MODULE- 4

Programmable ASIC Interconnect: Actel ACT routing resources, Elmore's constant, RC delay in anti-fuse connections, anti-fuse parasitic capacitance, Synthesis and Simulation with example designs. (7.11,7.1.2,7.1.3, 7.1.4, 12 & 13)

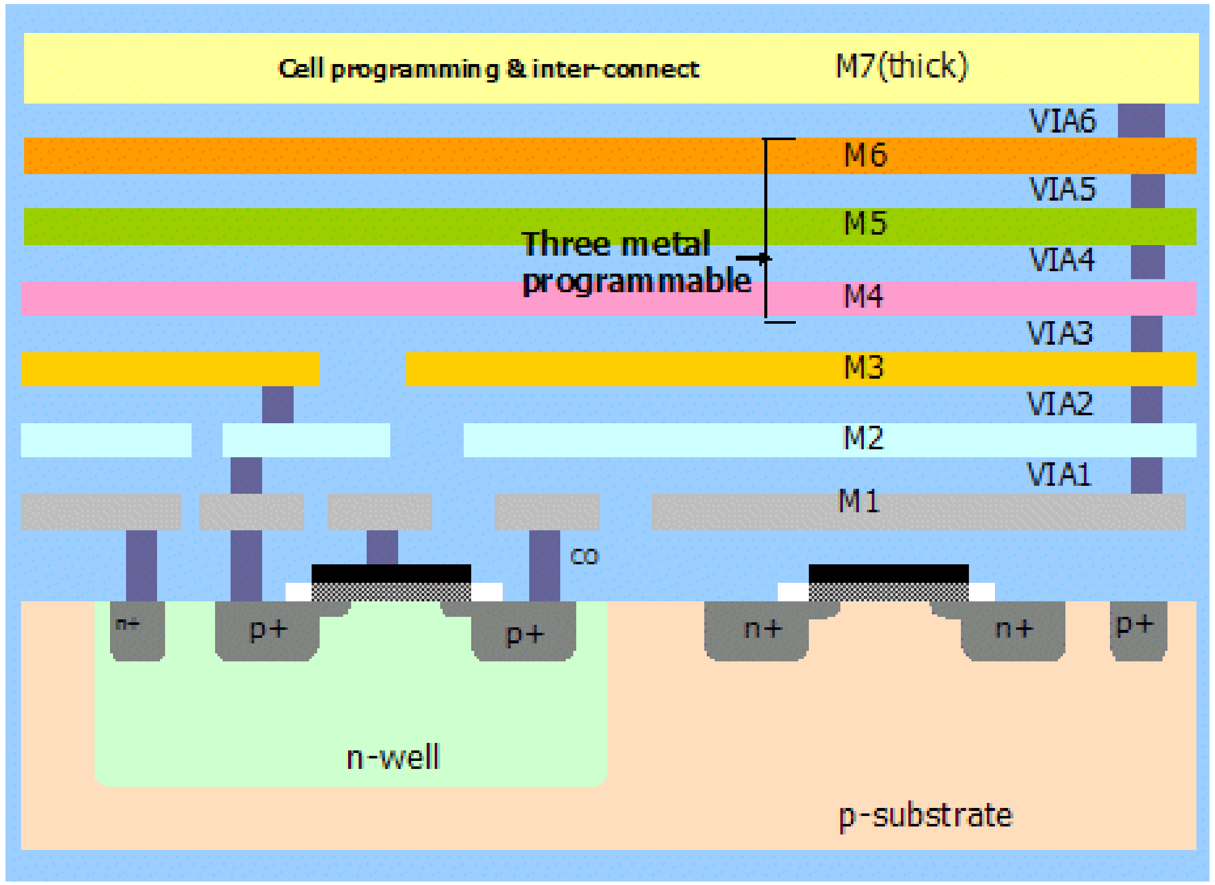
by,

Dr.P.Vimala
Department of ECE
Dayananda Sagar College of Engineering
Bangalore

Figures from M.J.S .Smith, - "**Application - Specific Integrated Circuits**"

