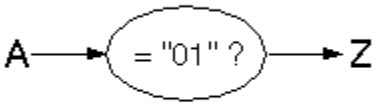
# Hardware Description Languages

## Basic Concepts

مرتضی صاحب الزمانی

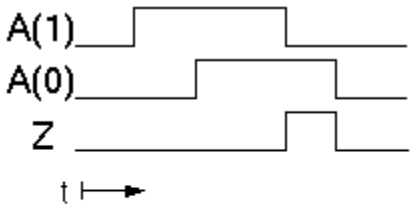
# Application of HDLs

Modelling

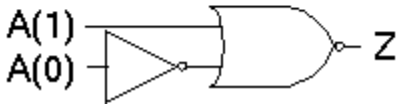



```
Z <= '1' when A="01"
     else '0' ;
```

Simulation

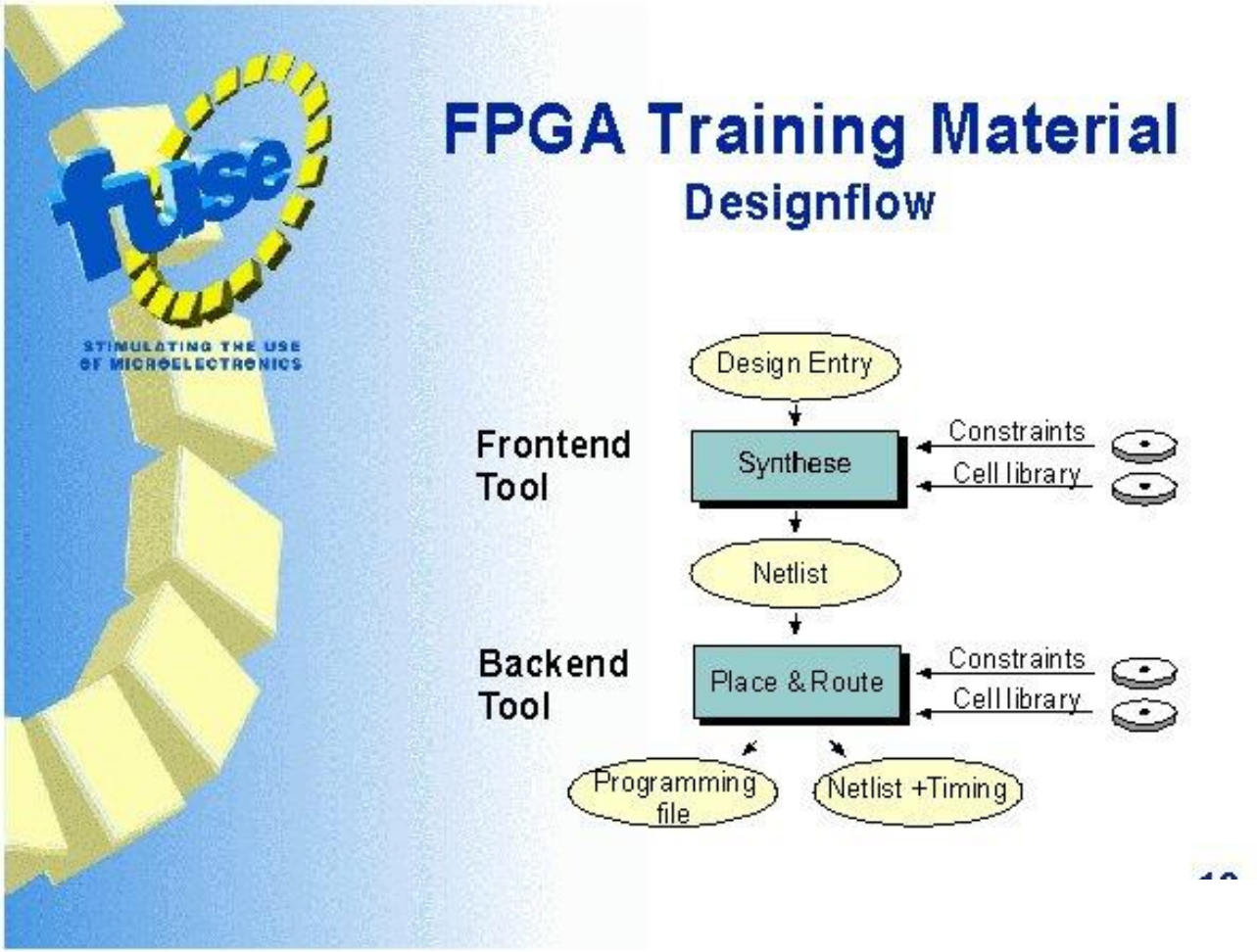



Synthesis



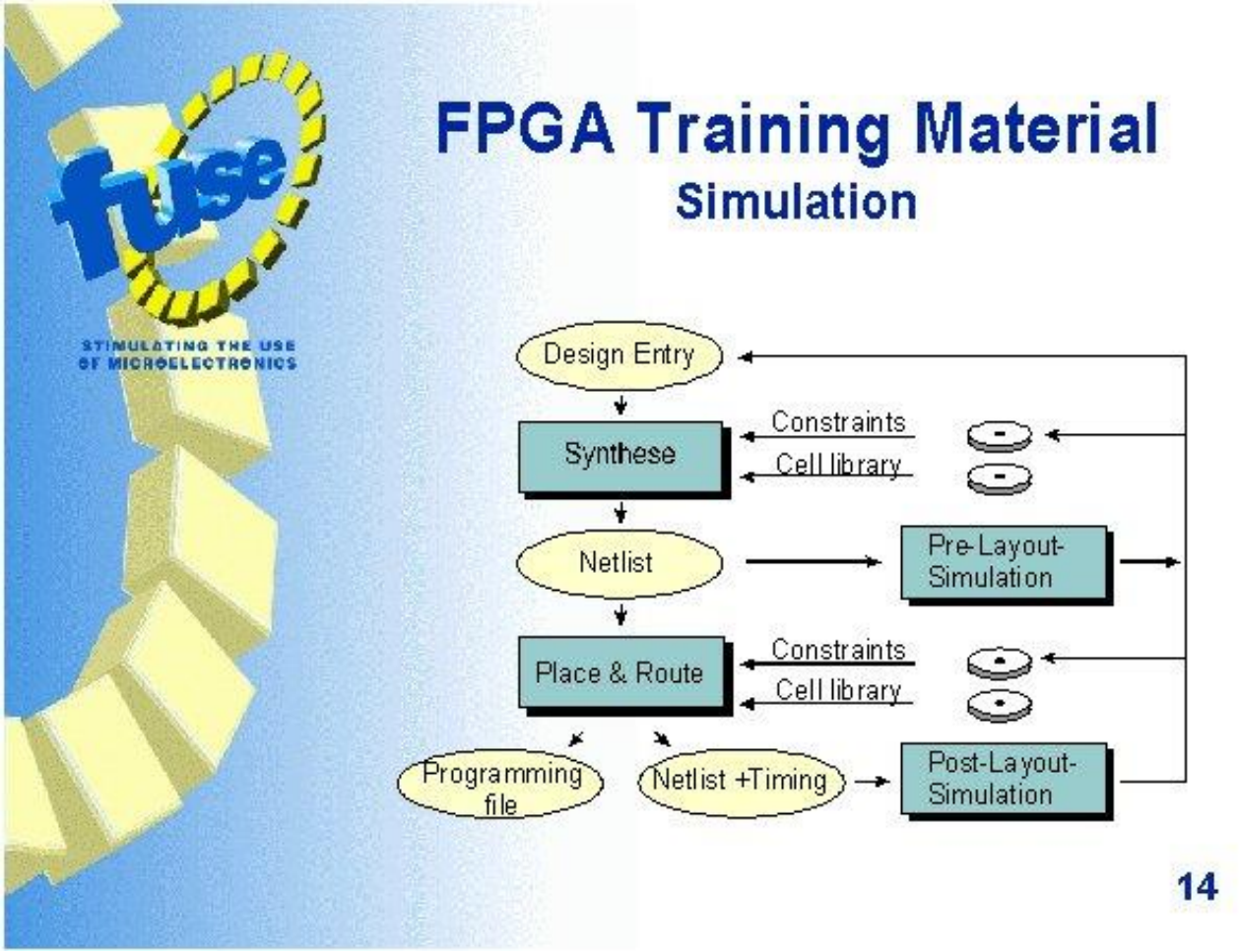

- VHDL به عنوان زبان مستندسازی (توصیف فرمال و بدون ابهام).
- سنتز: تبدیل (اتوماتیک یا دستی) یک توصیف به توصیفی با جزئیات بیشتر

