus (b) bus driver

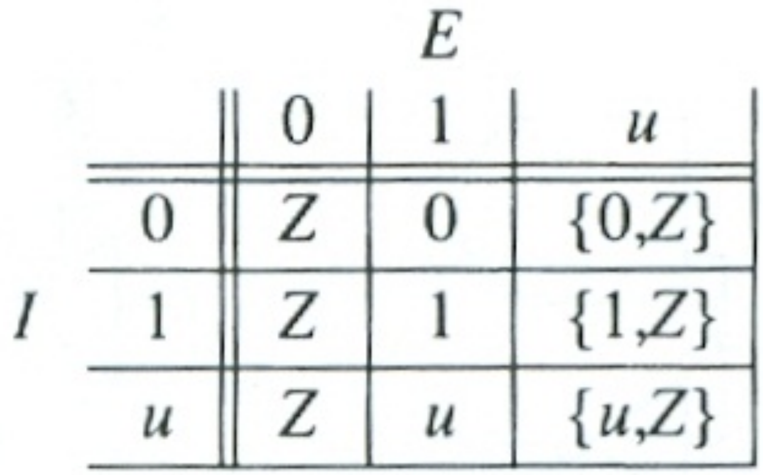
Logic Function – Bus



	0	1	Z	*u*
0	0^1	u^2	0	u^3
1	u^2	1^1	1	u^3
Z	0	1	Z^4	u
u	u^3	u^3	u	u^3

Figure 3.37 Logic function of a bus with two inputs (1 — report multiple drivers enabled; 2 — report conflict; 3 — report potential conflict; 4 — transform to 1(0) if pull-up (pull-down) present)

Logic Function – Bus Driver



		E		
		0	1	*u*
I	0	*Z*	0	{0,*Z*}
	1	*Z*	1	{1,*Z*}
	u	*Z*	*u*	{*u*,*Z*}

Figure 3.38 Truth table for a bus driver

Compiled vs. Event-driven

Compiled	Event-driven
Applicable only to synchronous circuits	Both Synchronous and asynchronous
Allows inputs to change only when circuit is stable	Allows real-time inputs
Inefficient	Efficient – simulates only when required
Cannot handle races/hazards	Can handle races/hazards

Other delay models

- We will not consider the following delay models.
  - Rise & Fall Delay Models
    - Straightforward extension of Algorithms 3.1-3.4. How?
  - Inertial Delay Model
  - Ambiguous Delays
  - Oscillation Control

Summary

- Logic Simulation provides a mechanism to verify the behavior of the system
- Two types of simulation
  - Compiled Simulation
  - Event Driven Simulation (more efficient and commonly used)
- Delay models
  - Transport, Inertial, Rise, Fall, Ambiguous

Backup

Compiled Simulation: Asynchronous Circuits

- Delay modeling in the feedback path is necessary
- Compiled sim model of (a) will not give correct results!

