HDL Simulation

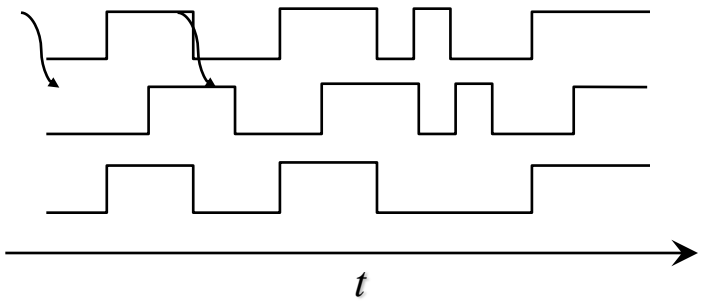
Franco Fummi
Alessia Bozzini

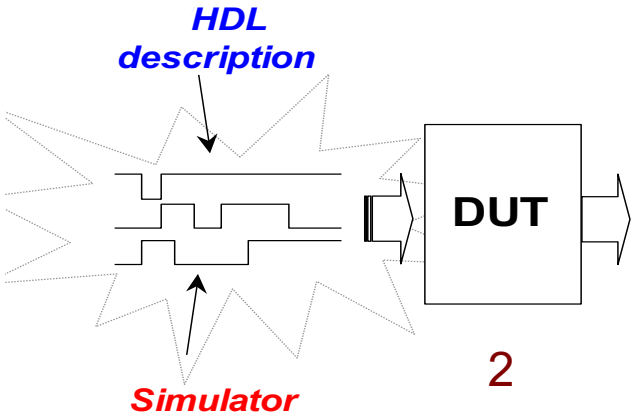
UNIVERSITÀ di VERONA
Dipartimento di INFORMATICA

Version 1.1

Contents

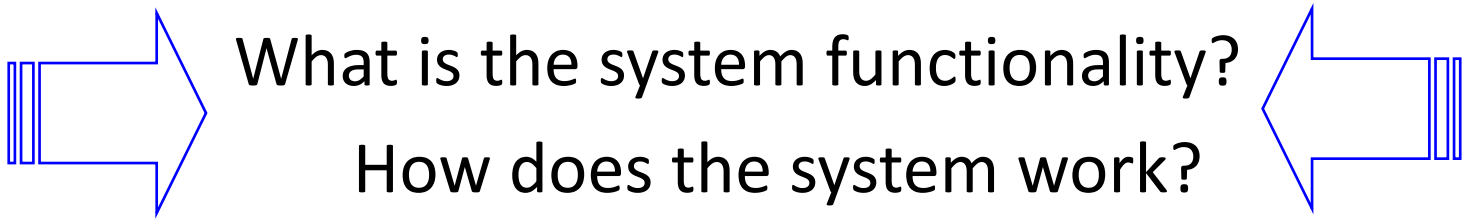
1. Simulation goal
2. The simulation process:
   - Dummy execution
   - HDL model
   - Synchronization
   - Signal
3. HDL Timing Models
   - Time and delay in VHDL
   - Time and delay in Verilog
4. Signal and driver
5. Statement analysis and comparison
   - Signal vs. Variable assignment
   - Wait vs. Sensitivity List
   - Transaction vs. event
   - Wait until conditions
6. Source code optimization
7. Test bench creation in Verilog and VHDL





Simulation Goal

- The HDL language has been defined for simulation purposes, aiming at verifying:
  - the correctness of the specification
  - the correctness of the device functionality w.r.t. user's requirements



Simulation Goal

- When working with complex devices and adopting a Top-Down or Bottom-Up design approach:
  - correctness of each module/sub-module,
  - correctness of partitioning into several sub-modules w.r.t. to a more complex unique module at an higher description level,
  - correctness of connecting several sub-modules for achieving a more complex functionality.