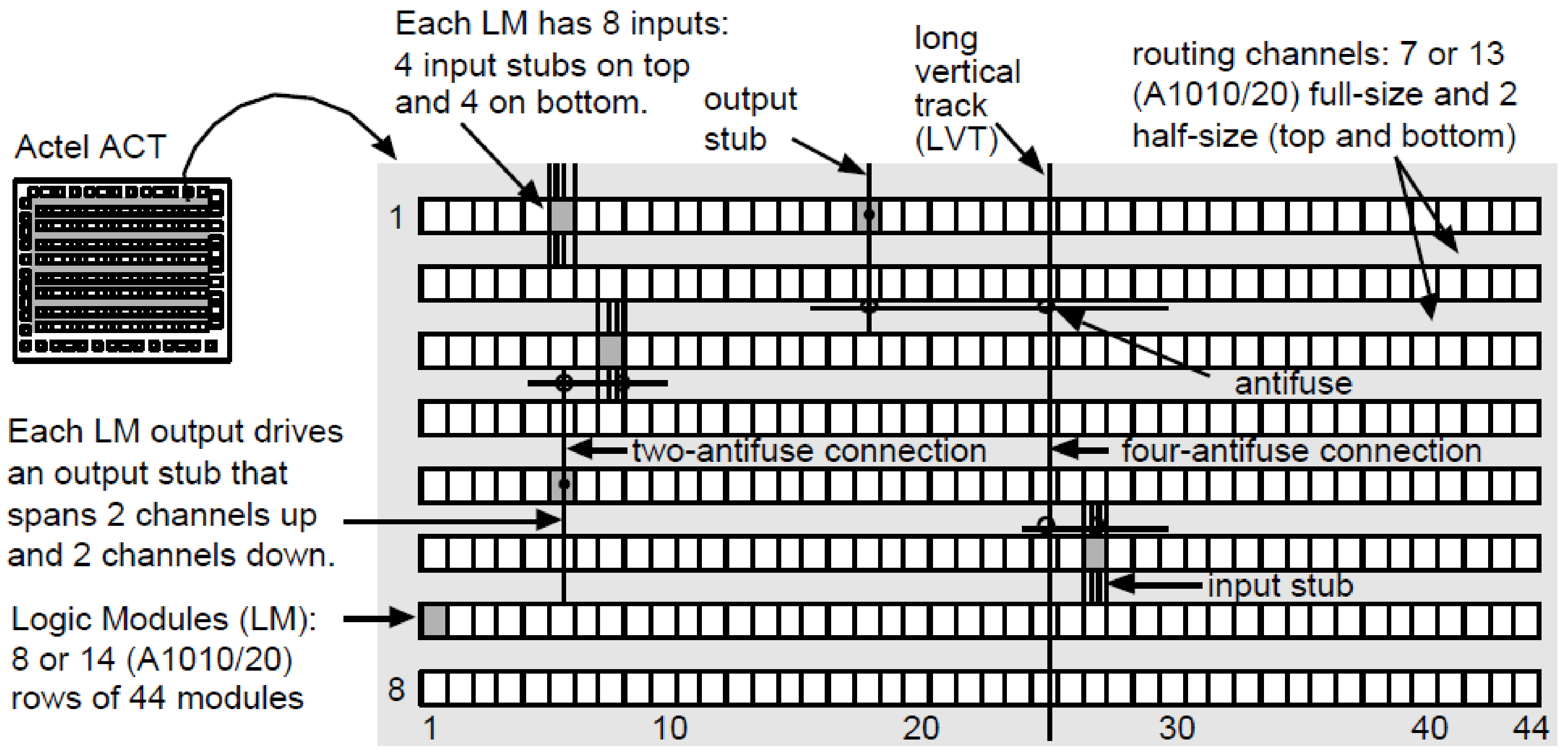
PROGRAMMABLE ASIC INTERCONNECT

- programmable ASICs must have programmable interconnect **to connect cells together for form logic function**
- Structure and complexity of the interconnect is largely **determined by the programming technology** and **the architecture of the basic logic cell.**
- The raw material that we have to work with in building the interconnect is **aluminum-based metallization.**
  - Resistance of approximately 50 mW/square
  - Line capacitance of approximately 0.2 pF/cm
- The first programmable ASICs were constructed using **two layers of metal**; newer programmable ASICs use **three or more layers of metal interconnect**.



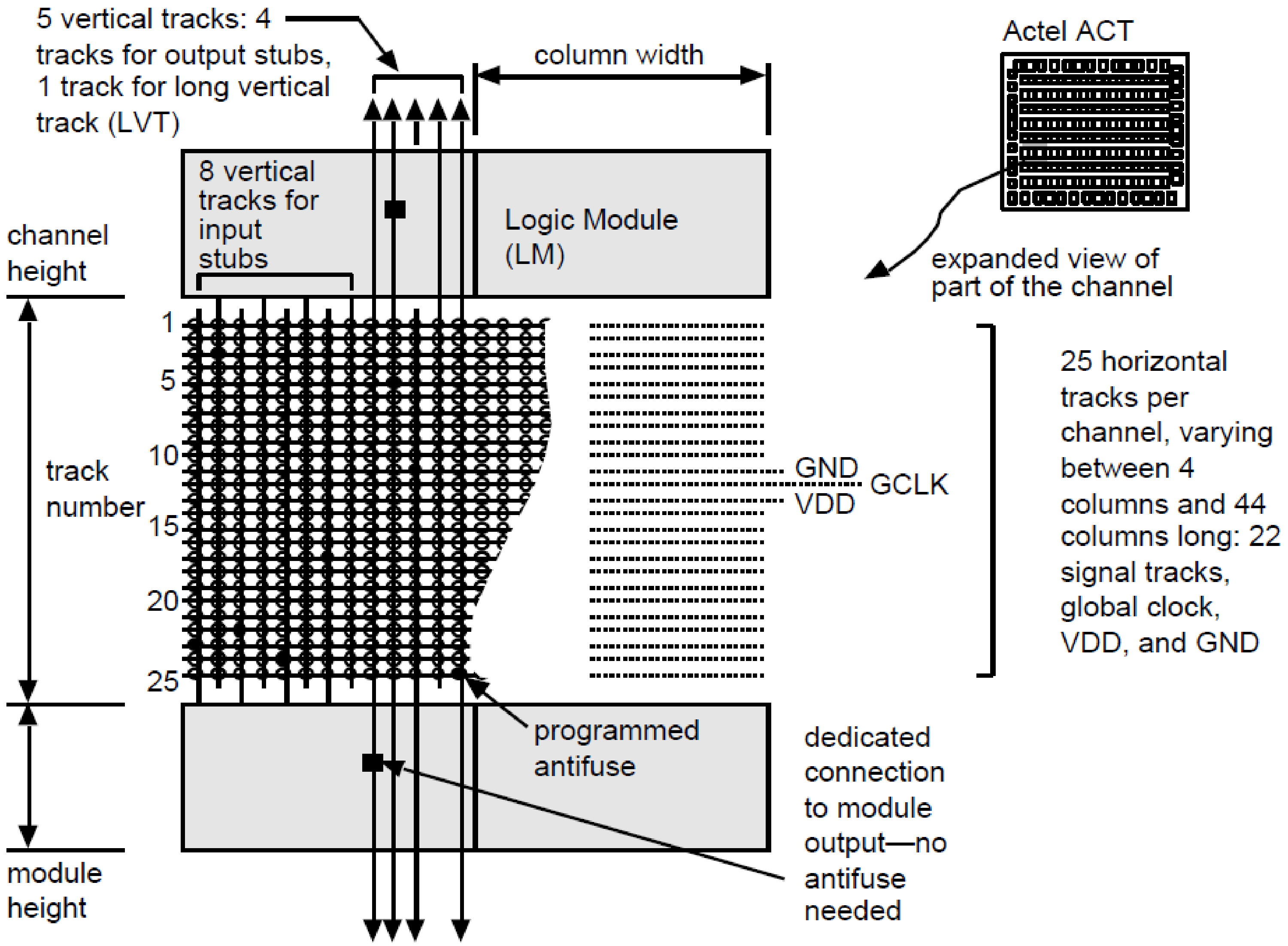
Programming layers for 7 metals process in Metal Programmable Cell Array (MPCA) **technology**

ACTEL ACT


Actel ACT
Each LM has 8 inputs: 4 input stubs on top and 4 on bottom.
output stub
long vertical track (LVT)
routing channels: 7 or 13 (A1010/20) full-size and 2 half-size (top and bottom)
antifuse
two-antifuse connection
four-antifuse connection
input stub
Each LM output drives an output stub that spans 2 channels up and 2 channels down.
Logic Modules (LM): 8 or 14 (A1010/20) rows of 44 modules
1
8
1
10
20
30
40
44

- The interconnect architecture used in an Actel ACT family FPGA.
- similar to a **channeled gate array.**

- The channel routing uses dedicated rectangular areas of fixed size within the chip called **wiring channels (or just channels).**
- The **horizontal channels** run across the chip in the horizontal direction.
- In the vertical direction there are similar **vertical channels** that run over the top of the basic logic cells, the Logic Modules.
- Within the horizontal or vertical channels wires run horizontally or vertically, respectively, within **tracks**
- **Each track holds one wire**.


