

Unit 6. Writing Testbenches

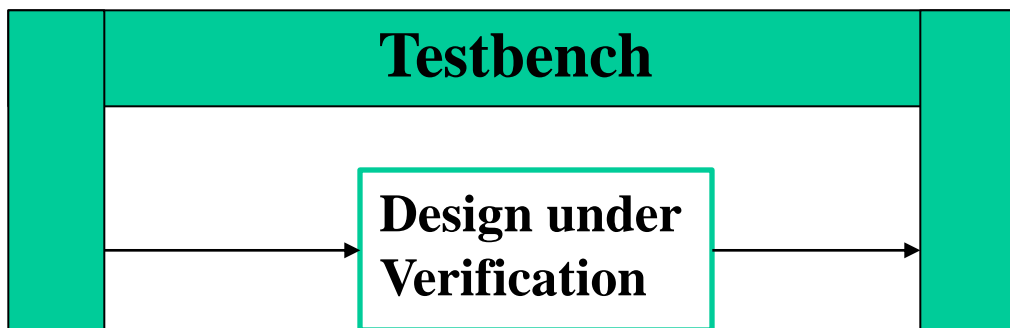
- 6.1 The Importance of Verification
- 6.2 Verification Tools
 - 6.2.1 Logic Simulation Overview
 - 6.2.2 True Value Simulation
 - 6.2.3 Event-Driven Simulation
- 6.3 Behavioral HDL
- 6.4 Writing Testbenches

6.1 The Importance of Verification

- Verification is a process used to demonstrate the functional correctness of a design.
- Verification is not a testbench, nor is it a series of testbenches.

What is a testbench?

- The term “testbench”, in VHDL and Verilog, usually refers to the code used to create a pre-determined input sequence to a design, then optionally observe the response.



6.1 The Importance of Verification (cont'd)

- Verification consumes about 70% of the design effort.
- Verification is on critical path.
- Verification time can be reduced through parallelism. To parallelize the verification effort, it is necessary to be able to write-and debug- testbenches in parallel with each other as well as in parallel with the implementation of the design.
- Using abstraction reduces control over low-level details. Verification time can be reduced through automation.

6.2: Verification Tools

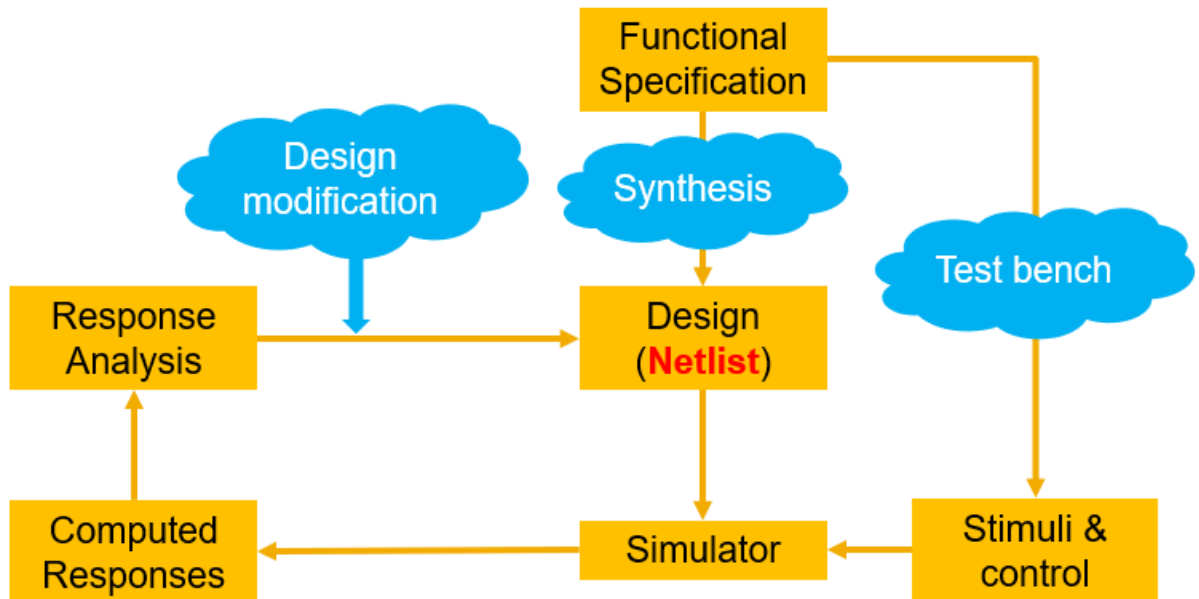
Different kinds of verification tools can be used to verify different aspect of the design and identify different types of errors.

- Linting tools
- Simulators
- Waveform viewers
- Code coverage analysis
- Revision control and issue tracking

Simulator

- A tool for design verification testing;
- Ascertain a design perform its specified behavior (function and timing)

Logic Simulation Overview



6.2.1 Logic Simulation Overview

Netlist: a format (or language) that describes a design as an interconnection of modules. Netlist may use hierarchy.

Modules, blocks or components

described by:

- I/O functions
- Delays associated with I/O signals
- Examples: FET, resistors, capacitors, gates, adder etc.

Interconnectors represent:

- Ideal signal carriers or ideal electrical conductors

6.2.2 True Value Simulation

Determine the output for certain inputs

Find design errors, timing problems, etc.