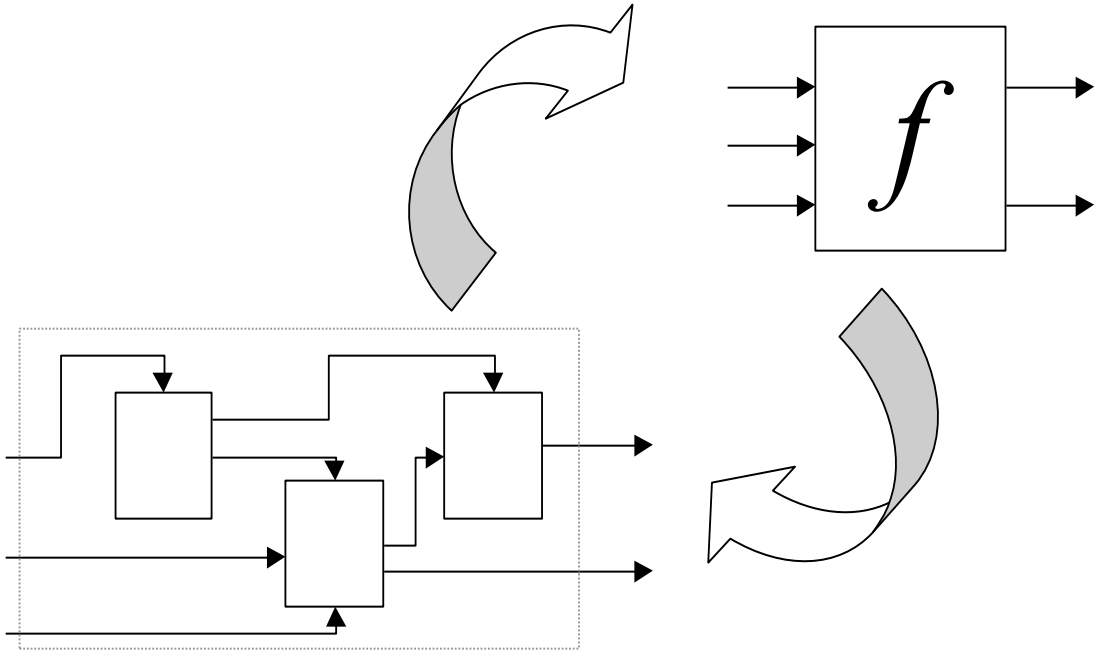
Simulation Goal

- Validation means for partitioning and aggregation

We need a golden model ⇒ E.g.: FLOATING POINT DEVICE UNIT

```
float a;
float b;
float c;
c=a+b;
```

USO FUNZIONALITÀ DI ALTRI LINGUAGGI COME GOLDEN MODULE GIÀ IMPLEMENTATE



Simulation: what we need ...

- Inputs:
  - HDL code description of the device
    - compiled entity & architecture [package]
  - Stimuli for the device
    - test bench
- Outputs:
  - Analysis of the device response to the stimuli:
    - output monitoring
    - additional component for autonomous result confrontation

The simulation process

- HDL Timing model
  - Simulation is event-driven
    - ↳ events on signals trigger the process
  - Times advances to the instant an event occurs
    - ↳ there is not a continuous time evolution
  - Preemptive time model
    - ↳ there are events that are **scheduled** to occur

VHDL & Verilog => no specific clock
Any signal can create events

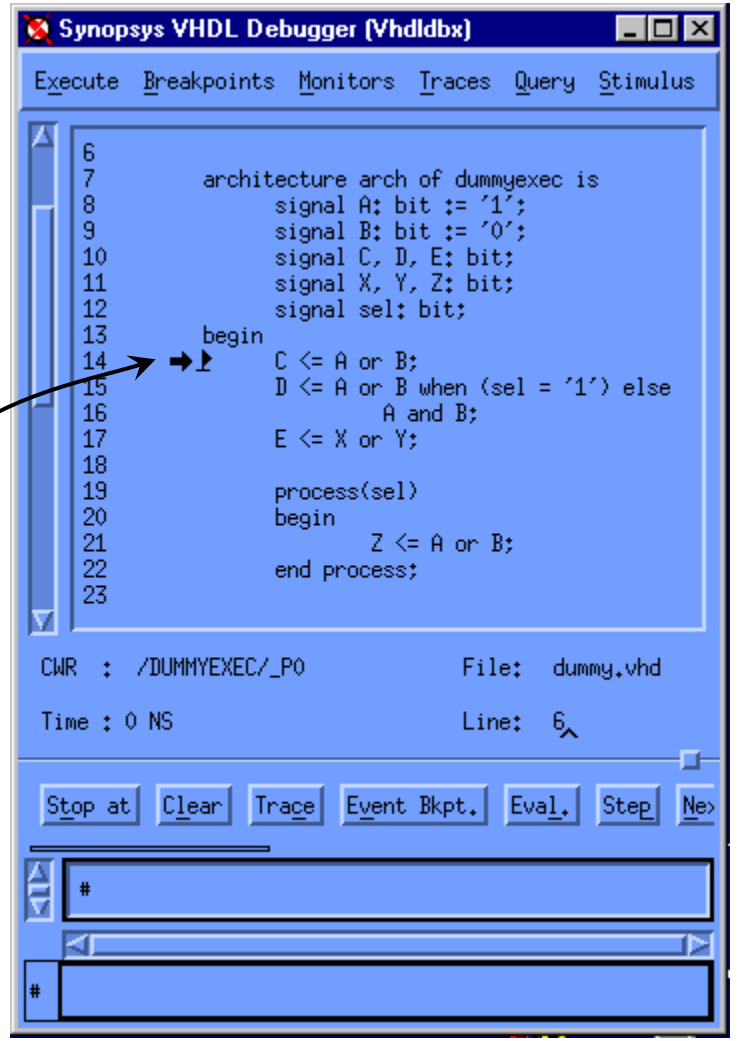
The simulation process

- Simulation cycle => all set of operation in simulator to simulate current time
  - A dummy execution is performed to initialize all signals! => So we can reach time t=0, the starting point of simulation