5 vertical tracks: 4 tracks for output stubs, 1 track for long vertical track (LVT)
column width
Actel ACT
expanded view of part of the channel
8 vertical tracks for input stubs
Logic Module (LM)
channel height
track number
1
5
10
15
20
25
GND
VDD
GCLK
25 horizontal tracks per channel, varying between 4 columns and 44 columns long: 22 signal tracks, global clock, VDD, and GND
programmed antifuse
dedicated connection to module output—no antifuse needed
module height

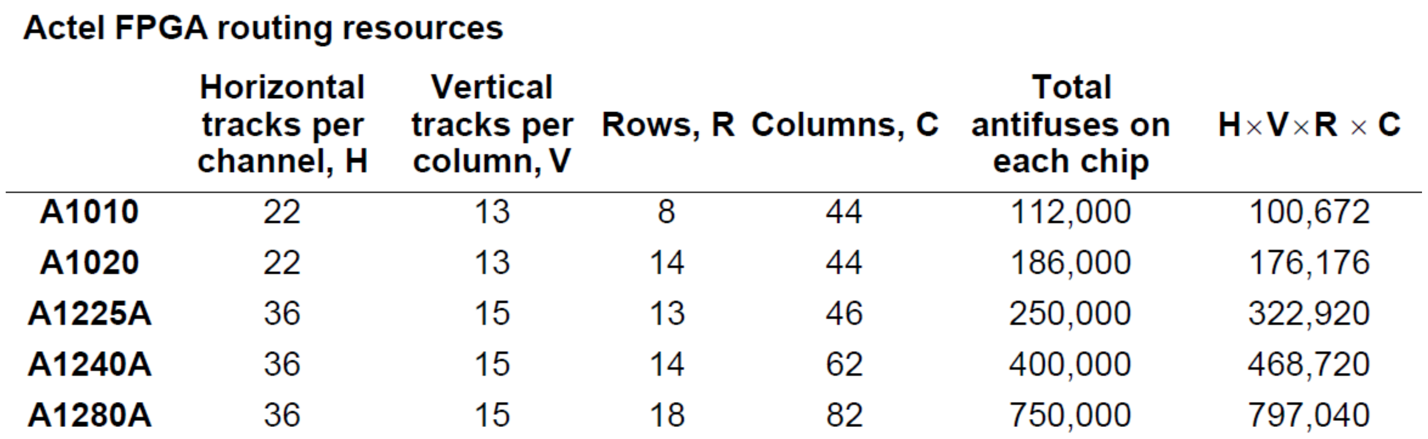
- The capacity of a fixed wiring channel is equal to the **number of tracks** it contains.
- Wires in track are divided into segments of various lengths - **segmented channel routing**
- **Long vertical tracks** (LVT) extend the entire height of the chip
- Each logic module has connections to its inputs and outputs called **stubs**

- **Input stubs** extend vertically into routing channels above and below logic module
- **Output stub** extends vertically 2 channels up and 2 channels down
- **Wires** are connected by antifuses

- In a **channeled gate array the designer decides** the location and length of the interconnect within a channel.
- In an **FPGA the interconnect is fixed** at the time of manufacture.
- **To allow programming of the interconnect**, Actel **divides the fixed interconnect wires within each channel into various lengths** or wire segments.
- We call this segmented channel routing, a variation on channel routing.
- **Antifuses** join the wire segments.
- The designer then **programs the interconnections by blowing antifuses** and making connections between wire segments; **unwanted connections are left unprogrammed.**
- A statistical analysis of many different layouts determines the optimum number and the lengths of the wire segments.

Routing Resources

- The ACT 1 interconnection architecture uses **25 Horizontal tracks** per channel
  - 22 horizontal tracks per channel for signal routing
  - three tracks dedicated to VDD, GND, and the global clock (GCLK)
- **13 vertical tracks** per column in the ACT 1 architecture
  - eight for inputs
  - four for outputs
  - one for an LVT



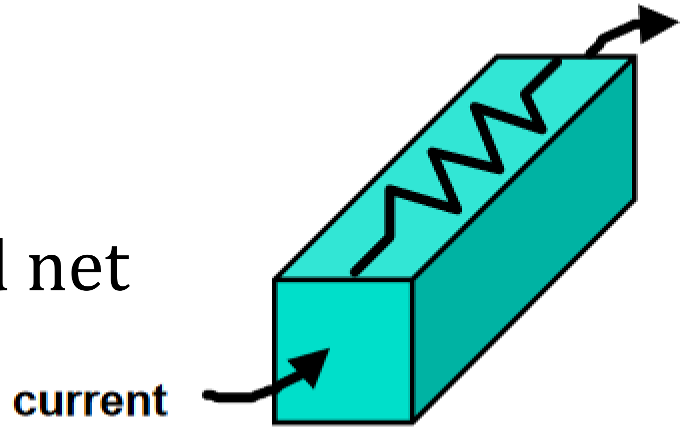
Actel FPGA routing resources

	Horizontal tracks per channel, H	Vertical tracks per column, V	Rows, R	Columns, C	Total antifuses on each chip	$H \times V \times R \times C$
A1010	22	13	8	44	112,000	100,672
A1020	22	13	14	44	186,000	176,176
A1225A	36	15	13	46	250,000	322,920
A1240A	36	15	14	62	400,000	468,720
A1280A	36	15	18	82	750,000	797,040