# A digital system design process




مرتضی صاحب الزمانی

# Simulation


fuse
STIMULATING THE USE OF MICROELECTRONICS
FPGA Training Material
Simulation
Design Entry
Synthese
Constraints
Cell library
Netlist
Pre-Layout-Simulation
Place & Route
Constraints
Cell library
Programming file
Netlist +Timing
Post-Layout-Simulation
14

# VHDL - Overview

- **Very High Speed Integrated Circuit Hardware Description Language**
  - **Modeling of digital systems**
  - **Concurrent and sequential statements**
  - **Design lifetime > Designer lifetime**
  - **Man- and machine-readable documentation**
- **International Standards**
  - **IEEE Std 1076-1987**
  - **IEEE Std 1076-1993, 2000, 2002, …**

# VHDL - History

- **2003: VHDL-200X by VASG (VHDL Analysis and Standardization Group)**
  - **Responsible for maintaining and extending the VHDL standard (IEEE 1076).**
- **2005: Accellera VHDL TSC took over.**
- **2006: Accellera VHDL-2006-D3.0 approved**
  - **DATE 2007: date_vhdl_tutorial.pdf**
- **Pure definition of language in the LRM (Language Reference Manual)**
- **No standards for application or methodology**

# VHDL Standards

- علاوه بر استانداردهای خالص, تلاشهایی برای استاندارد کردن عوامل مربوط به VHDL انجام گرفته است:
- پکیج های
  - Std_logic_1164
  - Numeric_bit
  - Numeric_std
- زیرمجموعهٔ قابل سنتز: استاندارد 1076.6
- VHDL-AMS 1076.1

# VHDL - Overview

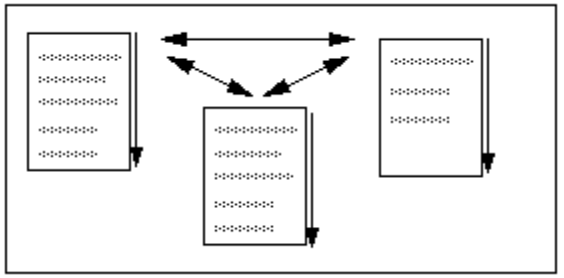
- در حال حاضر برای VHDL-AMS فقط شبیه سازی امکان پذیر است چون سنتز آنالوگ بسیار پیچیده است.
- شبیه سازی Mixed Signal هم مسایل همگام سازی شبیه سازهای دیجیتال و آنالوگ و الگوریتمهای حل معادلات دیفرانسیل غیرخطی را دارد.

# VHDL-Application Field

- **Hardware design**
  - **ASIC: technology mapping**
    - توصیف به gate-level netlist تبدیل می شود و اجزا از کتابخانهٔ ASIC انتخاب می شوند.
  - **FPGA, CPLD: LUT/PAL mapping**
  - **SPLD: smaller structures, hardly any use of VHDL**

# Concepts of VHDL

- **Execution of Statements:**
  - **Sequential**
  - **Concurrent**



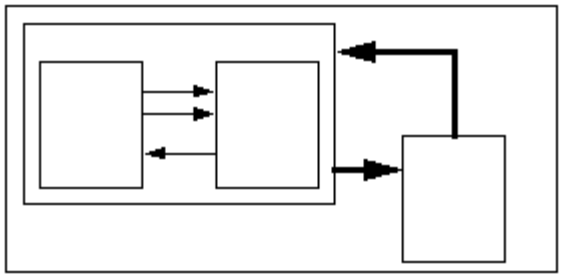
- **Methodologies:**
  - **Abstraction**
  - **Modularity**
  - **Hierarchy**

# Concepts of VHDL

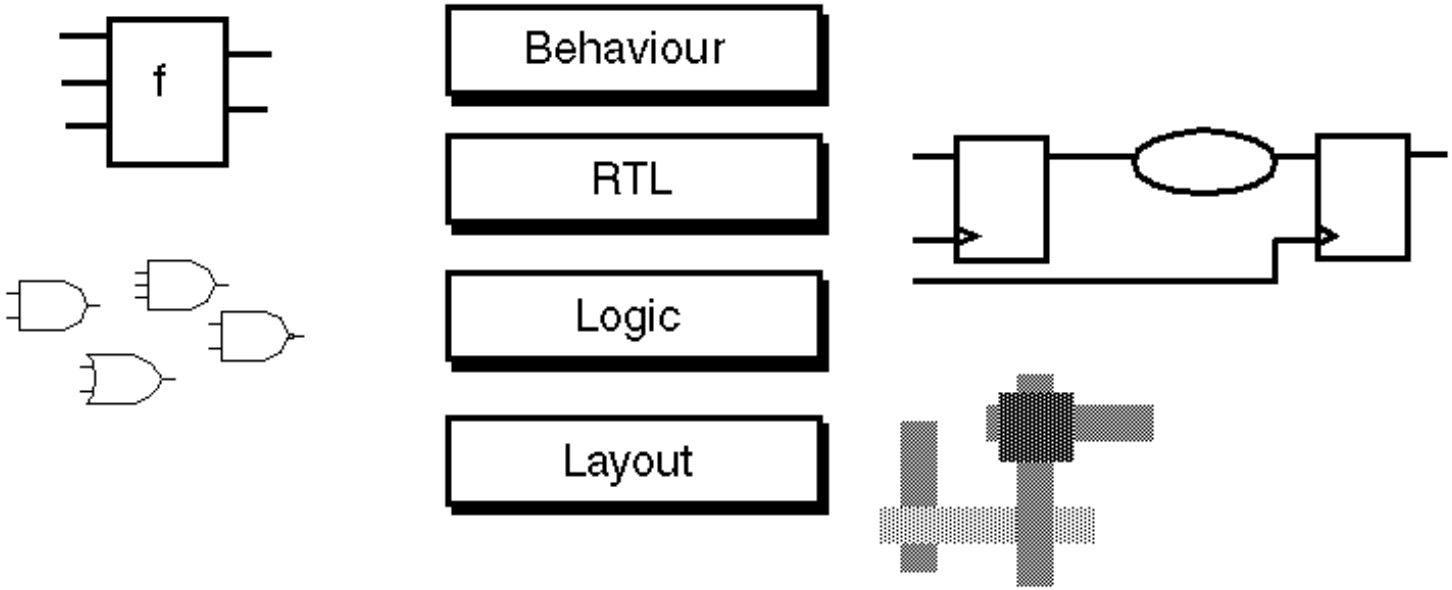
- **Abstraction: طرح را می توان در سطوح مختلفی از جزئیات توصیف کرد:**
  - **برای مدلسازی، سطوح بالا کافی است.**
  - **برای سنتز، ممکن است جزئیات بیشتری لازم باشد.**

# Concepts of VHDL

- **Modularity**: می توان بلوک بزرگ پیچیده را به بلوکهای کوچکتر تقسیم کرد و برای هر بخش یک مدل نوشت.
- **Hierarchy**: تشکیل یک درخت سلسله مراتبی
  - هر کدام از نودها ممکن است در سطح متفاوتی از **abstraction** توصیف شده باشد.