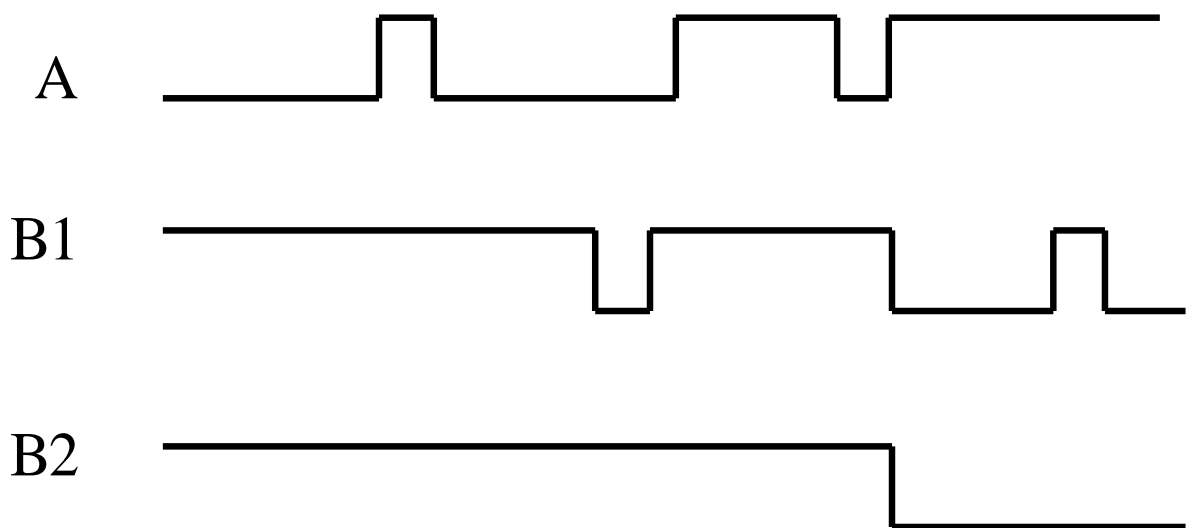
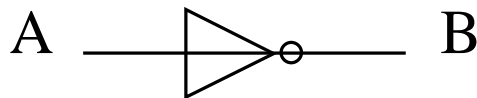
Limitations:

- The set of design errors is not enumerable
- No formal procedure to generate tests
- A system passes the test is shown correct only with the applied test cases
- Simplified circuit model
- A circuit passes the simulation test may not work when being wired up

6.2.2 True Value Simulation (cont'd)

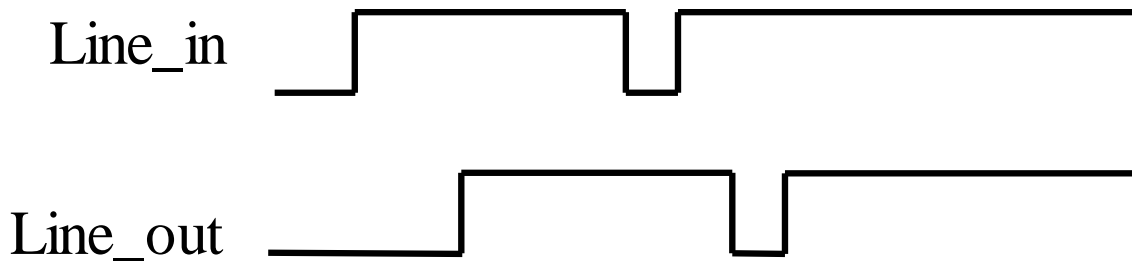
Timing :delay models in VHDL

- Transport delays
- Initial delays



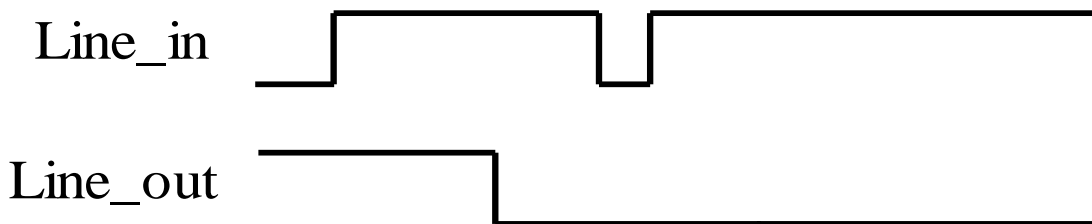
Transport delay:

- Model transmission lines.
- `Line_out <= transport line_in after 3 ns;`
- Remarks: Output is assigned with specified delay independently from the width of the pulse in waveform.



Inertial delay (default):

- Model the devices. Default delay model implemented in VHDL.
- `Line_out <= inertial not line_in after 3 ns;`
- `Line_out <= not line_in after 3 ns;`
- `Line_out <= reject 500ps inertial not line_in after 3 ns;`
- Remarks: Impulses shorter than the specified switching time are not transmitted.