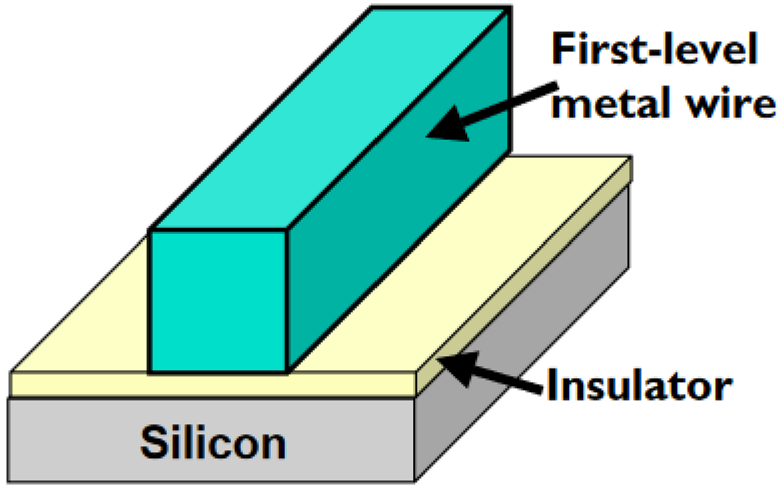
The last two columns show

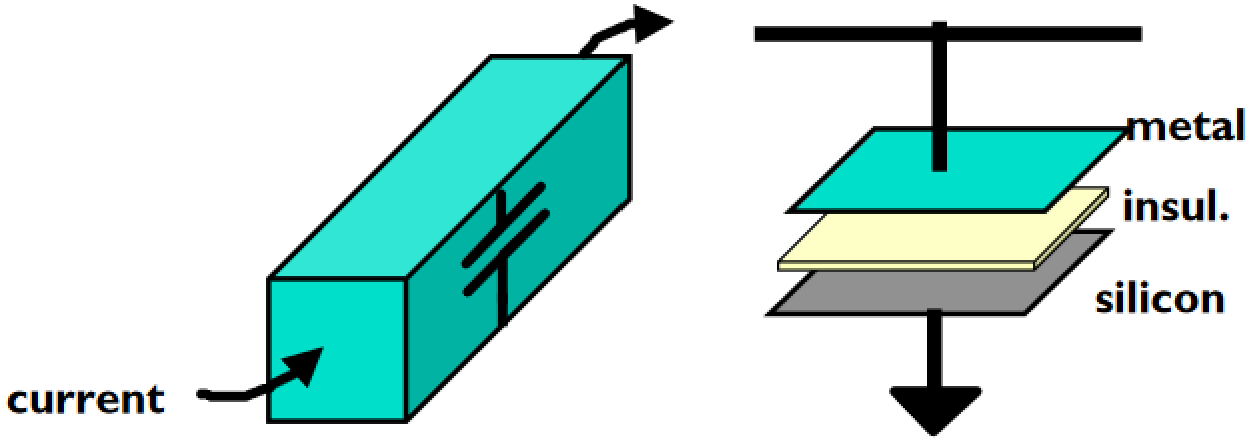
- the total number of antifuses (including antifuses in the I/O cells) on each chip
- the total number of antifuses assuming the wiring channels are fully populated with antifuses (an antifuse at every horizontal and vertical interconnect intersection).

- Delay comes from parasitic loading of the interconnect
- Depends critically on exact shape of the wired net



Metal wire has resistance = R to current flowing down its length



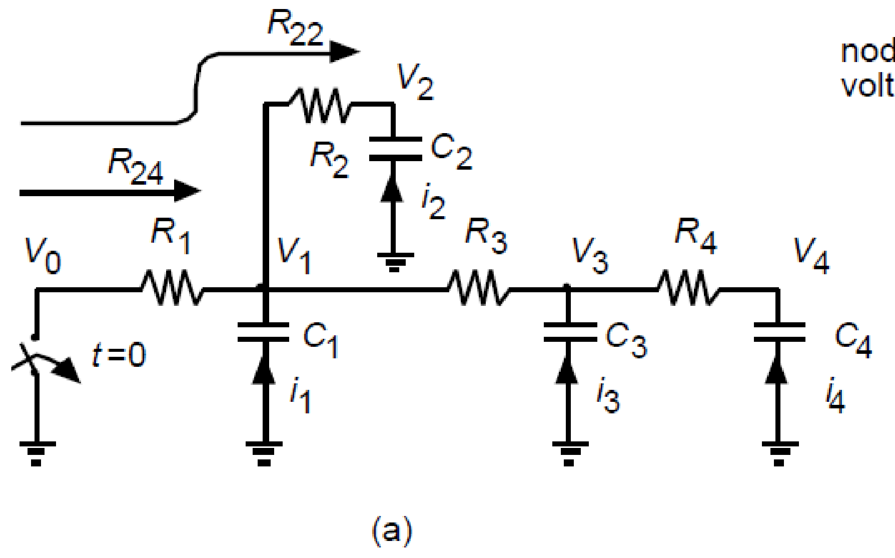


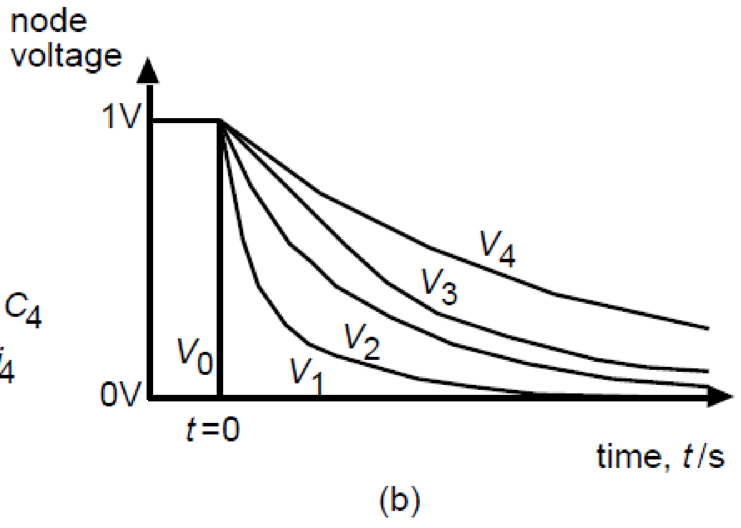
Metal wire has capacitance to silicon substrate, with insulator between

Elmore's Constant

$$V_i(t) = \exp(-t/\tau_{Di}) \quad ; \qquad \tau_{Di} = \sum_{k=1}^{n} R_{ki} C_k$$

The time constant τ_{Di} is often called the **Elmore delay** and is different for each node.





Measuring the delay of a net. (a) An RC tree.
(b) The waveforms as a result of closing the switch at t=0.

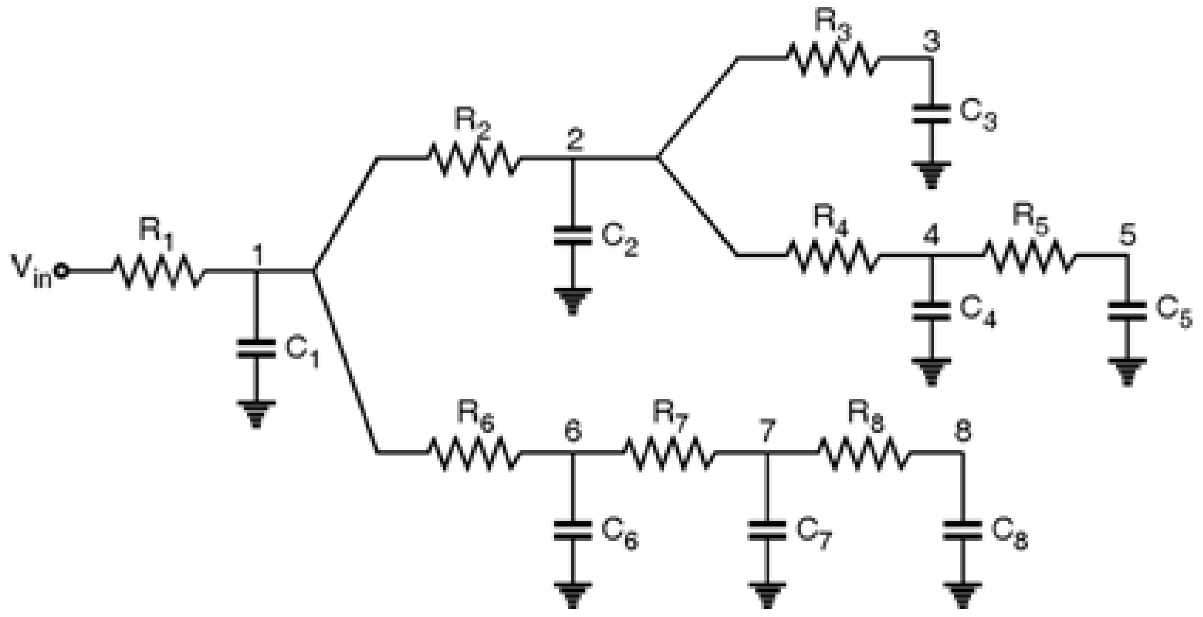
Approximation of waveform at node i:
where R_{ki} is the resistance of the path to V0 shared by node k and node i