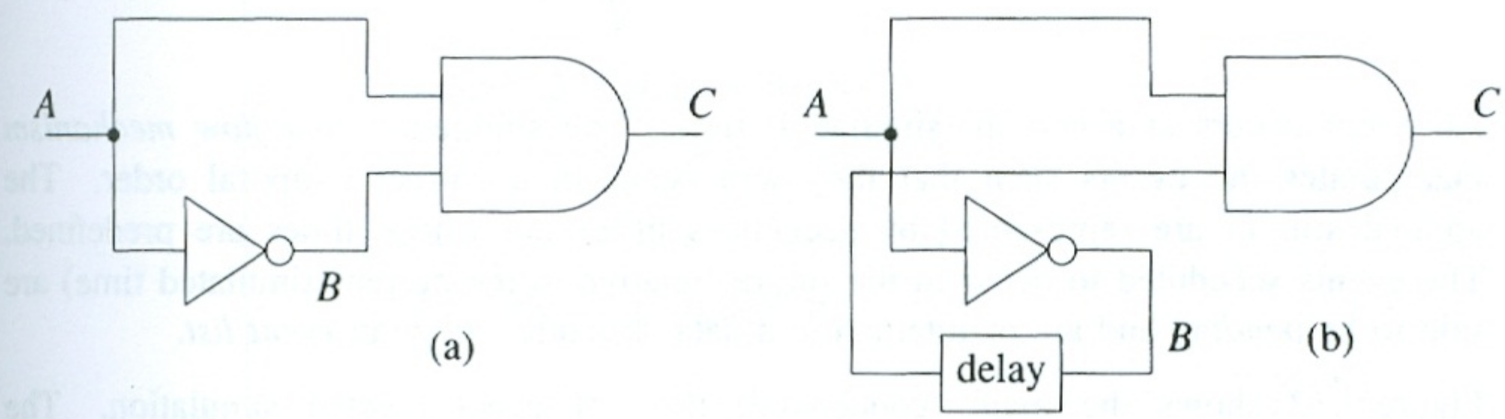
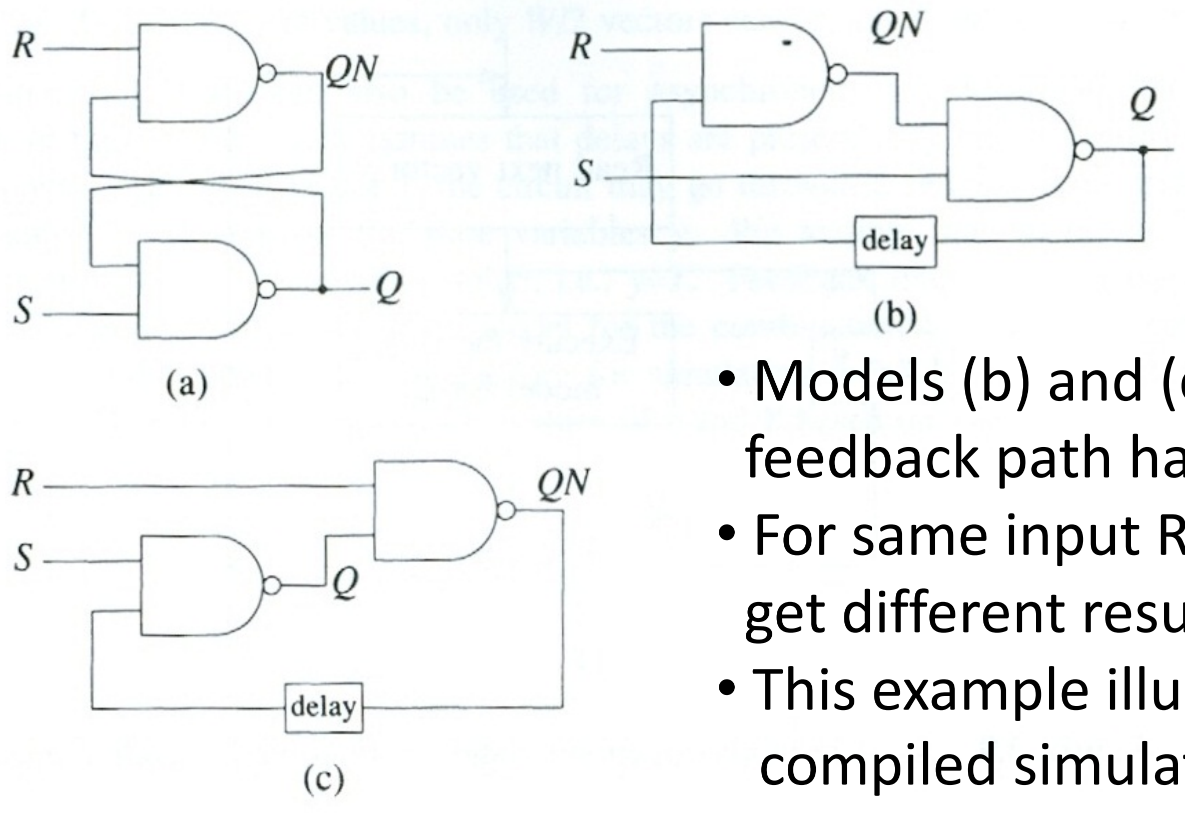


Figure 3.10 (a) Circuit used as pulse generator (b) Correct model for compiled simulation