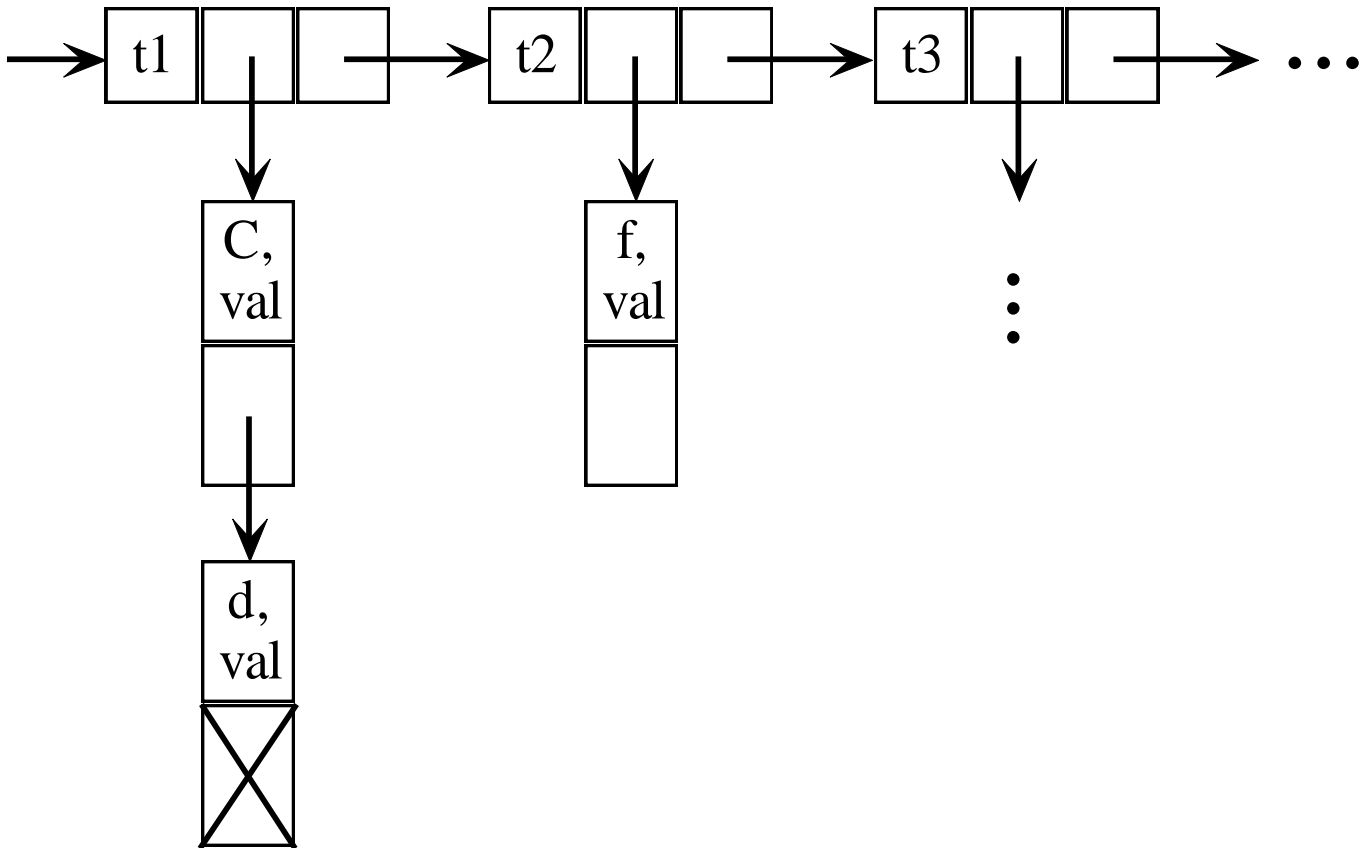
6.2.3 Event-driven Simulation

- General concept behind most simulation algorithms
- An event represents a change in the value of a signal line at some simulated time t
- If the value of a signal line x changes, then all gates having x as input are activated
- If the activated gates change their output values, new events are generated.

6.2.2 Event_driven Simulation (cont'd)

- An event list can be organized as a linked list stored in increasing time order

event
list



6.2.2 Event_driven Simulation (cont'd)

Two-pass algorithm for gate-level event-driven simulation

While(event_list is not empty) do begin

For (every event(x,val) at time t) do begin

 Get the event (x,val)

If(current_val_of_x <>val) then begin

 Current_val_of_x=val

For (each gate J on fanout list of x) do begin

 Change the value of fanout line going into gate J

 Add gate J to activated_gates

End

End

End

For (each gate J of activated_gates) do begin

 Compute output value of gate J

If(output_value_gate_j <> last_scheduled_value_gate_j) then begin

 Schedule (output_line_number_gate_J,output_value_gate_J)
 for time t+delay of gate_J