Examples:

R24 = R1,

R22 = R1+R2,

R31 = R1



RC tree network

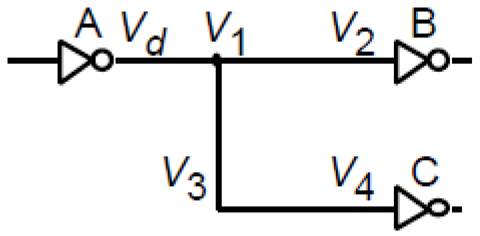
Elmore delay at node 7 is calculated as:

D7 = R1 C1 + R1 C2 + R1 C3 + R1 C4 + R1 C5 +
(R1 + R6) C6 + (R1 + R6 + R7) C7 + (R1 + R6 + R7) C8

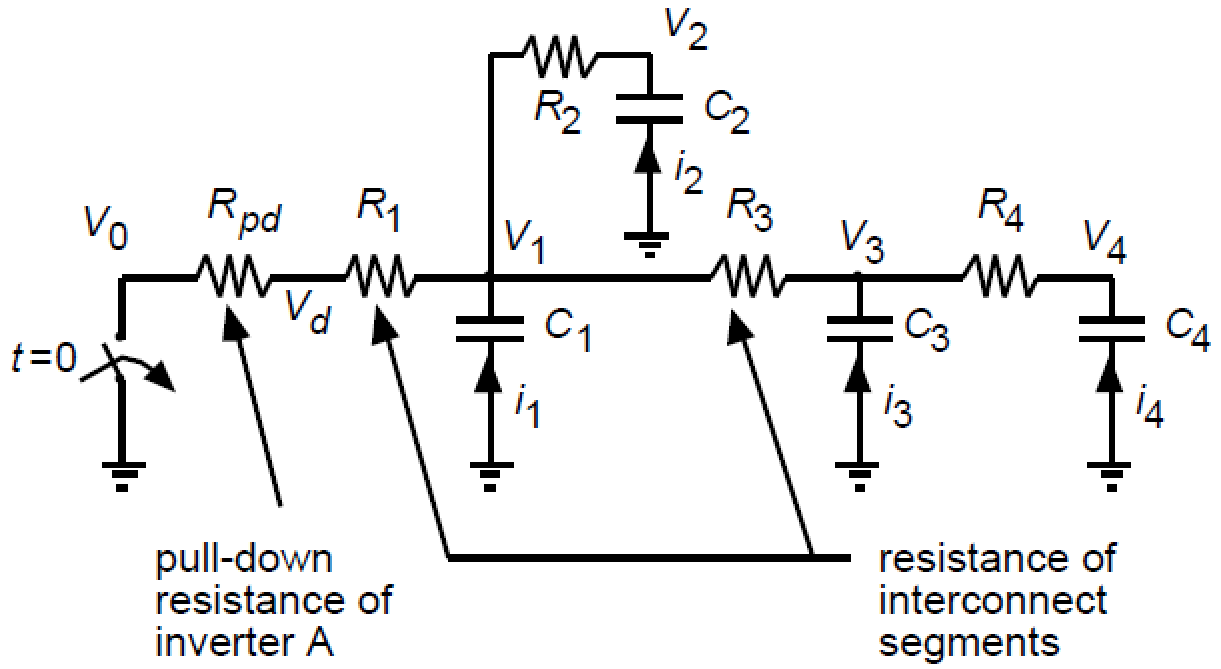
Elmore delay at node 5 is calculated as,

D5 = R1 C1 + (R1 + R2) C2 + (R1 + R2) C3 + (R1 + R2 + R4) C4 +
(R1 + R2 + R4 + R5) C5 + R1 C6 + R1 C7 + R1 C8

A simple circuit with an inverter A driving a net with a fanout of two.

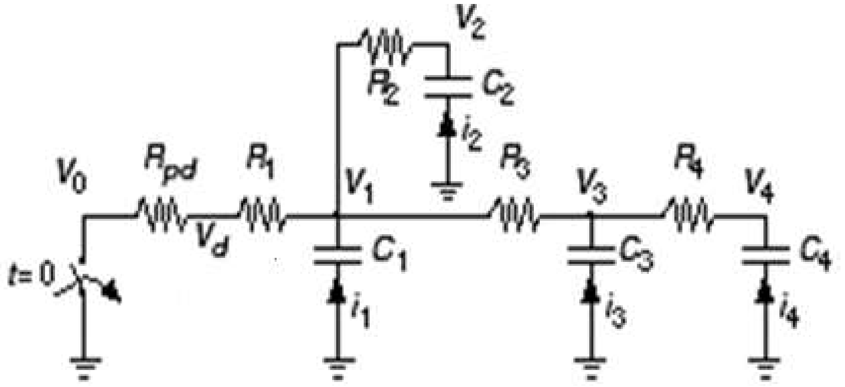


The RC model with each segment replaced by a capacitance and resistance.
The ideal switch and pull-down resistance *Rpd* model the inverter A.



- The Elmore constant for node 4 (labeled V_4) in the network shown in Figure 17.3 (c) i