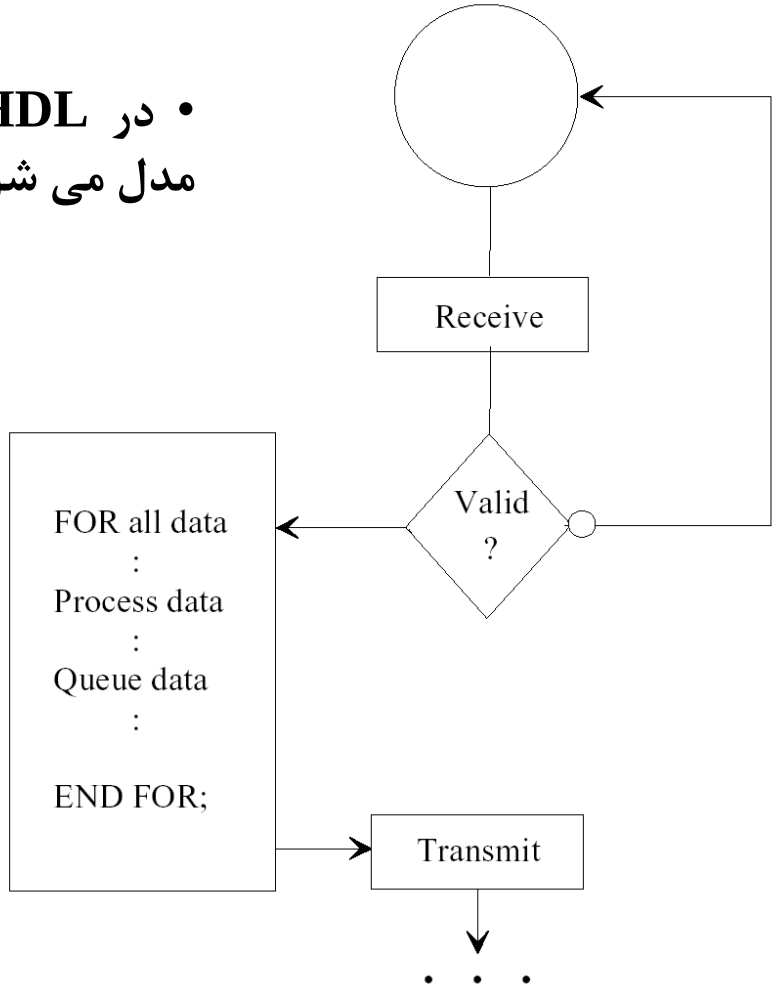
# Abstraction Levels in IC Design




- VHDL برای سطح layout مناسب نیست.

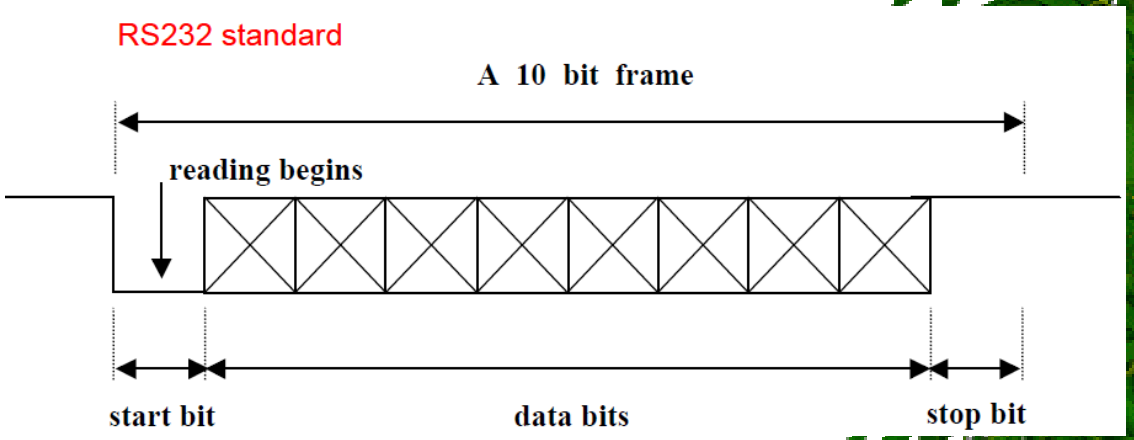
# Behavioral Description in VHDL

- در VHDL رفتار عملیاتی با process مدل می شود.



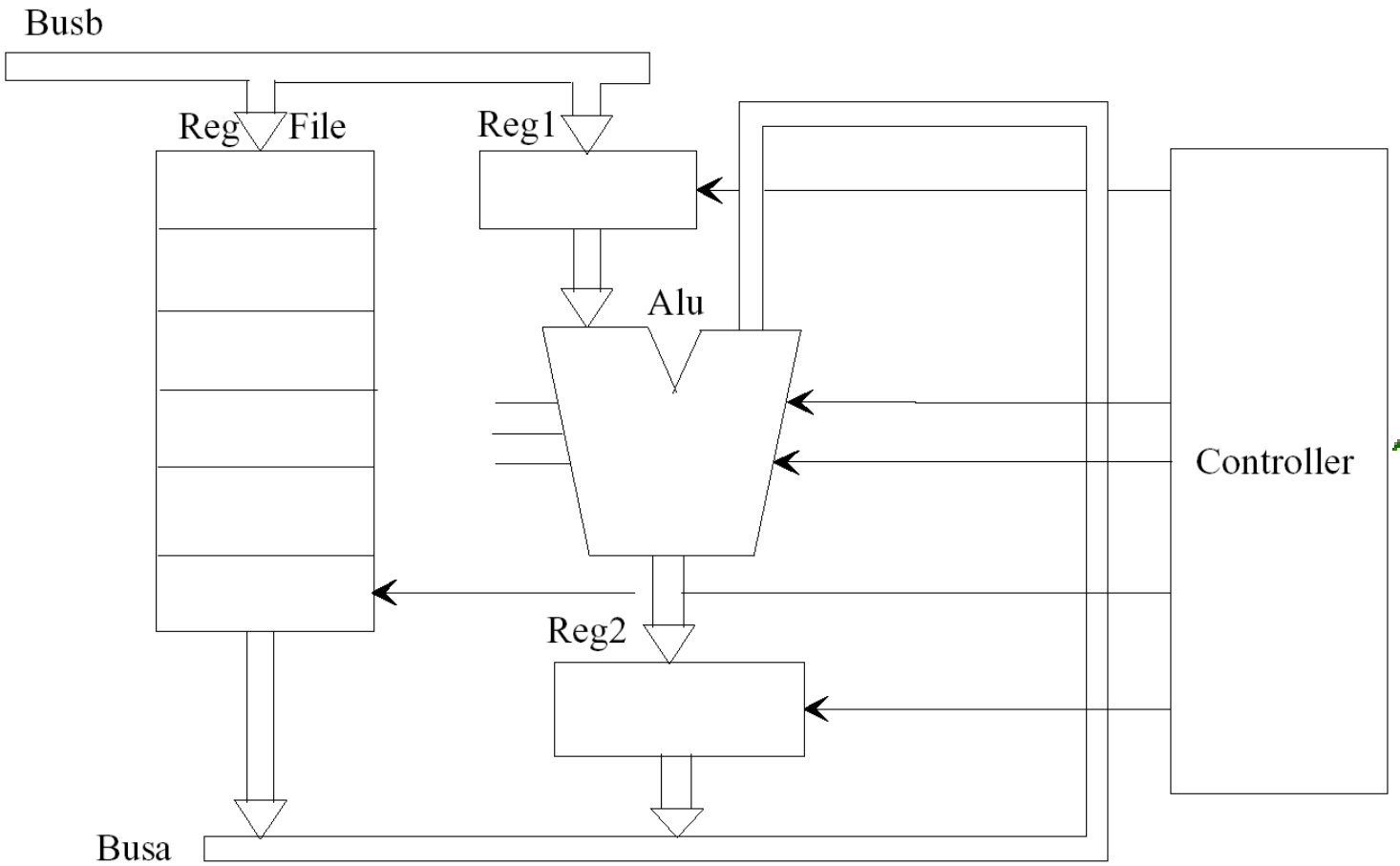
# Behavioral Description

```
collect : PROCESS
BEGIN
   WAIT UNTIL serial = '0';
   WAIT FOR half_bit;
   FOR count IN 0 TO 7 LOOP
      WAIT FOR full_bit;
      buff (count) := serial;
   END LOOP;
   WAIT FOR full_bit;
   IF serial = '0' THEN
      frame_error <= '1';
      WAIT UNTIL serial = '1';
   ELSE
      frame_error <= '0';
      dataready <= '1';
      parallel_out <= buff;
      WAIT UNTIL received = '1';
      WAIT UNTIL received = '0';
      dataready <= '0';
   END IF;
END PROCESS collect;
```




```
constant half_bit : time := 50 ns;
constant full_bit : time := 100 ns;
```