Schedule events that in reality are in parallel //

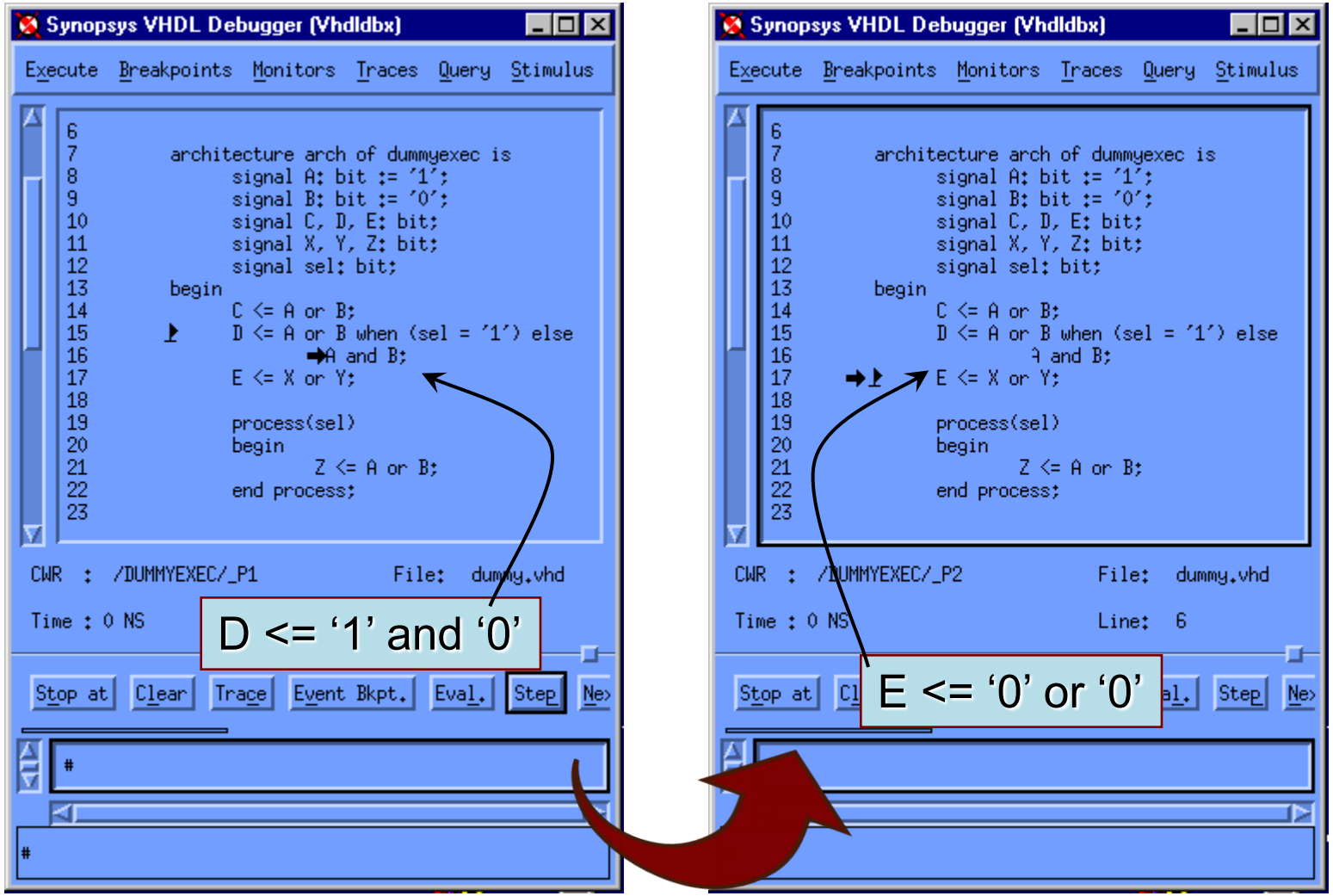
Divide activities

Simulation guarantees there is no difference between real system and simulated one