$$\zeta_4 = \sum_{k=1}^{4} R_{k4} C_k \qquad (17.1)$$



$$= R_{14} C_1 + R_{24} C_2 + R_{34} C_3 + R_{44} C_4 ,$$

- where,

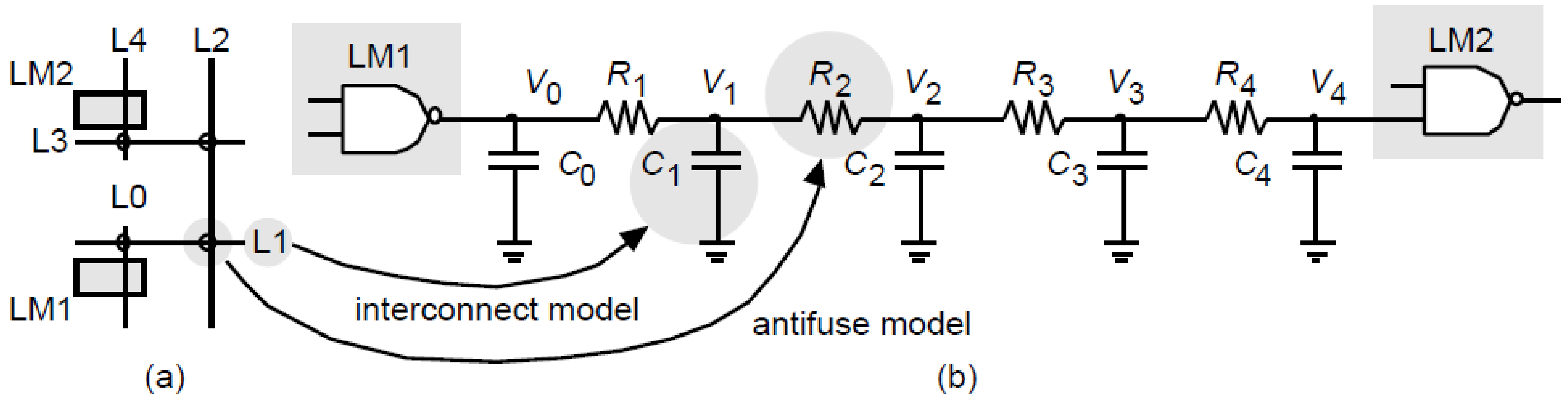
$R_{14} = R_{pd} + R_1$ (resistance to V_0 shared by node 1 and 4) (17.2)

$R_{24} = R_{pd} + R_1$

$R_{34} = R_{pd} + R_1 + R_3$

$R_{44} = R_{pd} + R_1 + R_3 + R_4$

RC Delay in Antifuse Connections



Actel routing model (a) A four-antifuse connection. L0 is an output stub, L1 and L3 are horizontal tracks, L2 is a long vertical track (LVT), and L4 is an input stub **(b)** An RC-tree model. Each antifuse is modeled by a resistance and each interconnect segment is modeled by a capacitance.

For the four-antifuse connection:

$$t_{D4} = R_{14}C_1 + R_{24}C_2 + R_{34}C_3 + R_{44}C_4$$

$$= (R_1 + R_2 + R_3 + R_4)C_4 + (R_1 + R_2 + R_3)C_3 + (R_1 + R_2)C_2 + R_1C_1$$

- If all the **antifuse resistances are approximately equal**, then $R_1 = R_2 = R_3 = R_4 = R$, and the Elmore time constant is,

$$t_{D4} = 4RC_4 + 3RC_3 + 2RC_2 + RC_1$$

- The **capacitance of each interconnect segment is approximately constant**, and equal to C .

    - Two antifuses will generate a 3RC time constant
    - Three antifuses a 6RC time constant
    - Four antifuses gives a 10RC time constant

Antifuse Parasitic Capacitance

- A connection to the diffusion of an Actel antifuse has a **parasitic capacitance due to the diffusion junction.**
- The polysilicon of the antifuse has a **parasitic capacitance due to the thin oxide.**

resistance and capacitance values:

1. The antifuse resistance is assumed to be R = 0.5 k W .
2. C 1 = C 3 = 0.59 pF (0.52 pF due to antifuses, 0.07 pF due to metal) corresponding to a minimum-length horizontal track.
3. C 2 = 4.3 pF (3.5 pF due to antifuses, 0.8 pF due to metal) corresponding to a LVT in a 1020B.
4. The estimated input capacitance of a gate is C 4 = 0.02 pF (the exact value will depend on which input of a Logic Module we connect to).

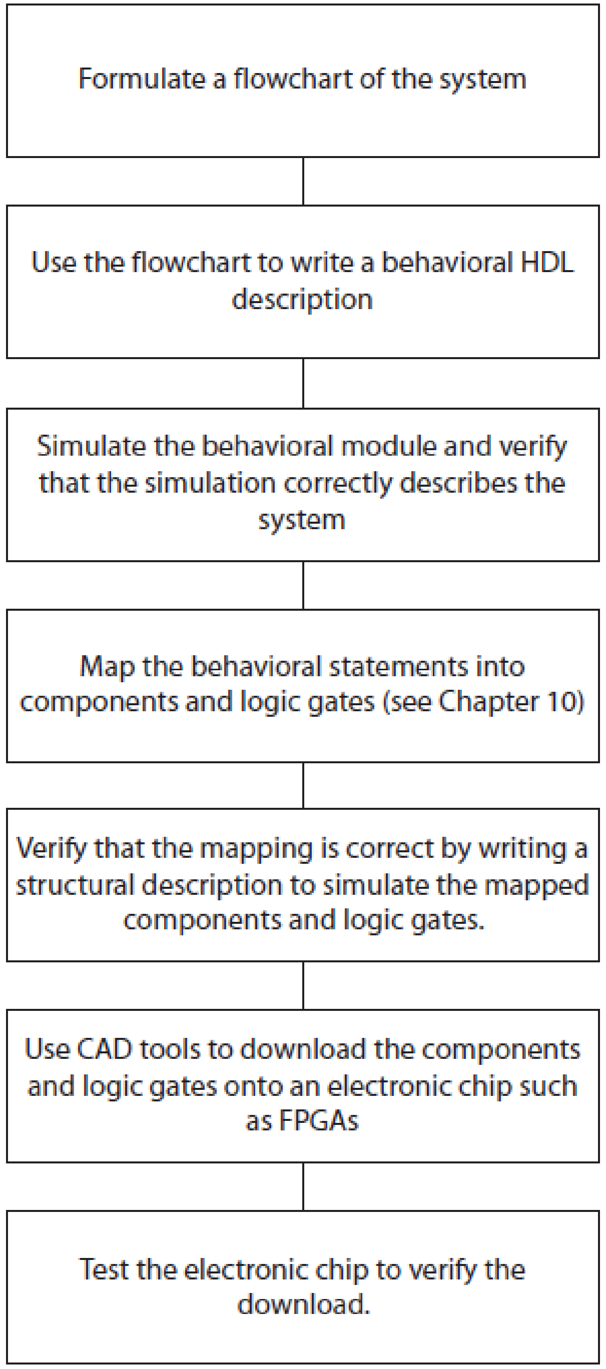
$$\begin{aligned} t_{D4} &= 4(0.5)(0.02) + 3(0.5)(0.59) + 2(0.5)(4.3) + (0.5)(0.59) \\ &= 5.52 \text{ ns} . \end{aligned}$$

This matches delays obtained from the Actel delay calculator.