# Dataflow (RTL) Description

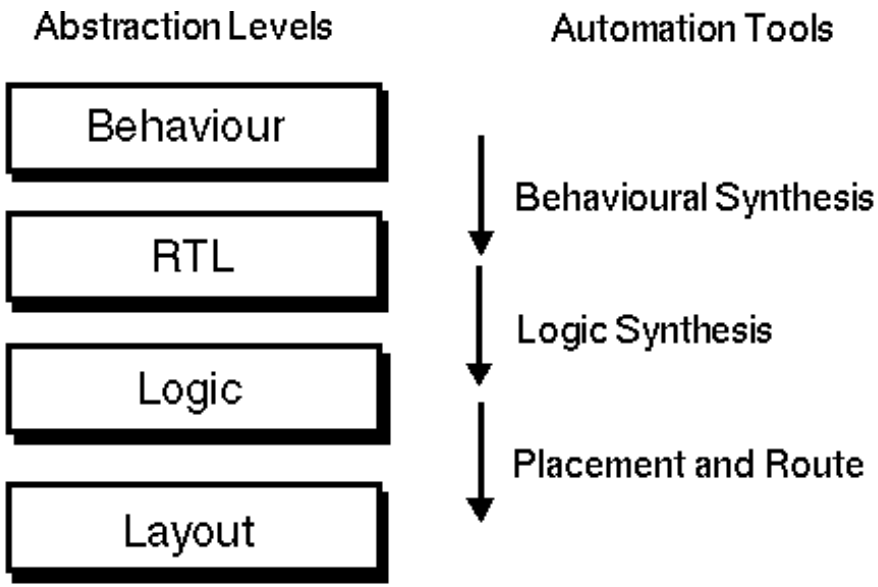


# Dataflow (RTL) Description

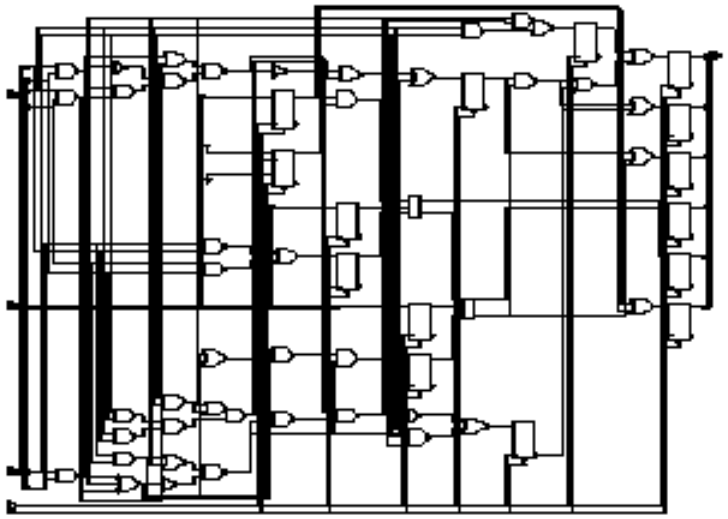
```
architecture RTL of MOORE_TEST is
    signal STATE,NEXTSTATE : STATE_TYPE ;
begin
    REG: process (CLK, RESET) begin
        if RESET='1' then    STATE <= START ;
        elsif CLK'event and CLK='1' then
            STATE <= NEXTSTATE ;
        end if ;
   end process REG ;


 OUTPUT_REG: process(CLK)
   begin
        if CLK'event and CLK='1' then
         Y <= S1 + S2;
         Z <= Z_I ;
        end if ;
   end process OUTPUT_REG ;
 end RTL ;
```