 Last_scheduled_value_gate_J=output_value_gate_J

End

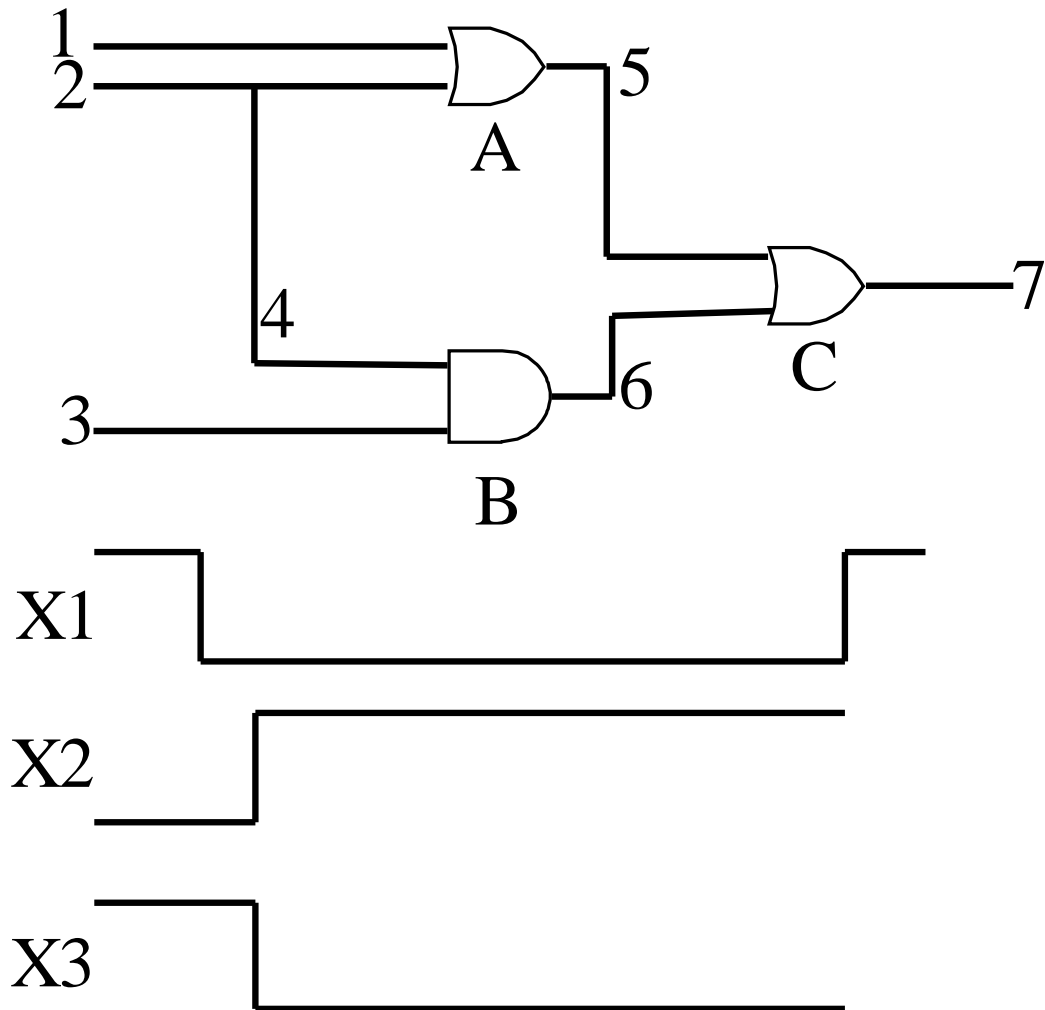
End

End

6.2.2 Event_driven Simulation (cont'd)

- Pass1
- Get the entries from the event list associated with the current time t
- Determine the activated gates
- This is to avoid multiple evaluations of a gate that is activated by more than one event
- Pass2
- Compute the new output values of the activated gates
- Schedule their computed value in the event list
- The algorithm keeps track of the last-scheduled value of a gate so as to schedule only “true” events

6.2.2 Event_driven Simulation (cont'd): an Example



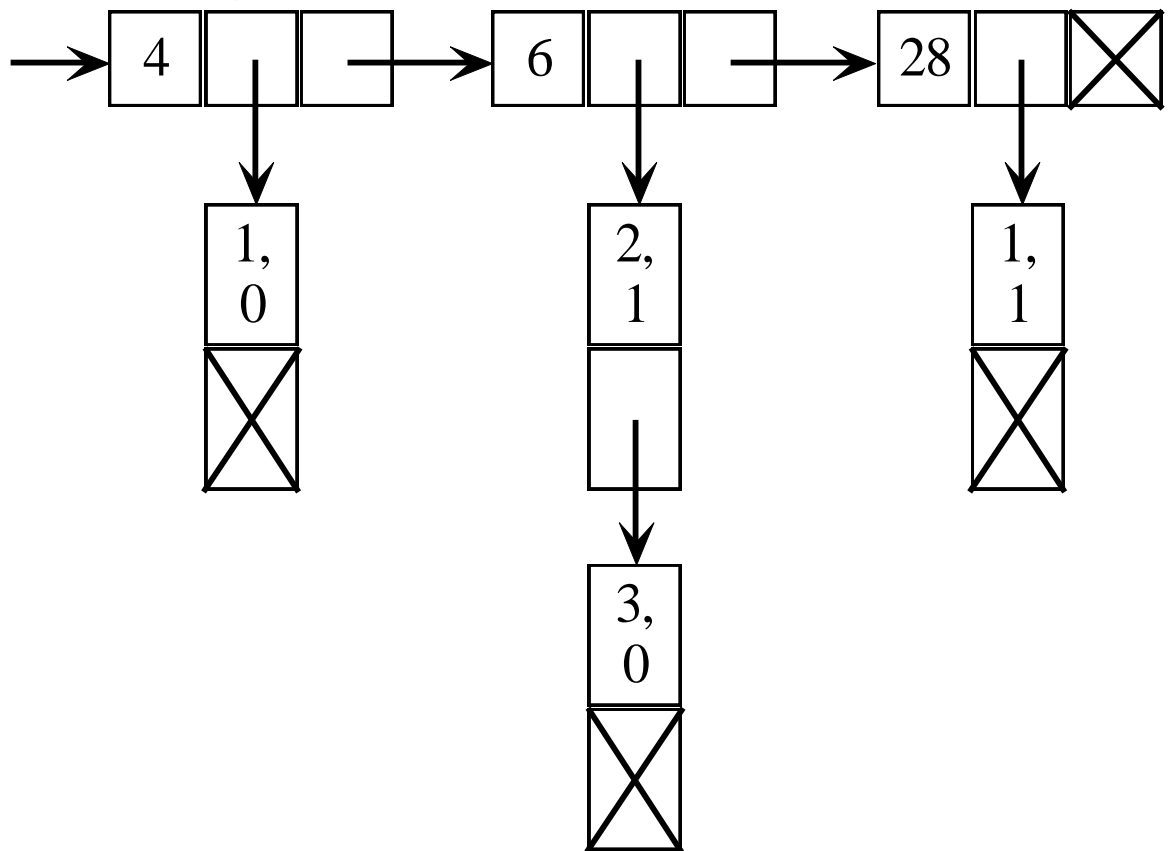
Gate	A	B	C
Propagation delay	10 ns	6 ns	10 ns