Logic synthesis

- Logic synthesis provides a link between an HDL (Verilog or VHDL) and a netlist.
- Designers use **graphic or text design entry to create an HDL behavioral model** , which does not contain any references to logic cells.
- The behavioral model is **simulated** to check that the design meets the specifications and then **the logic synthesizer** is used to generate a netlist, a structural model , which contains only references to logic cells.
- **Layout for any type of ASIC may be generated** from the structural model produced by logic synthesis.
- Synthesis coverts **HDL behavioural code into logic gates or components.**
- These logic gates and components can be downloaded into an electronic chip.
- Synthesis **maps between the simulation (S/w) domain and the hardware domain.**

General synthesis steps


Formulate a flowchart of the system
Use the flowchart to write a behavioral HDL description
Simulate the behavioral module and verify that the simulation correctly describes the system
Map the behavioral statements into components and logic gates (see Chapter 10)
Verify that the mapping is correct by writing a structural description to simulate the mapped components and logic gates.
Use CAD tools to download the components and logic gates onto an electronic chip such as FPGAs
Test the electronic chip to verify the download.

Step 1: If the behavioral description of the system is available, go to Step 3.Otherwise, formulate a flowchart for the behavior of the system.

Step 2: Use the flowchart to write a behavioral description of the system. Be sure to review the instructions of your synthesis tools to see if there are constraints on any of the behavioral statements you plan to use.

Step 3: Simulate the behavioral code and verify that the simulation correctly describes the system.

Step 4: Map the behavioral statements into components or logic gates (this chapter shows you how to do that). Be sure that the components used are downloadable into the selected chip.

Step 5: Write a structural- or gate-level description of the components and logic gates of Step 4. Simulate the structural description and verify that this simulation is similar to that of Step 3.

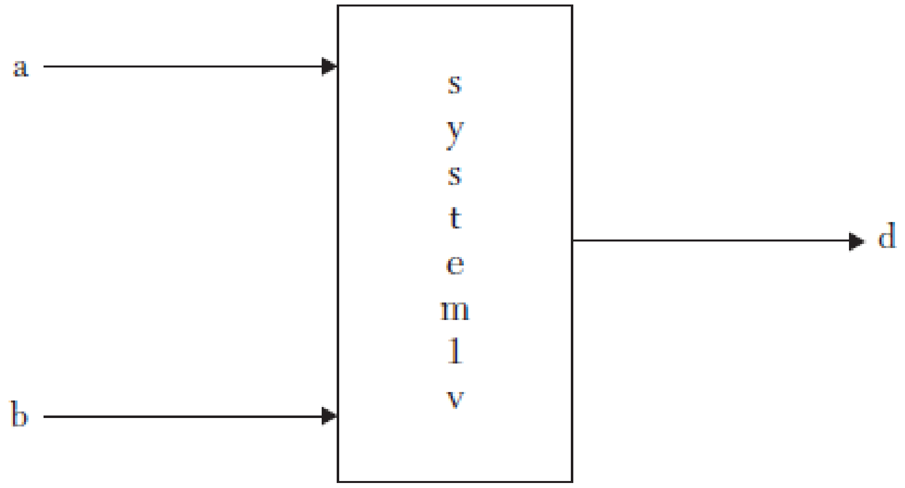
Step 6: Use CAD tools to download the gates and components of Step 4 into the electronic chip, usually a FPGA chip.

Step 7: Test the chip by inputting signals to the input pins of the chip and observe the output from the output pins. This step is similar to the verification done in Step 5, except the test here is on real, physical signals.

Verilog Synthesis Information From Module Inputs/Outputs

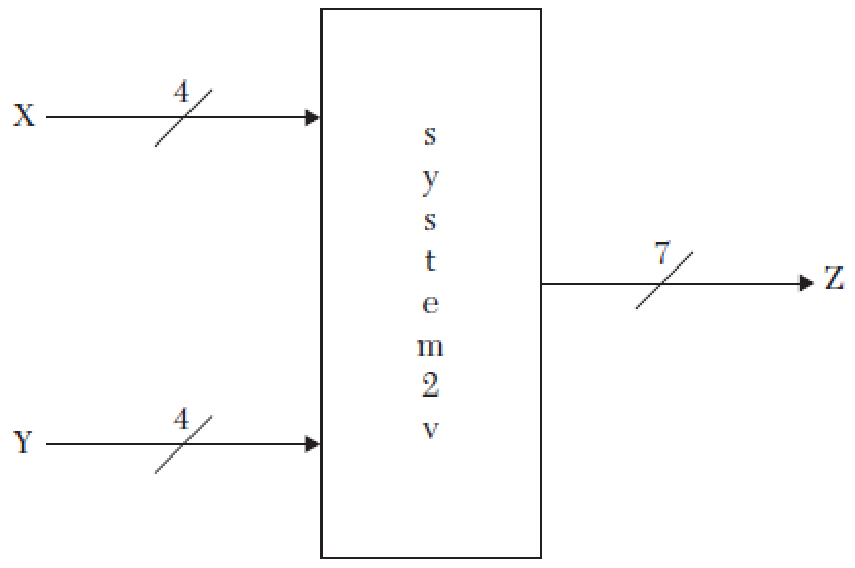
```
module system1v (a, b, d);
input a, b;
output d;
endmodule
```

- system1v has two input signals, a and b, each of one bit, and one output signal d of one bit.
- All signals can take 0, 1, or high impedance.



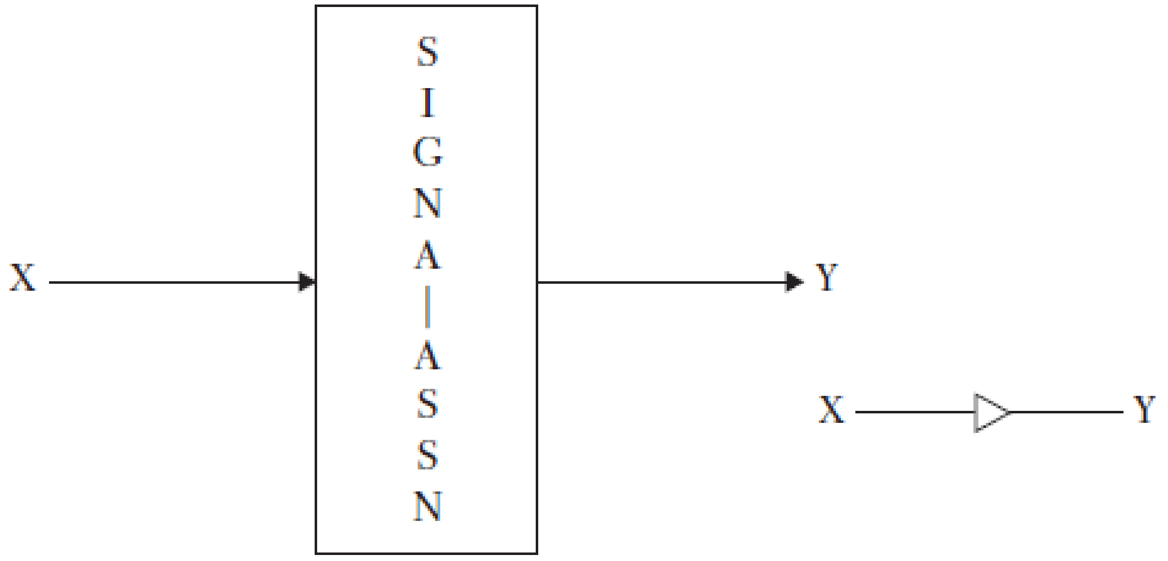
```
module system2v (X, Y, Z);
input [3:0] X, Y;
output [7:0] Z;
reg [7:0] Z
........
endmodule
```

- system2v with two input signals, X and Y, each of four bits, and one output signal, Z, of eight bits.
- The statement reg [7:0] Z; does not convey any additional information to the hardware domain;
- its use is solely for simulation.