# Abstraction Levels and VHDL



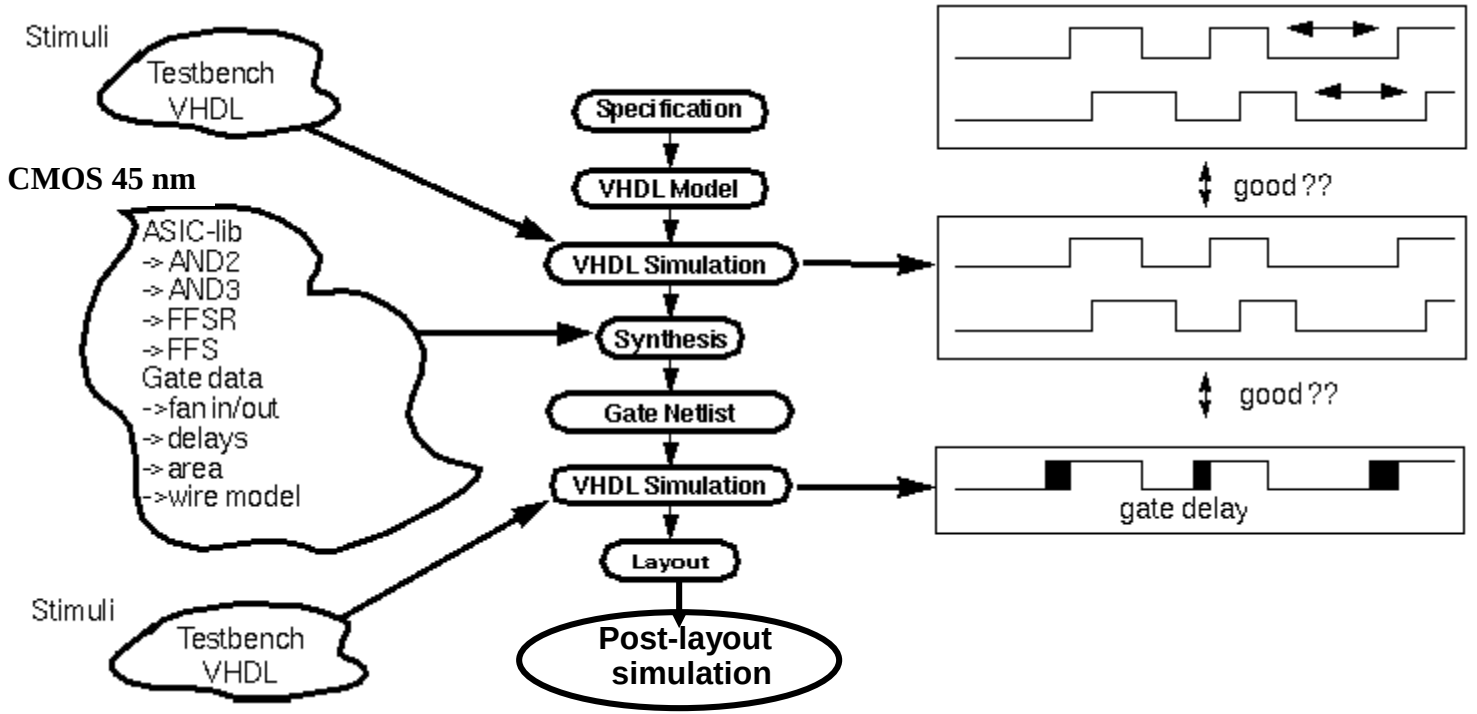
# Structural Description



```
U86 : ND2 port map( A => n192, B => n191, Z => n188);
U87 : ND2 port map( A => I3_2, B => I2_0, Z => n175);
U88 : ND2 port map( A => I2_2, B => I3_0, Z => n173);
U89 : NR2 port map( A => mul_36_PROD_not_0,
        B => n174, Z => n185);
U90 : EN port map( A => n181, B => n182, Z => n180);
U91 : ND2 port map( A => I3_2, B => I2_1, Z => n181);
U92 : ND2 port map( A => I2_2, B => I3_1, Z => n182);
U93 : IVP port map( A => n180, Z => n192);
U94 : AO6 port map( A => n173, B => n174, C => n175,
        Z => n172);
U95 : NR2 port map( A => n174, B => n173, Z => n176);
U96 : ND2 port map( A => I3_1, B => I2_1, Z => n174);
U97 : EN port map( A => n183, B => n178,
        Z => product64_4);
U98 : ND3 port map( A => I2_2, B => I3_2, C => n174,
        Z => n183);
```

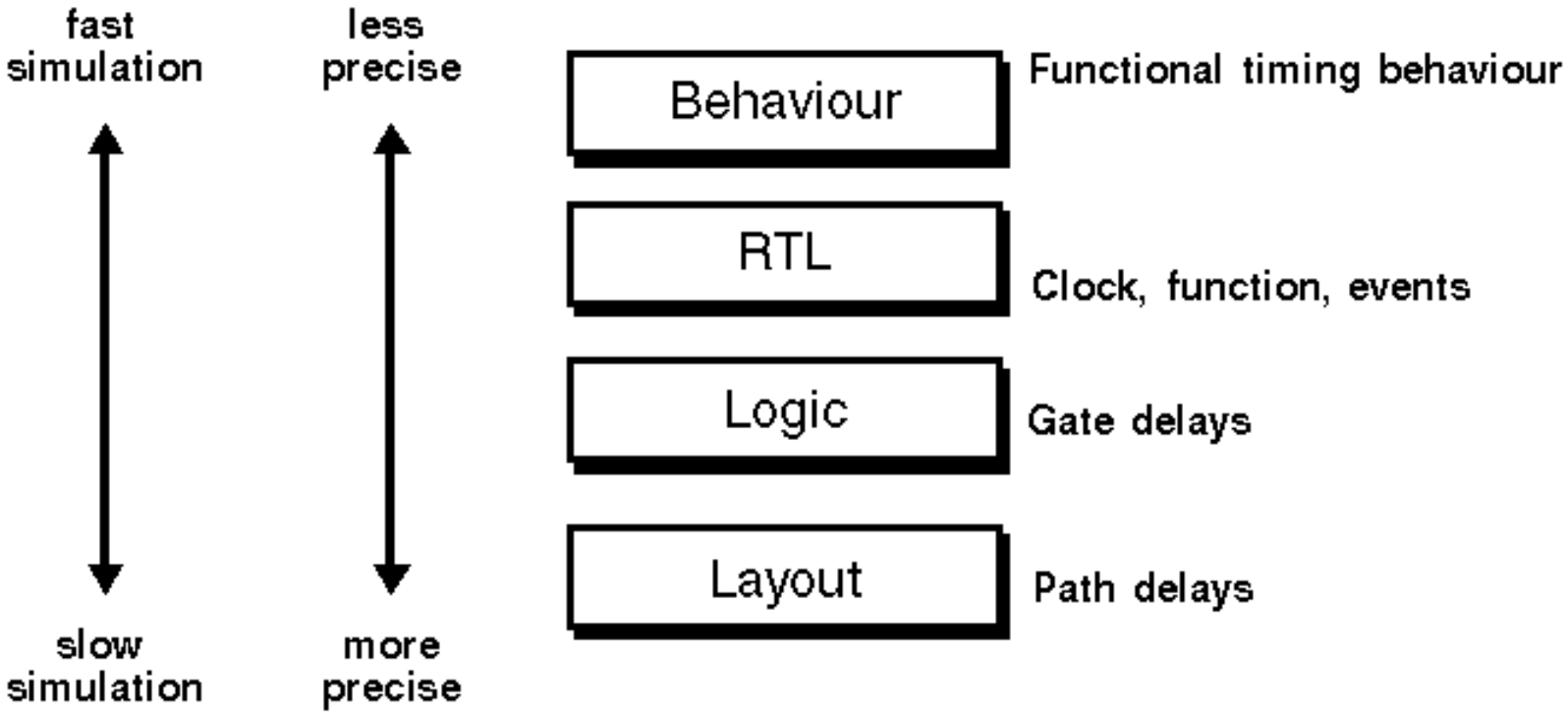
- **اجزا ممکن است گیت پیچیده باشند.**

# ASIC Development

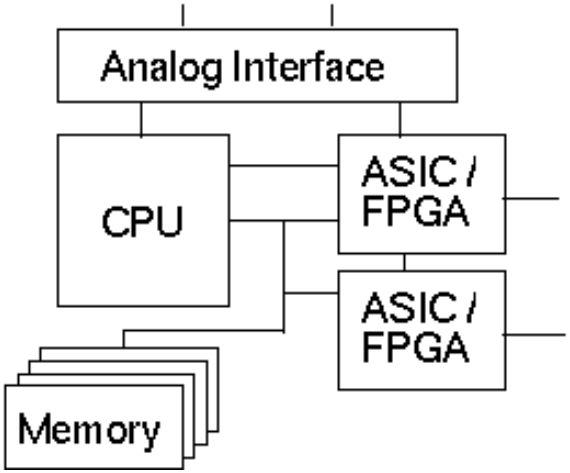


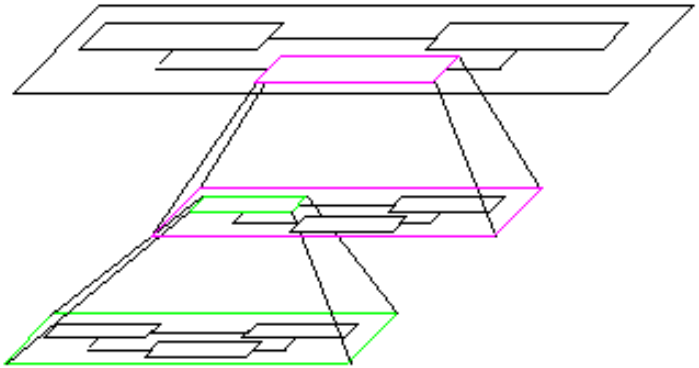

- بعد از شبیه سازی **post-synthesis** می توان ماکزیمم فرکانس کلاک را تخمین زد (بر اساس مسیر بحرانی موجود)
- بعد از شبیه سازی **post-layout** می توان ماکزیمم فرکانس کلاک را به دست آورد

# Information Content in Abstraction Levels



# Modularity and Hierarchy



- **Partitioning in several partial designs**
  - **Restrict complexity**
  - **Enable teamwork**
  - **Study of alternative implementations**

• گاهی اوقات simulation modelهایی از اجزا یا تراشه های استاندارد وجود دارند که می توان به طرح خود متصل کرد و نتایج شبیه سازی را در محیط واقعی مشاهده کرد.