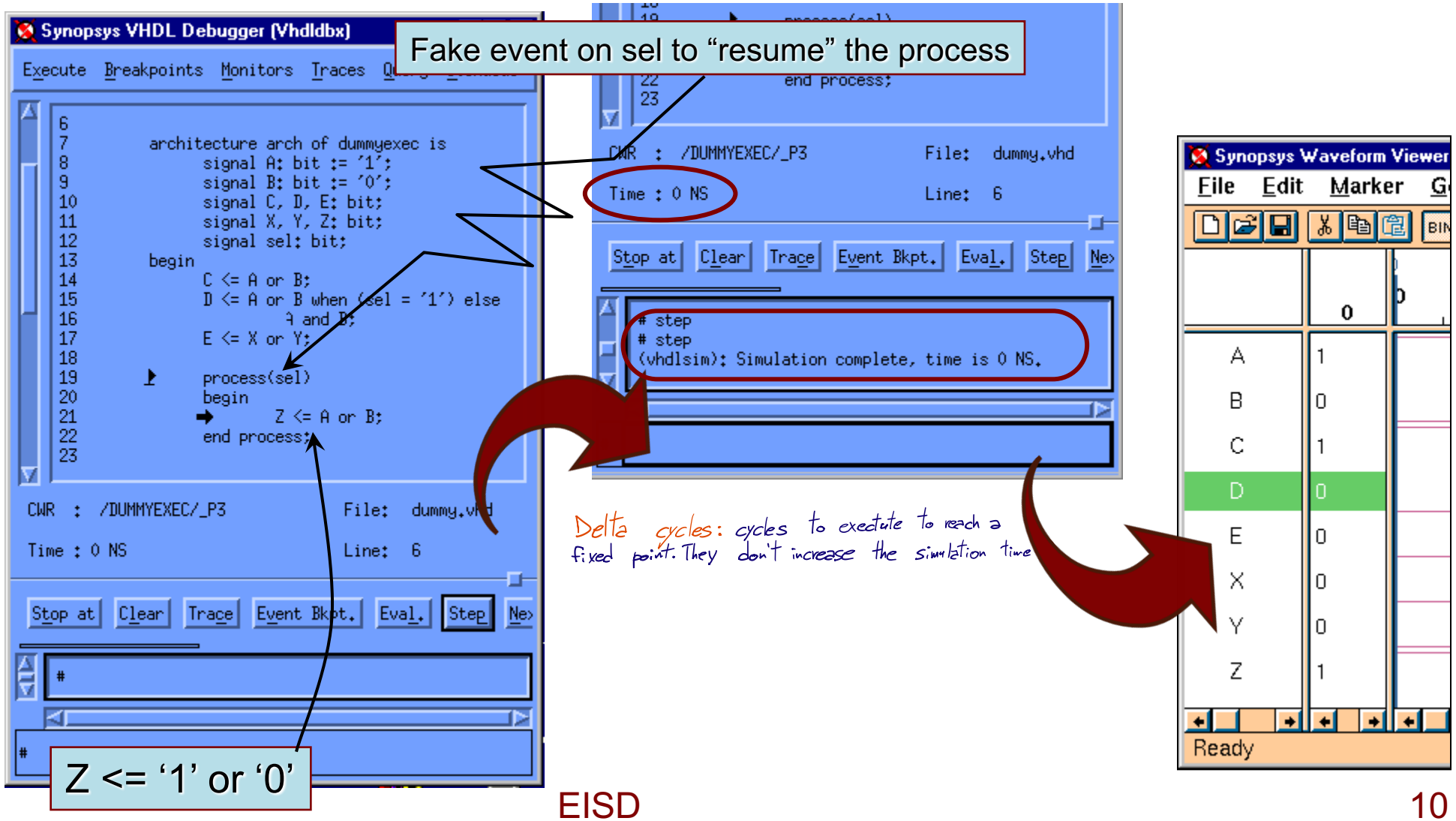


C <= '1' or '0'

The simulation process: dummy execution

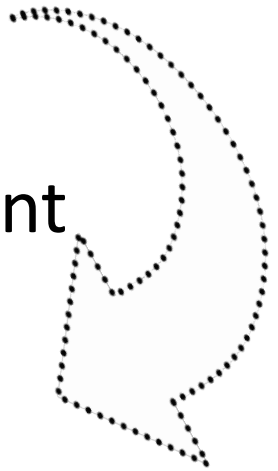


The simulation process: dummy execution



The simulation process

- Simulation cycle
  - At the beginning time is assumed to be 0
  - There are three steps:
    - Time advances to the nearest instant when a driver becomes active or a suspended process resumes
    - Signal values are updated. This operation causes an event (eventually delayed) on each signal changing value
    - Suspended process resume and are executed until they suspend

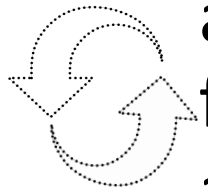


The simulation process: HDL model

- Each concurrent statement is equivalent to a process sensitive to signals on the right-hand side:
  - Each statement and each process is executed asynchronously with respect to each other depending on events.
  - To create a “control” on process (statement) execution hand-shake protocols are implemented

The simulation process: VHDL model

- Process: once resumed, the “internal sequential” code is concurrently executed.
- A process can be viewed as an infinite loop:



after the last statement is executed, the first is executed again and if conditions are met, the loop continues …

- Time does not advance while the process loop evolves unless explicit timing is used:
  - WAIT or AFTER …

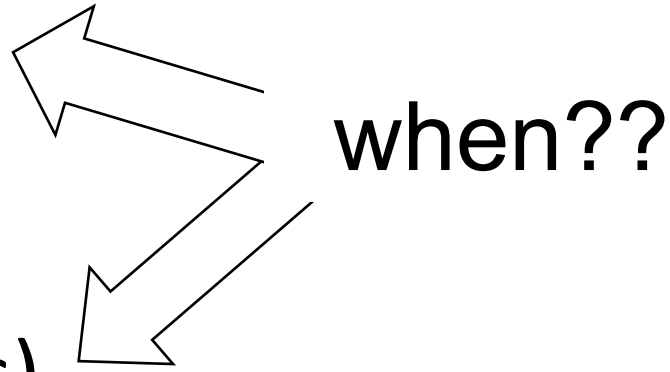
Time advances only because we schedule future events with AFTER or WAIT FOR ...