6.2.2 Event_driven Simulation (cont'd): an Example

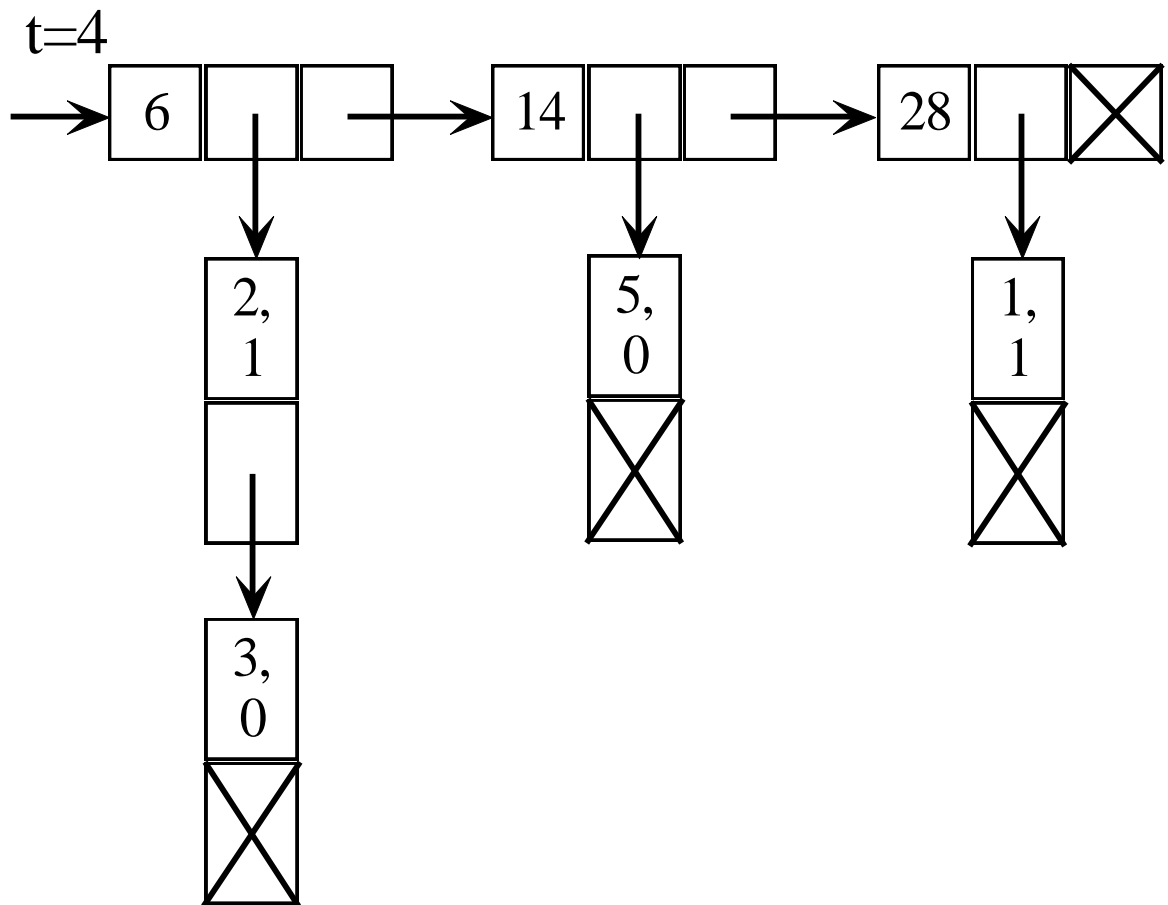
	Phase 1							phase2			
	Line values							Gates affected			Scheduled events
Time	1	2	3	4	5	6	7	A	B	C	
Init	1	0	1	0	1	0	1				
4	0							X			(5,0) at 4+10
6		1	0	1				X	X		(5,1) at 6+10
14					0					X	(7,0) at 14+10
16					1					X	(7,1) at 16+10
24							0				
26							1				
28	1							X			