# Design Tools

## Simulation

# Tool Set

- **Synthesis**
- **Place & Route**
- **Simulation Tools**
- **Timing Analysis**
- **Power Analysis**
- **Verification**

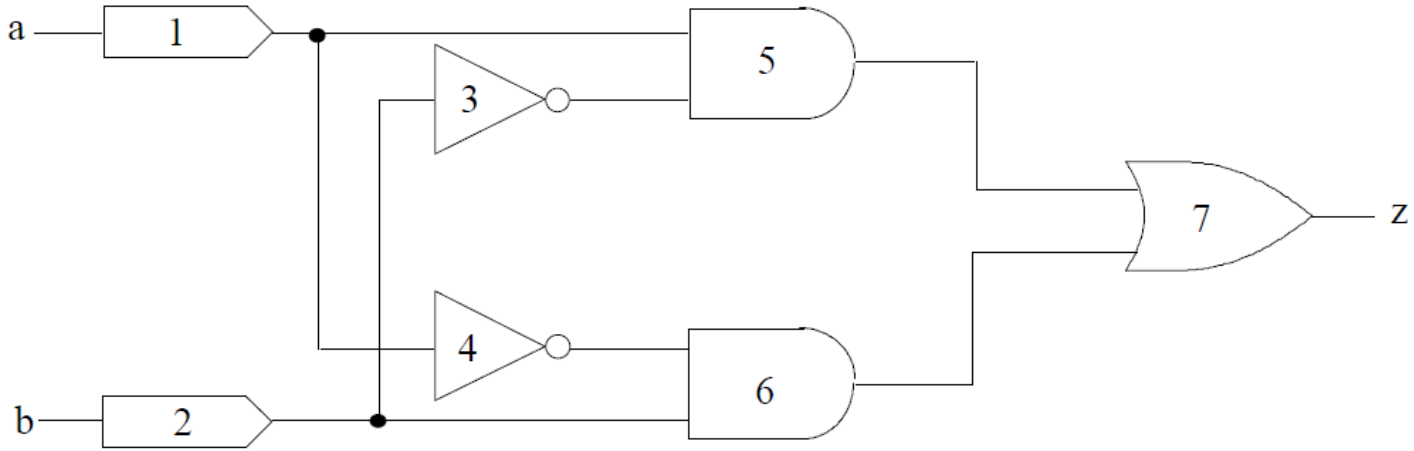
# Simulation Tools

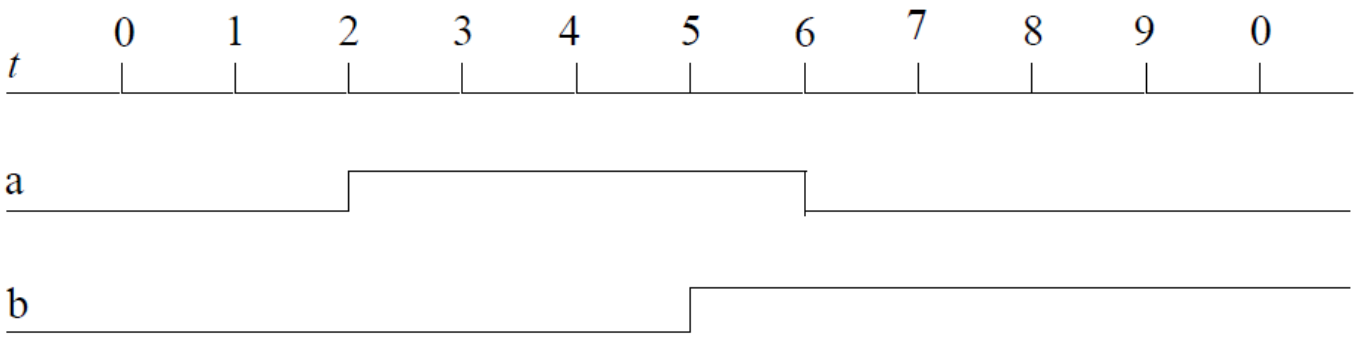
- **Simulation Tools**
  - Oblivious simulation
  - Event-driven logic simulation
  - Mixed-language simulation
  - Cycle-based simulation
  - Post-layout simulation

# Simulation Tools

- Types of Logic Simulators:
  - Oblivious:
    - Simple but inefficient
  - Event driven:
    - See the world as a series of discrete events
  - Cycle-based
    - Rough simulation for synchronous sequential designs

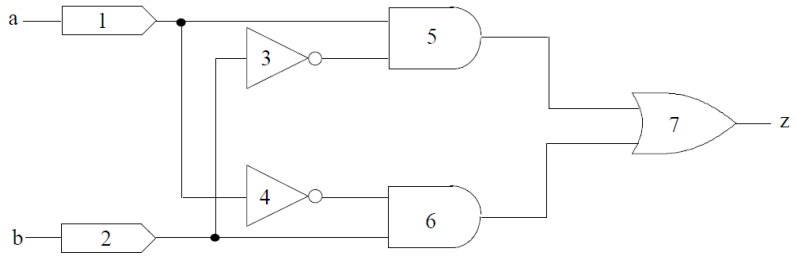
# Simulation Concept



# Oblivious Simualtor

- **Starts by a table and initial values**
- **Evaluates periodically (even when no change in inputs)**
  - **Repeats until nothing is changed in a pass.**