The simulation process: Synchronization

- Different processes (or concurrent statements) are synchronized by means of WAIT statements:
  - WAIT ON signal list
  - WAIT FOR time delay
  - WAIT UNTIL condition
- Interprocess communication is handled through signals

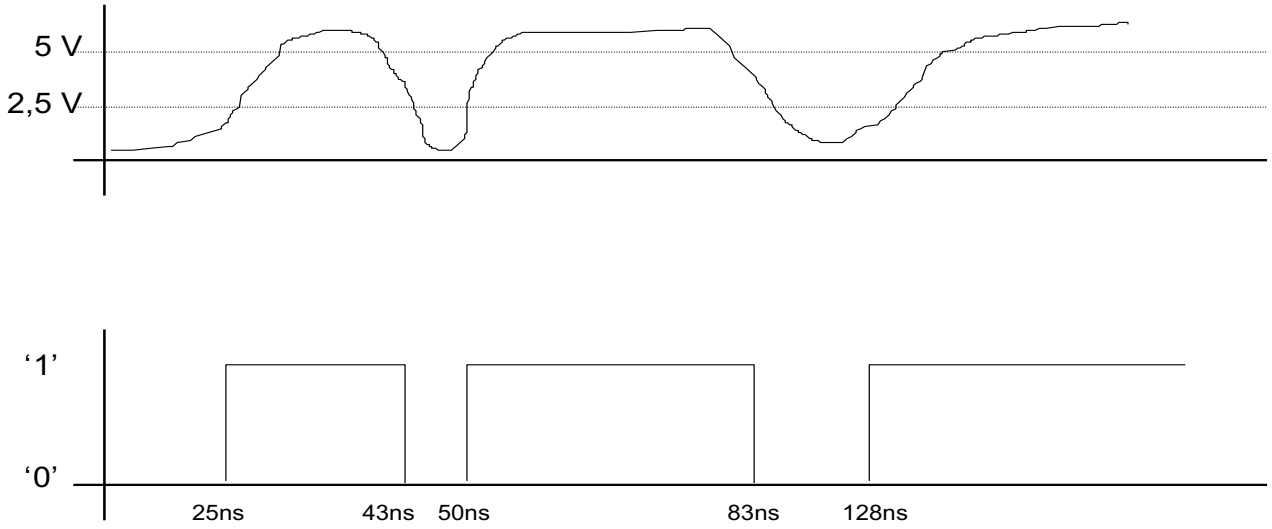
The simulation process: Signals

- A signal is an object with a past history of values
- A signal acts as a medium waveform propagation:
  - physical wires
  - memory elements (latches)



The simulation process: Signals

- Waveforms
  - All transient values are discarded
  - Waveform is a sequence of elements called **transactions**
  - Each change of value is called **event**

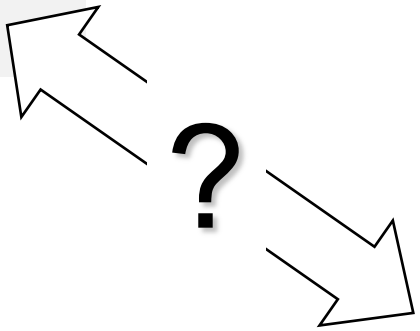


The simulation process: Signals

- Signal assignment
  - A waveform “wants” to describe the physical circuit behavior: carry values at any time

```
P1: Process
begin
    wait until CK = '1';
    A <= B or C;
end process;
```

Event on rising edge of clock



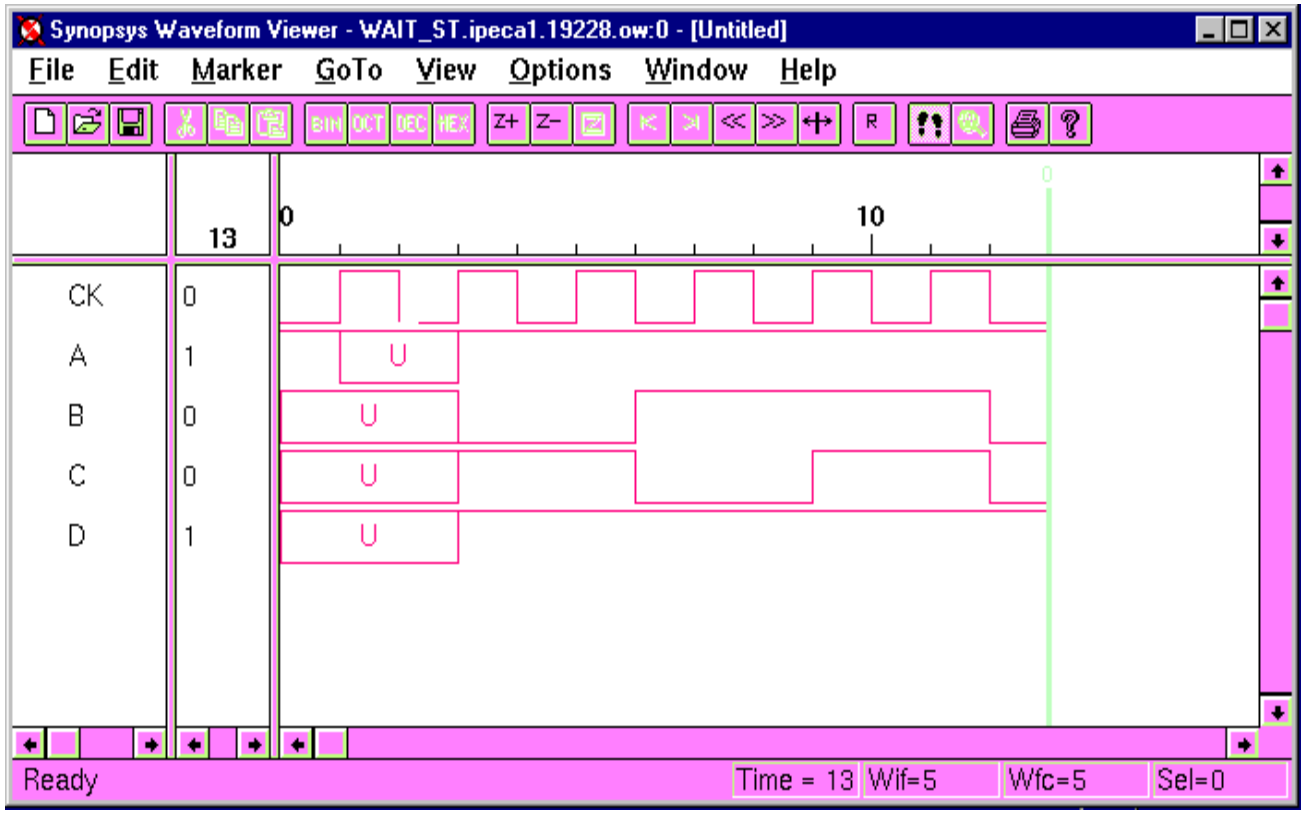
A=? B=?
before the 1st clock edge

```
P2: Process
begin
    A <= B or C;
    wait until CK = '1';
end process;
```