Mapping Always in the Hardware Domain

```
module SIGNA_ASSN (X, Y);
input X;
output Y;
reg y;
always @ (X)
begin
Y = X;
end
endmodule
```



a) Logic symbol. b) Gate-level logic diagram.

1. Write Verilog code for signal assignment statement Y=2*X+3 for 2 bit input. Show the synthesized logic symbol and gate level diagram. Write structural code in Verilog using gate level diagram.
2. Write Verilog description for 3:8 decoder using case statement and show the synthesized logic symbol and gate level diagram.
3. Write Verilog description for 8:3 encoder using case statement and show the synthesized logic symbol and gate level diagram.
4. Draw the gate level synthesis information extracted from 8:1 Multiplexer. Also write the Verilog code using case statement.
5. Draw the gate level synthesis information extracted from 1:8 De-Multiplexer. Also write the Verilog code using case statement.

6. Find the gate level synthesis mapping for a 3bit input ‘a’ and single bit output ‘y’ has the condition as output is high when the input value is less than 5 otherwise low.

7. Explain the mapping of else if statement by taking the following example.
The system with BP (input) and ADH (output) are of natural type ranging 0 to 7 and 0 to 15 respectively. Assume that if BP less than or equal to 2, then ADH=15 or if BP greater than 5 then ADH is 0 otherwise ADH = BP *(-5) +25.

8. Draw the gate level synthesis information extracted from the following Verilog code

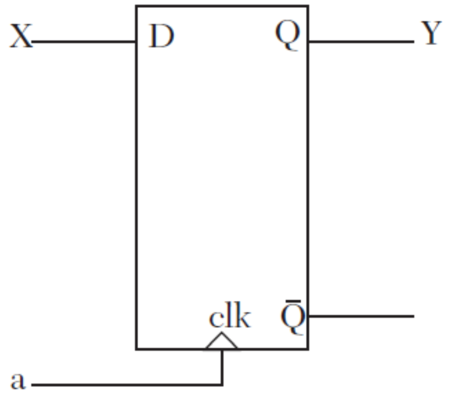
```
always @ (a, X, X1)
begin
if (a == 1'b1)
Y = X;
else
Y = X1;
end
```

9. Write Verilog code for y=2*a+5 for 3-bit input. If a is greater than four, y retains its previous value. Also, show the synthesized gate level diagram and logic symbol for the given example.

```
module If_state (a, X, Y);
input a, X;
output Y;
reg Y;
always @ (a, X)
begin
if (a == 1'b1)
Y = X;
end
endmodule
```

- when a = 0, there should be no change in the values of any signal. This means that the value of all signals should be stored during the execution of the IF statement.
- To store signals in the hardware domain, latches or flip-flops are used.

- signal a is implemented as a clock to a D-latch
- input to the latch is the signal X.
- If a = 0, then the output of the latch stays the same.
- If a = 1, then the output follows the input X.



Gate level diagram

Simulation

- Engineers used to prototype systems to **check designs**
- **Breadboarding** is feasible for systems constructed from a few TTL parts
- It is impractical for an ASIC
- Instead engineers turn to **simulation**

Simulators are usually divided into the **following categories or simulation modes** :

1. Behavioral simulation
2. Functional simulation
3. Static timing analysis
4. Gate-level simulation
5. Switch-level simulation
6. Transistor-level or circuit-level simulation

- The list is ordered **from high-level to low-level simulation** (high-level being more abstract, and low-level being more detailed).

- Proceeding from high-level to low-level simulation, the simulations **become more accurate**, but they also become progressively **more complex** and take longer to run.

- While it is just **possible to perform a behavioral-level simulation** of a complete system, it is **impossible to perform a circuit-level simulation** of **more than a few hundred transistors**.

- **Behavioral simulation:** Models large pieces of a system as black boxes with inputs and outputs. This type of simulation often using VHDL or Verilog.

- **Functional simulation:** Ignores timing and includes unit-delay simulation , which sets delays to a fixed value (for example, 1 ns).

Static timing analysis: Once a behavioral or functional simulation predicts that a system works correctly, the next step is **to check the timing performance.**

- At this point a system is **partitioned into ASICs** and a timing simulation is performed for each ASIC separately (otherwise the simulation run times become too long).
- One class of timing simulators employs timing analysis that analyzes **logic in a static manner, computing the delay times for each path.**
- This is called static timing analysis because it **does not require the creation of a set of test (or stimulus) vectors .**
- Timing analysis works **best with synchronous systems** whose maximum operating frequency is determined by the longest path delay between successive flip-flops.
- The path with the longest delay is the **critical path .**

- **Gate-level simulation :** Also known as Logic simulation
  - Logic simulation can also be **used to check the timing performance of an ASIC**.
  - In a gate-level simulator a **logic gate or logic cell** (NAND, NOR, and so on) is **treated as a black box modeled by a function** whose variables are the input signals.
  - The function may also **model the delay through the logic cell.**
  - Setting all the delays to unit value is the equivalent of **functional simulation.**

- **Switch-level simulation:**
  - If the timing simulation provided by a black-box model of a logic gate is not accurate enough, the **next, more detailed, level of simulation** is switch-level simulation which models transistors as switches—on or off.
  - Switch-level simulation **can provide more accurate timing predictions** than gate-level simulation, but **without the ability to use logic-cell delays** as parameters of the models.
- **Transistor-level simulator:**
  - The **most accurate, but also the most complex and time-consuming**, form of simulation is transistor-level simulation .
  - A transistor-level simulator requires **models of transistors**, **describing their nonlinear voltage and current characteristics.**

- Each type of simulation normally uses a **different software tool.**
- A **mixed-mode simulator** permits different parts of an ASIC simulation to use different simulation modes.
  - For example, a critical part of an ASIC might be simulated at the transistor level while another part is simulated at the functional level.
  - Be careful not to confuse **mixed-level simulation** with a **mixed analog/digital simulator**, these are mixed-level simulators .
- Simulation is used at **many stages during ASIC design**.
- Initial **prelayout simulations** include logic-cell delays but **no interconnect delays.**
- Estimates of capacitance may be included after completing logic synthesis, but only after **physical design** is it possible to perform an **accurate postlayout simulation** .