Event_driven Simulation (cont'd): an Example

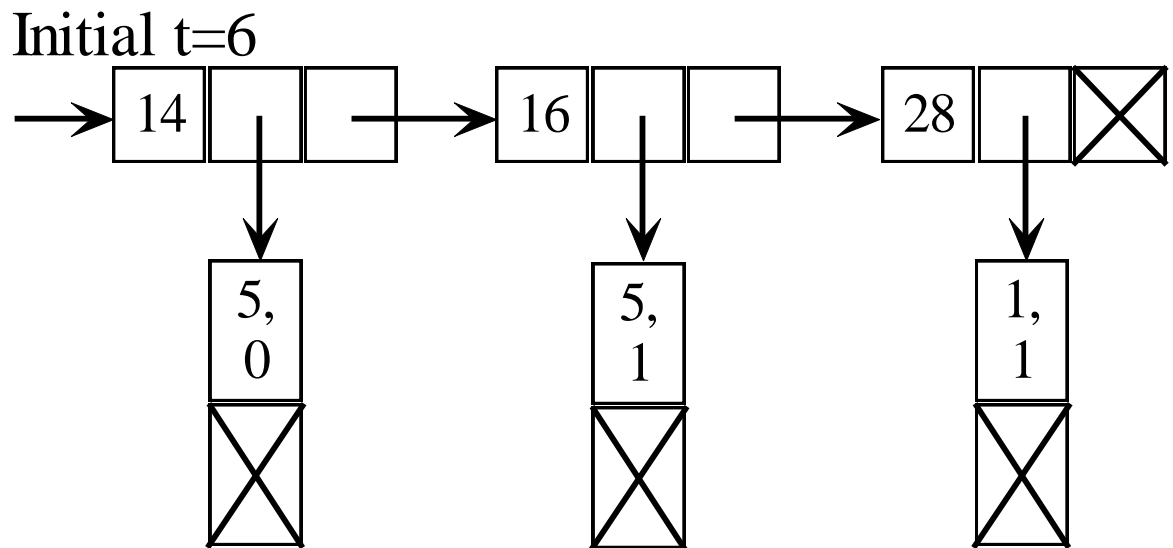
Initial t=0



Event_driven Simulation (cont'd): an Example

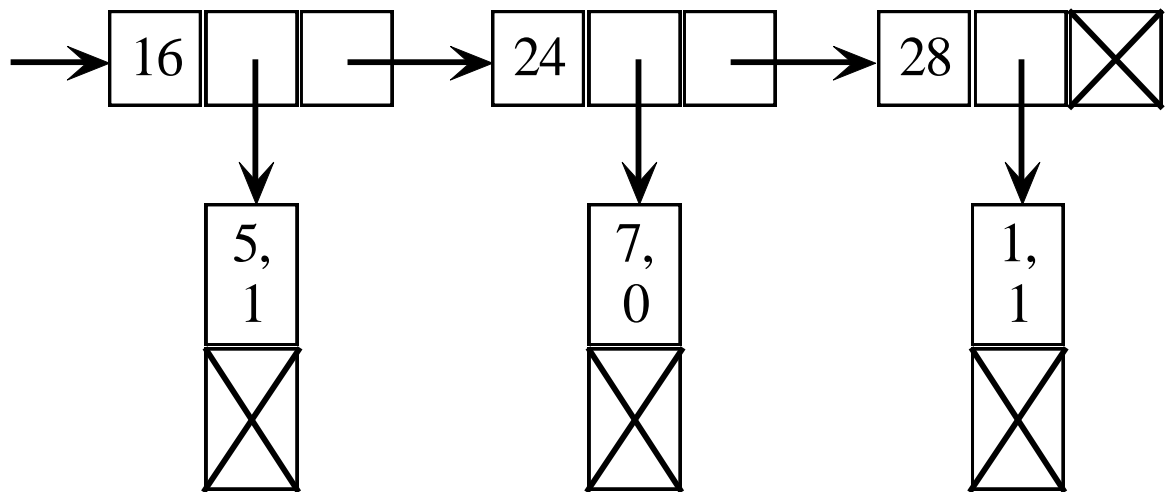


Event_driven Simulation (cont'd): an Example

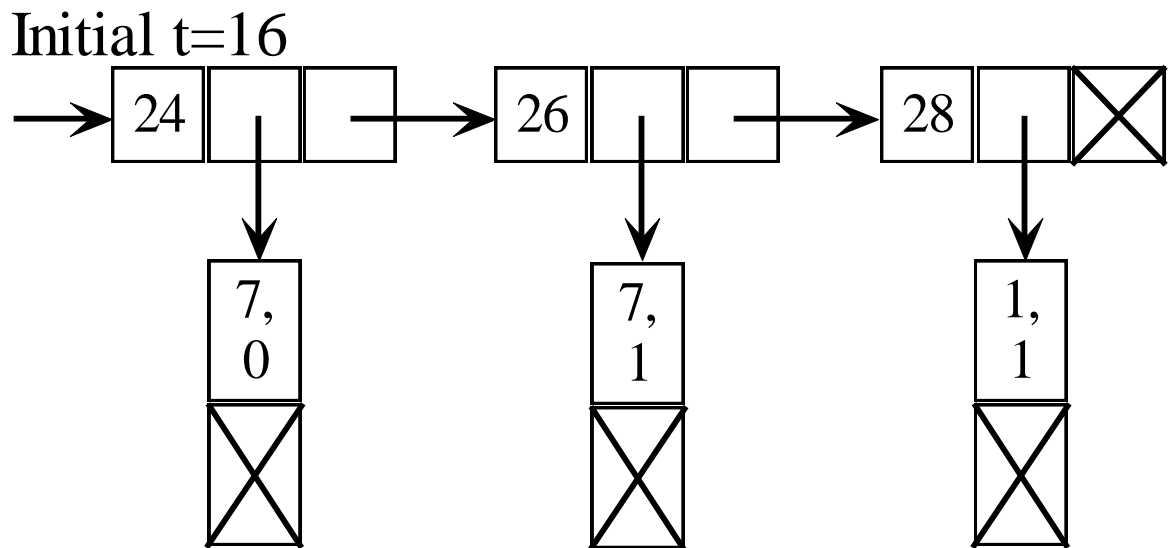


Event_driven Simulation (cont'd): an Example

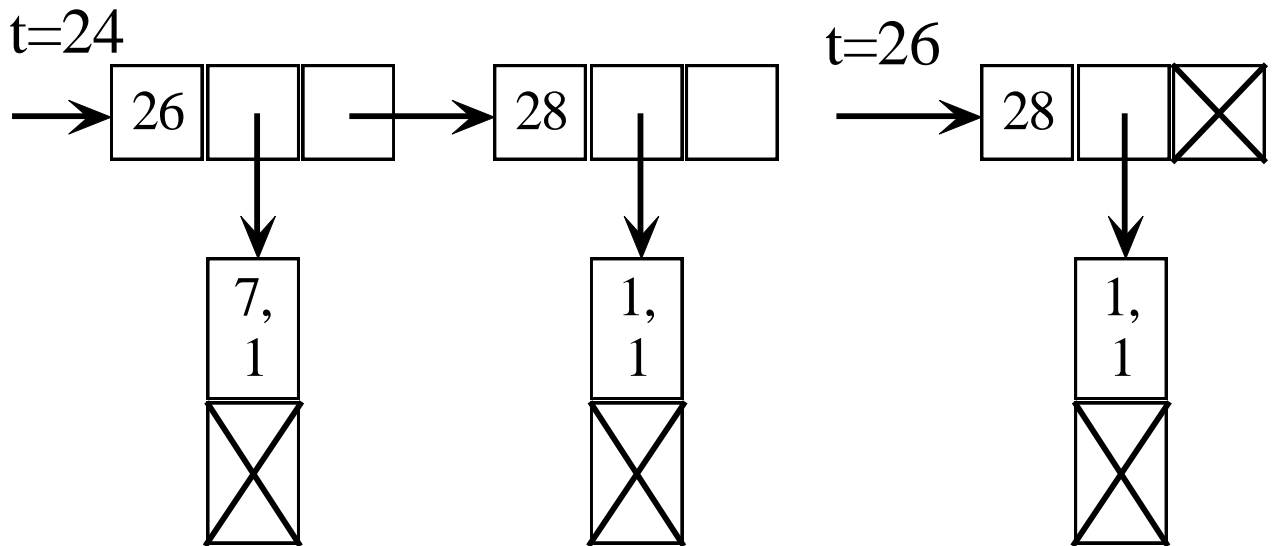
Initial t=14



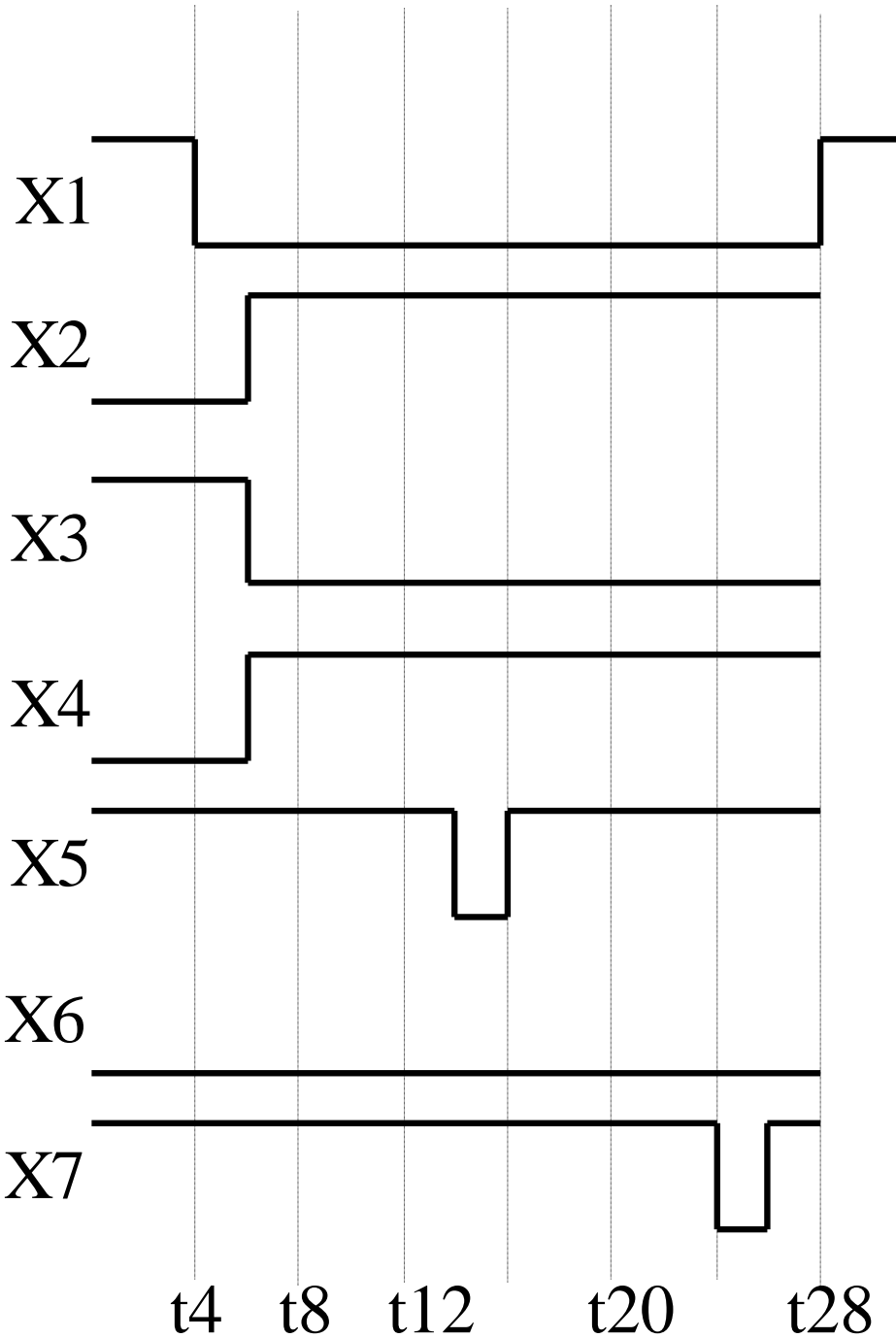
Event_driven Simulation (cont'd): an Example



Event_driven Simulation (cont'd): an Example



Event_driven Simulation (cont'd): Timing Diagram