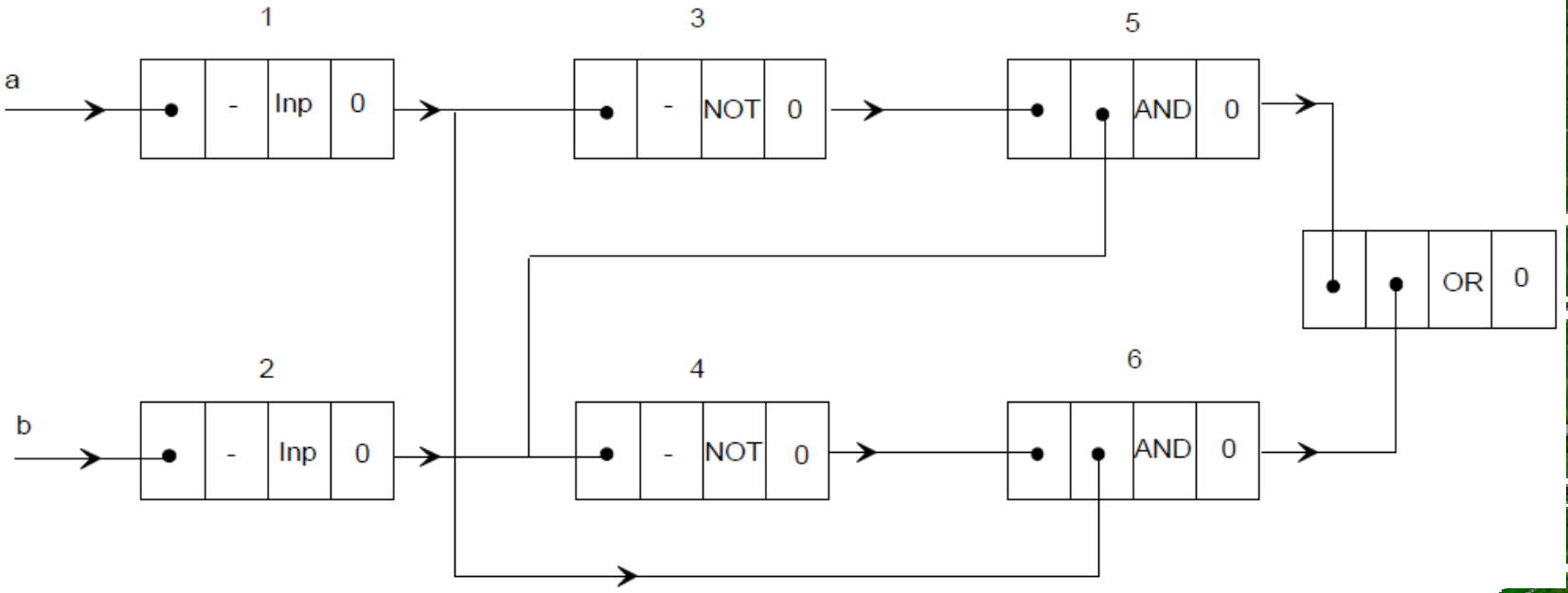



| GATE | FUNCTION | INPUT 1 | INPUT 2 | VALUE |
|---|---|---|---|---|
| 1 | Input | a | -- | 0 |
| 2 | Input | b | -- | 0 |
| 3 | NOT | 2 | -- | 1 |
| 4 | NOT | 1 | -- | 1 |
| 5 | AND | 1 | 3 | 0 |
| 6 | AND | 4 | 2 | 0 |
| 7 | OR | 5 | 6 | 0 |

# Event-Driven Simulator

| In1 | In2 | Fnc | Out |
|---|---|---|---|

- **Simulates when an event (change) at input(s) of a component**
- **Only part of the design is evaluated**

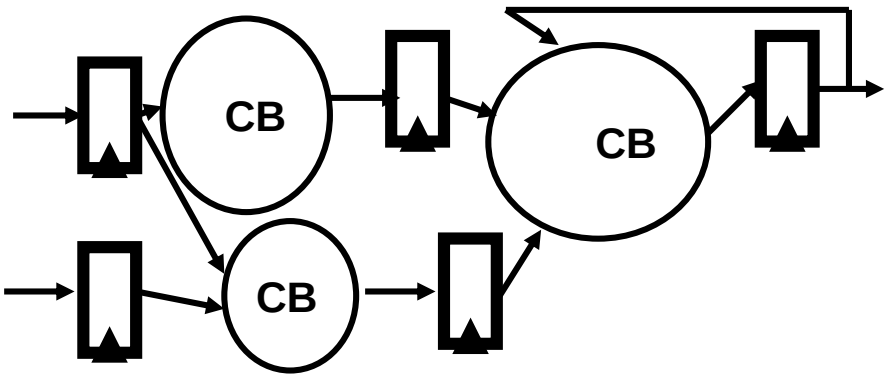


# Event-Driven Simulator

- **When a gate input changes, it places a transaction (value, delay) on the driver of the delayed signal (e.g. gate output)**
  - **It will expire after delay**
  - **→ Changes its value**
- **Repeats for other gates with changed inputs**
- **Concurrency is handled.**
- **Mostly used in industry**

# Cycle-based Simulation

- RTL:
  - At the beginning of each cycle, propagate values from primary inputs and from register outputs into *combinational blocks* (CBs).
  - Compute the stable outputs of CBs, and store them in register inputs (they'll move to outputs in the next cycle).



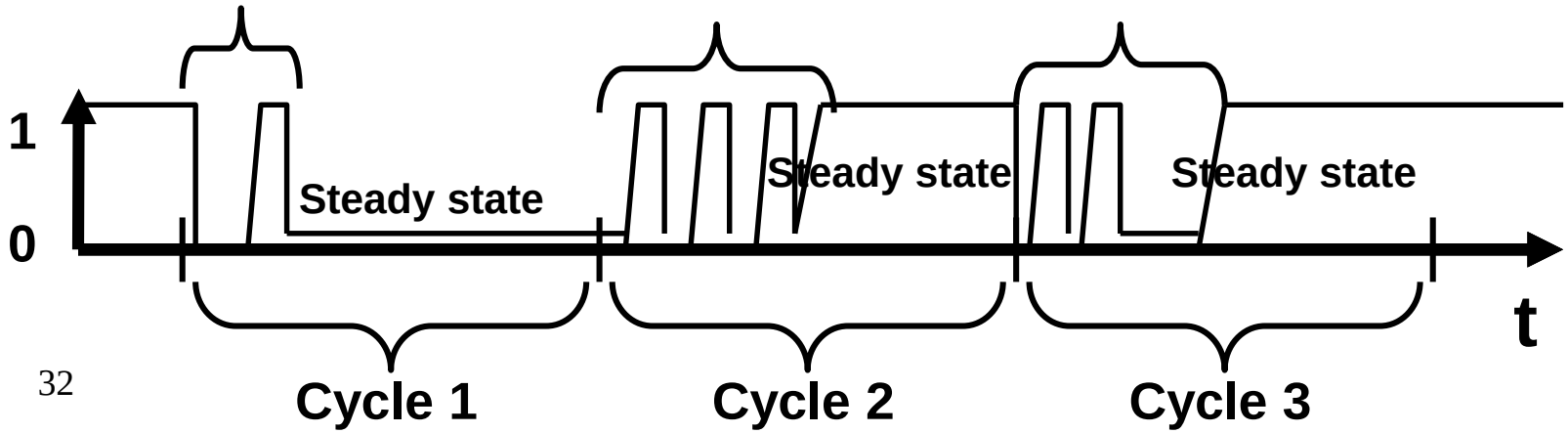

**RTL model**

# Zero-delay cycle-based simulation

- Functionality and timing can be verified separately
- Assumption:
  - Within each cycle, all signal changes occur in zero-delay

- Only evaluate on the clock edge
  - First: evaluate all combinational logic
  - Next: latch values into state registers
  - Repeat on next clock edge

Assume ‘zero-delay’ (don’t care about the transient delays)