The simulation process: WAIT position

```
entity update_sig is
        port(CK: in std_ulogic);
end update_sig;
architecture arch of update_sig is
        signal A: std_ulogic  := '1';
        signal D: std_ulogic  := '1';
begin
        B <= '0' after 3 ns, '1' after 6 ns, '0' after 12 ns;
        C <= '1' after 3 ns, '0' after 6 ns, '1' after 9 ns,'0' after 12 ns;

        P1: process
        begin
                wait until CK = '1';
                A <= B or C;
        end process P1;

        P2: process
        begin
                D <= B or C;
                wait until CK = '1';
        end process P2;

end arch;
```

The simulation process: WAIT position



The simulation process: Signal *driver*

- Signal

```
B <= ‘1’,'0' after 3 ns,'1' after 6 ns,'0' after 12 ns;
```

waveform elements

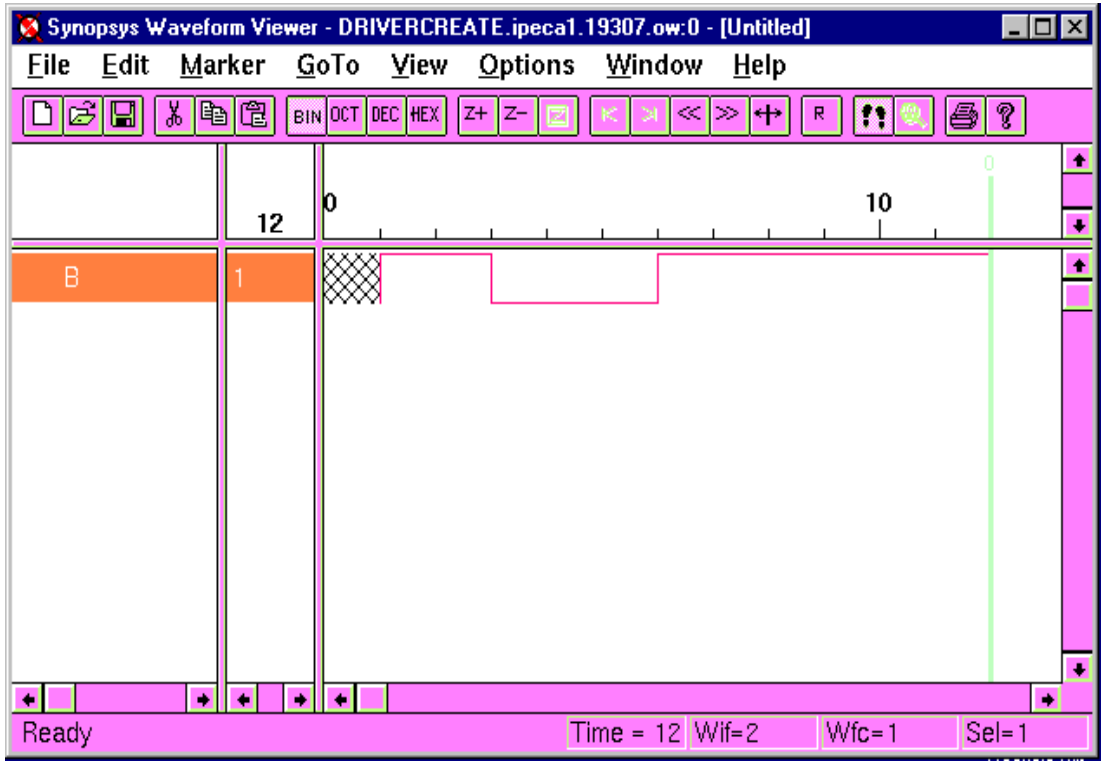
signal

- The simulator creates a DRIVER for the signal, which is a source for the values it will assume
- Each time the signal assignment statement executes, the value of the waveform is appended to the driver when the fixed time arrives
- The DRIVER is the item the simulator accesses for determining the value of the signal