6.3: Behavioral HDL

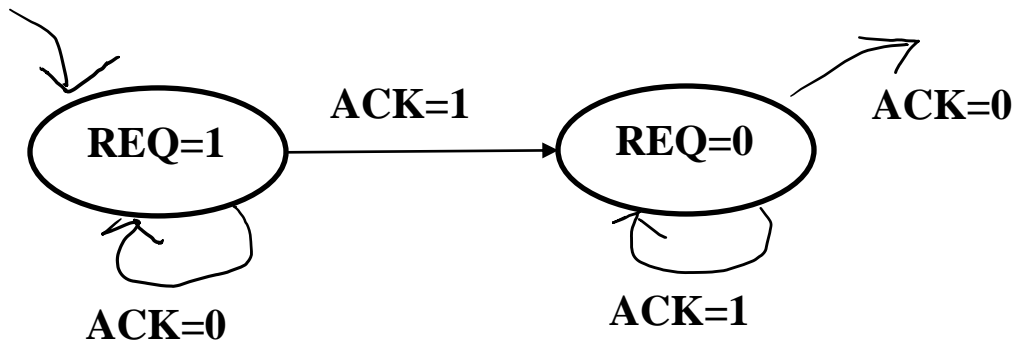
RTL coding guidelines help designer obtain efficient implementations: low area, high speed, or low power.

To avoid undesirable hardware structures:

- To avoid latches, set all outputs of combinational blocks to default values at the beginning of the block;
- To avoid tristate buffers, do not assign the value 'Z'.

To maintain simulation behavior:

- All inputs must be listed in the sensitivity list of a combinational block;
- The clock and asynchronous reset must be in the sensitivity list of a sequential block.