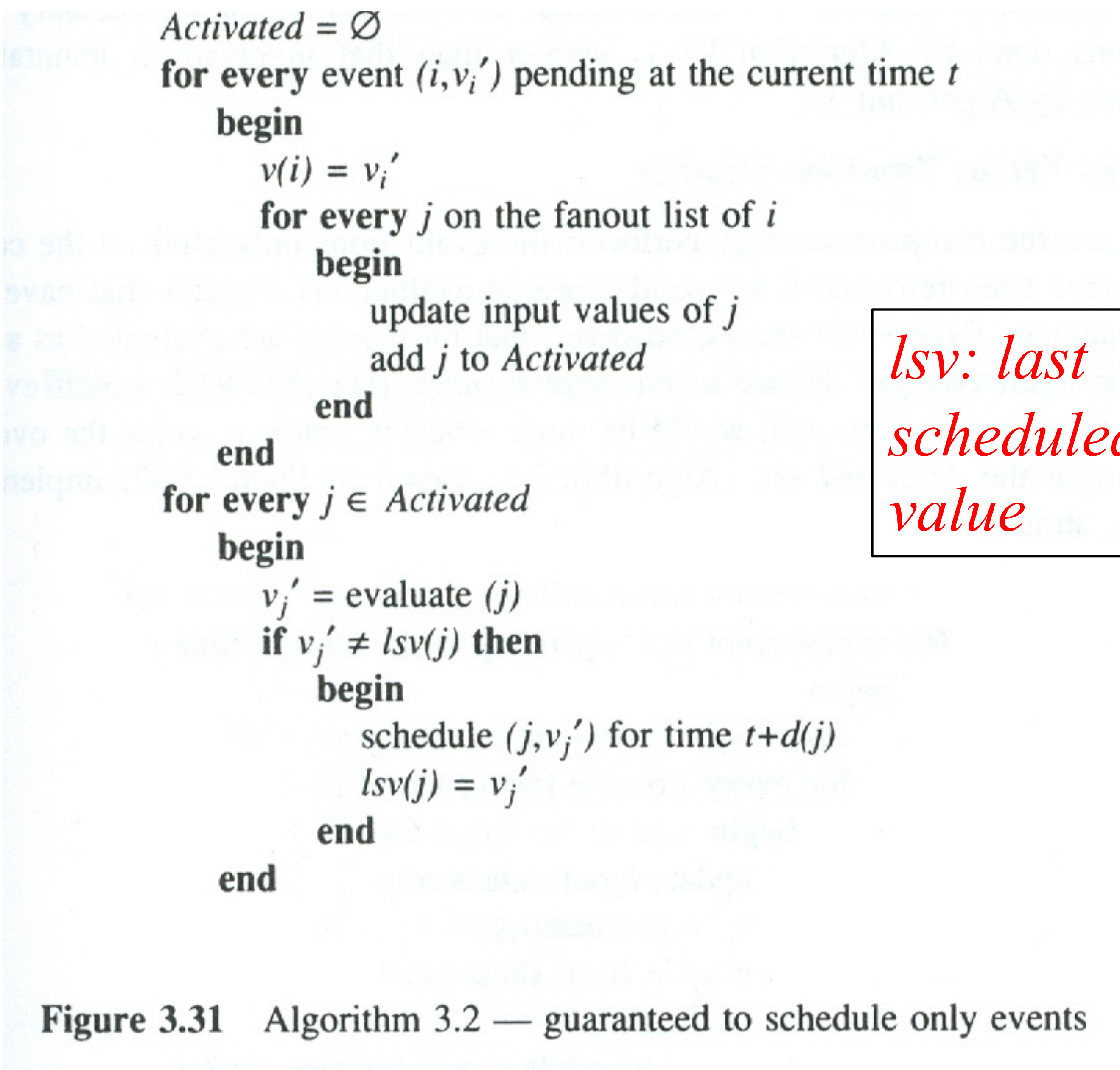
Races/Hazards



- Models (b) and (c) differ in which feedback path has delay
- For same input RS = 00 -> 11, we will get different results!
- This example illustrates that compiled simulation cannot handle races

Figure 3.11 (a) Latch (b)&(c) Possible models for compiled simulation

Algorithm 3.2 - Updated Algorithm



```
$Activated = \varnothing$
for every event $(i, v_i')$ pending at the current time $t$
    begin
        $v(i) = v_i'$
        for every $j$ on the fanout list of $i$
            begin
                update input values of $j$
                add $j$ to $Activated$
            end
    end
for every $j \in Activated$
    begin
        $v_j'$ = evaluate $(j)$
        if $v_j' \neq lsv(j)$ then
            begin
                schedule $(j, v_j')$ for time $t+d(j)$
                $lsv(j) = v_j'$
            end
    end
```

Figure 3.31 Algorithm 3.2 — guaranteed to schedule only events

lsv: last scheduled value