The simulation process: Signal *driver*

- Signal

```
B <= ‘1’,'0' after 3 ns,'1' after 6 ns,'0' after 12 ns;
```



Driver

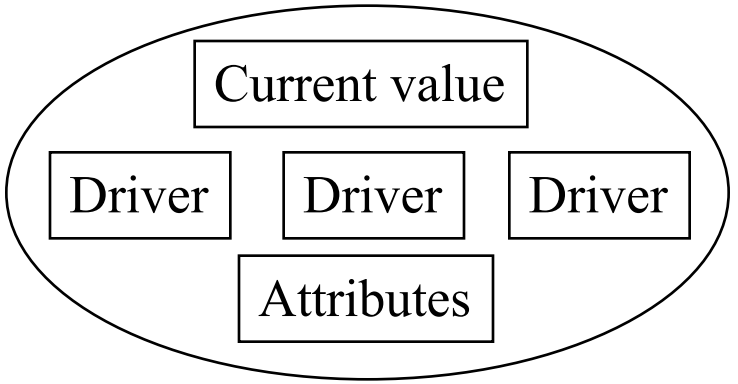
value	time
‘1’	0 ns
‘0’	3 ns
‘1’	6 ns
‘0’	12 ns

The simulation process: Signal *driver*

- Signal
  - A signal has as many drivers as the number of signal assignments which are not mutually exclusive (i.e. in different processes)
  - If a signal has more than one driver a **resolution function** is required: determines which value prevails
  - The resolution function is necessary even if no conflict arises

The simulation process: signal *driver*

- The signal is not a mere container of data.
- It also contains attribute information on:
  - past values
  - past events
- It can be used to generate new signals (delayed waveforms)



The simulation process: special signals

- Guarded signals
  - There is a guard (a boolean expression) which “turns off” the signal driver when a specified condition is not true.
  - Signal needs to be declared as:
    - REGISTER: retains the last output while the driver is disconnected
    - BUS: re-evaluates the output value

```
SIGNAL temp: wired_or bit_vector (0 to 7) BUS;
```

It applies to busses or buffers on bus

Resolution function
(it’s a bus!)

HDL Timing Models

- **Delay**: the time period between the current time and the instant time when an event will occur
  - Time and delay in VHDL
  - Time and delay in Verilog

HDL Timing Models: Time and delay in VHDL

- An implicit delay model:
  1. delta delay
- Two explicit kinds of delay models:
  2. inertial
  3. transport

Time and delay in VHDL

- Delays: delta DOES NOT IMPACT ON SIMULATION TIME
  - Although no time is specified for the signal assignment statement, its execution does not affect the current value of the signal
    - effects occur after an infinitesimally small delay called delta delay
  - A signal assignment never modifies the past or current value of a signal, it only affects the *scheduled* future values.
  - Example statement:

```
gamma <= alpha or beta;
```

Time and delay in VHDL

- Delays: inertial
  - Appropriate for modeling switching circuits.
    - A pulse shorter than the switching time of the circuit is not transmitted
  - It is modeled with an AFTER clause.
  - Example statement:

```
gamma <= alpha or beta AFTER 5 ns;
```

  - *All signal assignments are assumed to have an inertial delay of 0 ns whenever no specification is given.*

Time and delay in VHDL

- Delays: transport
  - Allows modeling transmission lines where every pulse, independent of its duration, should be transmitted.
  - It is achieved by adding a TRANSPORT keyword.
  - Example statement:

```
gamma <= TRANSPORT alp or bet AFTER 5 ns;
```

HDL Timing Models: Time and delay in Verilog

- A delay is specified by a `#` followed by the delay amount
- The exact duration of the delay depends upon timescale
- For example, if with `` `timescale 2ns/100ps ``, a delay with statement

```
#50;
```

will mean a delay of 100 ns

Time and delay in Verilog

- Intra assignment delay
  - Delays can also be specified within an assignment statement as in

```
p = #10 (a | b);  // Example of intra-assignment delay
```

This statement is interpreted as follows - First evaluate the right hand expression (a | b). Then wait for 10 units of time (again remember that unit of time is defined in timescale). After this wait, assign the value of RHS to LHS