Example: State diagram for handshaking protocol

Behavioral VHDL:

```
process
begin
...
    REQ <= '1';
    Wait until ACK = '1';
    REQ <= '0';
    Wait until ACK = '0';
...
end process;
```

Synthesizable VHDL code

Type state_typ is (... , make_req, release,...);

Signal state, next_state: state_typ;

COMB: process (state, ACK)

begin

 next_state <= state;

case state **is**

 ...

when make_req =>

 Req <= '1';

if ACK = '1' **then**

 Next_state <= release;

end if;

when release =>

 Req <= '0';

if ACK = '0' **then**

 Next_sate <=...;

end if;

 ...

end case;

end process COMB;

SEQ: process (CLK)

begin

if clk'event **and** clk='1' **then**

if reset = '1' **then**

 state <= ...;

else

 state <= next_state;

end if;

end if;

end process SEQ;

Connectivity, Time and Concurrency

- Connectivity is the ability of describing a design using simpler blocks then connecting them together.
- Time is the ability to represent how the internal state of a design evolves over time.
- Concurrency is the ability to describe actions that occur at the same time, independently of each other.

6.4 Writing Test-Benches

- Generating a Simple Waveform

```
.....  
constant clk_period : time := 1 ns;  
  
process  
begin  
    clk <= '0';  
    wait for clk_period/2;  
    clk <= '1';  
    wait for clk_period/2;  
end process;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
```

```
ENTITY testbench IS
END testbench;
```

```
ARCHITECTURE Behavior OF testbench IS
```

```
COMPONENT Addern
```

```
PORT ( X, Y : IN STD_LOGIC_VECTOR(15 DOWNT0 0);
      Cin : IN STD_LOGIC;
      S : OUT STD_LOGIC_VECTOR(15 DOWNT0 0);
      Cout : OUT STD_LOGIC );
```

```
END COMPONENT;
```

```
SIGNAL Cin : STD_LOGIC;
```

```
SIGNAL X : STD_LOGIC_VECTOR(15 DOWNT0 0);
```

```
SIGNAL Y : STD_LOGIC_VECTOR(15 DOWNT0 0);
```

```
SIGNAL S : STD_LOGIC_VECTOR(15 DOWNT0 0);
```

```
SIGNAL Cout : STD_LOGIC;
```

```
BEGIN
```

```
vectors: PROCESS
```

```
  BEGIN
```

```
    X <= X"0000"; Y <= X"0000"; Cin <= '0';
```

```
    WAIT FOR 20 ns;
```

```
    Y <= X"000A"; Cin <= '0';
```

```
    WAIT FOR 20 ns;
```

```
    X <= X"000A"; Cin <= '0';
```

```
    WAIT FOR 20 ns;
```

```
    Cin <= '1';
```

```
    WAIT FOR 20 ns;
```

```
    X <= X"FFF0"; Y <= X"000F"; Cin <= '0';
```

```
    WAIT FOR 20 ns;
```

```
    Cin <= '1';
```

```
      WAIT;
```

```
  END PROCESS;
```

```
U1: Addern PORT MAP (X, Y, Cin, S, Cout);
```

```
END;
```

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_signed.all;
```

```
ENTITY testbench IS  
END testbench;
```

```
ARCHITECTURE Behavior OF testbench IS
```

```
COMPONENT Accumulate
```

```
  PORT ( KEY : IN STD_LOGIC_VECTOR(0 DOWNTO 0);  
         SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);  
         CLOCK_50 : IN STD_LOGIC;  
         LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));  
END COMPONENT;
```

```
  SIGNAL CLOCK_50 : STD_LOGIC;  
  SIGNAL KEY : STD_LOGIC_VECTOR(0 DOWNTO 0);  
  SIGNAL SW : STD_LOGIC_VECTOR(9 DOWNTO 0);  
  SIGNAL LEDR : STD_LOGIC_VECTOR(9 DOWNTO 0);
```

```
BEGIN
```

```
  U1: Accumulate PORT MAP (KEY, SW, CLOCK_50, LEDR);
```

```
  clock_process: PROCESS
```

```
    BEGIN  
      CLOCK_50 <= '0';  
      WAIT FOR 10 ns;  
      CLOCK_50 <= '1';  
      WAIT FOR 10 ns;  
    END PROCESS;
```

```
  vectors: PROCESS
```

```
    BEGIN  
      KEY(0) <= '0'; SW <= "0000000000";  
      WAIT FOR 20 ns;  
      SW(9 DOWNTO 5) <= "01010";  
      WAIT FOR 20 ns;  
      SW(4 DOWNTO 0) <= "11110";  
      KEY(0) <= '1';  
      WAIT;  
    END PROCESS;
```

```
LIBRARY ieee;  
USE ieee.std_logic_1164.all;  
USE ieee.std_logic_signed.all;
```

```
ENTITY testbench IS  
END testbench;
```

```
ARCHITECTURE Behavior OF testbench IS
```

```
COMPONENT display
```

```
PORT ( KEY :IN STD_LOGIC_VECTOR(0 DOWNT0 0);  
SW :IN STD_LOGIC_VECTOR(0 DOWNT0 0);  
HEX0 :OUT STD_LOGIC_VECTOR(6 DOWNT0 0);  
LEDR :OUT STD_LOGIC_VECTOR(9 DOWNT0 0));
```

```
END COMPONENT;
```

```
SIGNAL KEY : STD_LOGIC_VECTOR(0 DOWNT0 0);
```

```
SIGNAL SW : STD_LOGIC_VECTOR(0 DOWNT0 0);
```

```
SIGNAL HEX0 : STD_LOGIC_VECTOR(6 DOWNT0 0);
```

```
SIGNAL LEDR : STD_LOGIC_VECTOR(9 DOWNT0 0);
```

```
BEGIN
```

```
U1: display PORT MAP (KEY, SW, HEX0, LEDR);
```

```
clock_process: PROCESS
```

```
BEGIN
```

```
KEY(0) <= '0';
```

```
WAIT FOR 10 ns;
```

```
KEY(0) <= '1';
```

```
WAIT FOR 10 ns;
```

```
END PROCESS;
```

```
vectors: PROCESS
```

```
BEGIN
```

```
SW(0) <= '0'; -- Resetn = 0
```

```
WAIT FOR 20 ns;
```

```
SW(0) <= '1'; -- Resetn = 1
```

```
WAIT;
```

```
END PROCESS;
```

```
END;
```