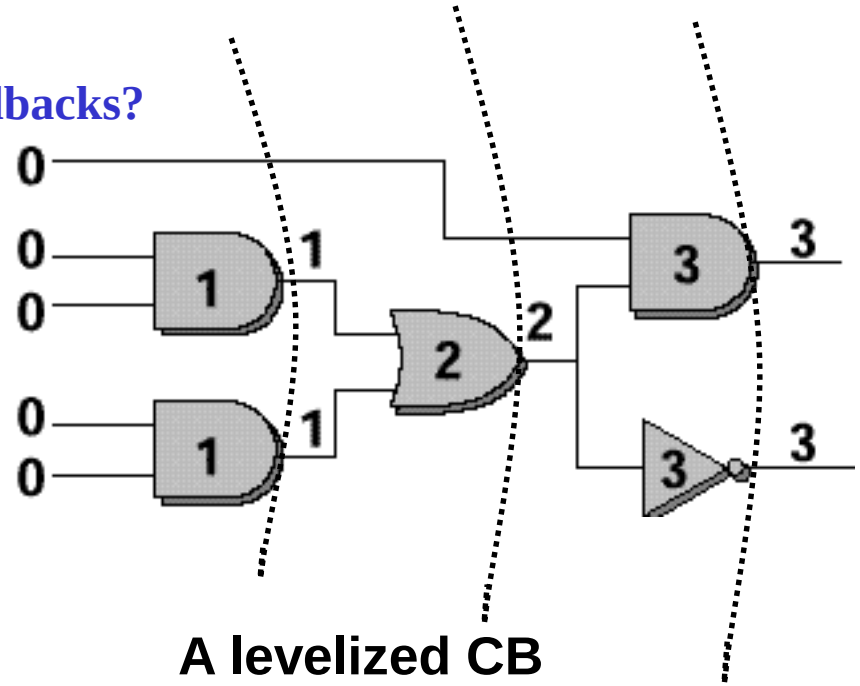


# Levelizing the gates

- Within each Combinational Block, gates can be *levelized* (sorted in *topological order)* for evaluation.
- Going by level number, every gate will be evaluated after its inputs have been computed.

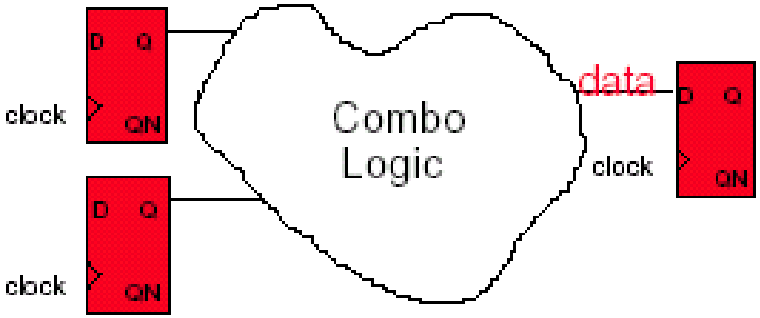
- **So: we order the gates by the flow of data, and we don't care about the real delays → assume "0 delay".**

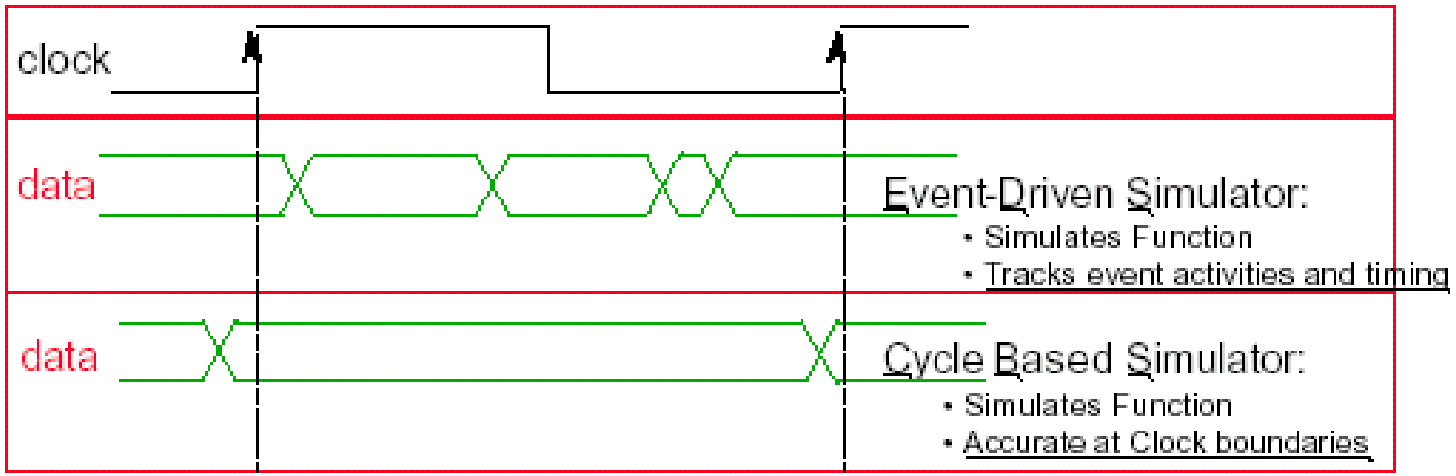
- **Combinational Feedbacks?**




**A levelized CB**

# Event vs. Cycle-Based Simulation



# Cycle-Based vs. Event-Driven

- **Cycle-based:**
  - Only boundary nodes
  - No delay information
- **Event-driven:**
  - Each internal node
  - Need scheduling and functions may be evaluated multiple times

- Cycle-based does not detect glitches and setup/hold time violations, while event-driven does

# Simulation: Performance vs. Abstraction

Cycle-based Simulator

Event-driven Simulator

SPICE

Abstraction

.001x

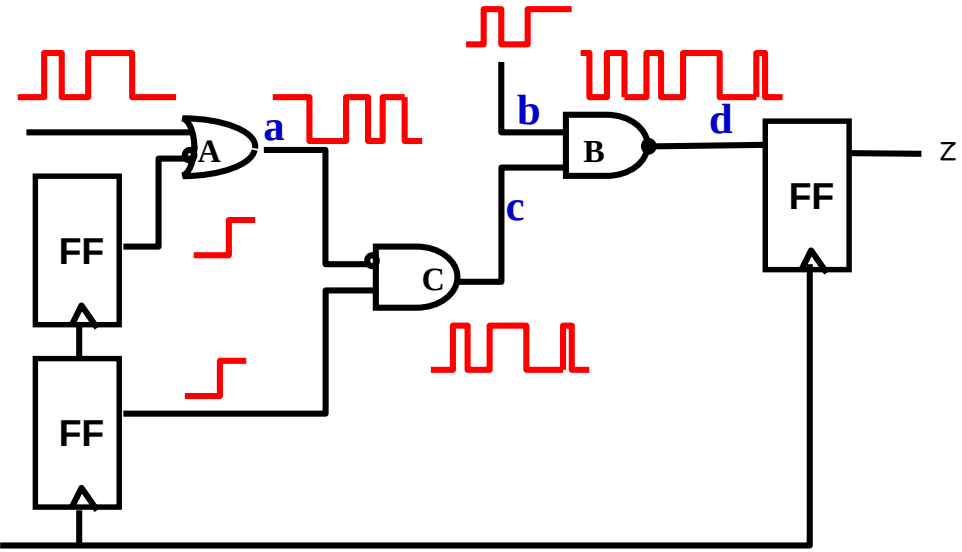
1x

10x

Performance and Capacity

مرتضی صاحب الزمانی

# Why is it faster?




- Node **a** has 5 transitions and **b** has 3 transitions.
- **Event-driven simulator** reevaluates when inputs change
- → Gate A simulates 5 times,

  Gate B: 9 times,

  Gate C: 6 times

  Total 20 times.

  But for only these 3 gates

- **Cycle-based simulator** simulates the entire circuit only once (at the end of current cycle)
- → All Gates in the circuit are simulated regardless whether their inputs have been changed.

- **Conclusion:**
  - Event-driven is faster if few transitions (typically less than 5% active nodes).
- In practice, experiments confirm that a cycle-based simulator is almost always 5-10 times faster (and uses less memory)