  - The above statement is equivalent to

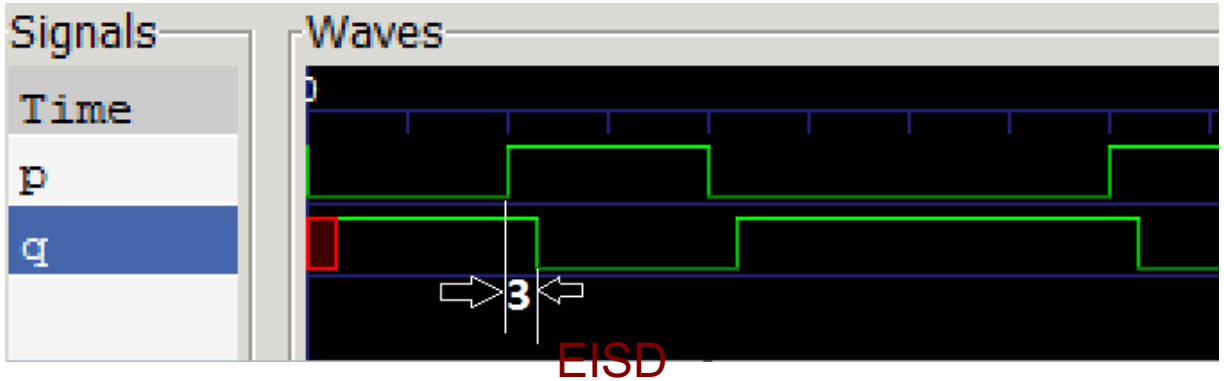
```
temp = (a | b);   // evaluate the RHS and hold it temporarily
#10;              // Wait for 10 units of time
p = temp;         // Assign the temporary evaluation to LHS
```

Time and delay in Verilog

- Delays for Primitive Gates
  - Real circuits in ASIC have delays. For example logic gates may have propagation delays associated with them
  - It is possible to specify the Gate Delay as in the following example

```
module notgatedelay( input p, output q );
     not #3 (q,p) ;
endmodule
```

The input p will be delayed by 3 units of time through the not gate for both the rising as well as falling edge of the input p. This can be seen in the following timing diagram



Time and delay in Verilog

- Rise and Fall Delay
  - It is possible to define the rise and fall delays separately as in follwing example

```
module andgatedelay( input p,q, output r );
     and #(2,4) u_and (r,p,q);
endmodule
```

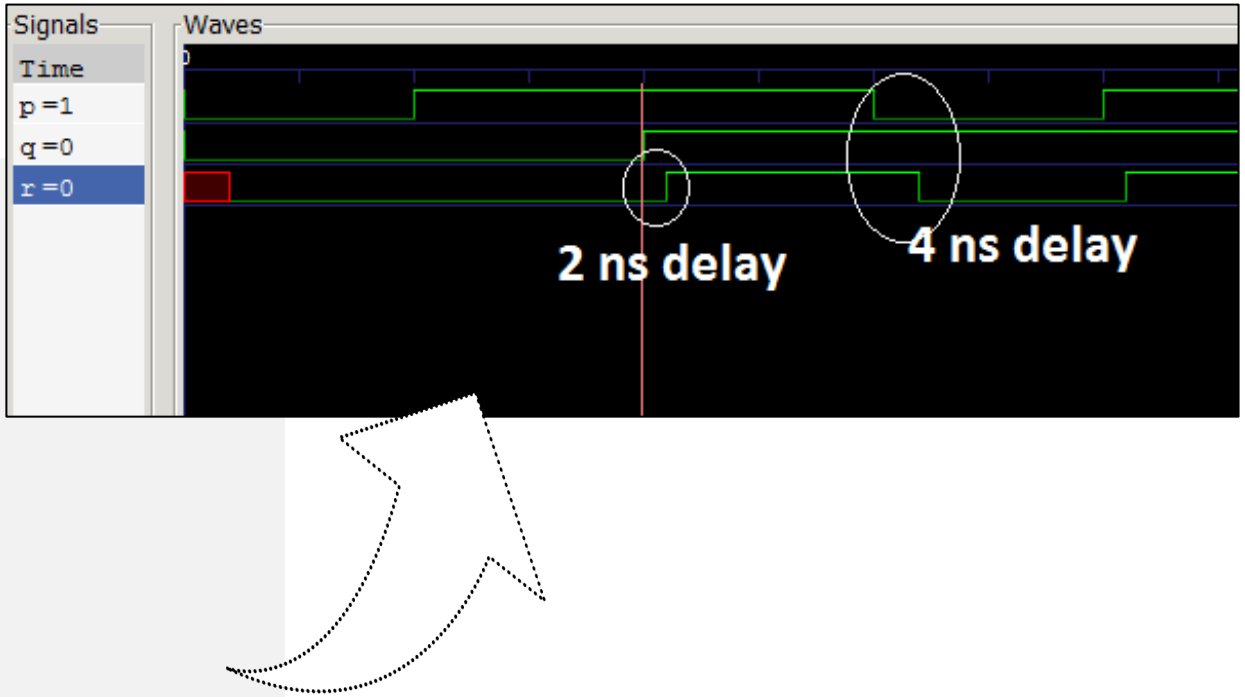
In this case the rise time delay is 2 units and the fall time delay is 4 units
The two delays are separated by comma as in

```
and #(2,4) u_and (r,p,q);
```

Time and delay in Verilog

- Rise and Fall Delay
  - The test bench for preceding code is

```
`timescale 1ns / 100ps
module stimulus;
      // Inputs
      reg p,q;
      wire r;
      andgatedelay n1 ( .p(p), .q(q), .r(r) );
      initial begin
            $dumpfile("test.vcd");
      $dumpvars(0,stimulus);
      p = 0; q =0;

      #20 p = 1;
      #20 q = 1;
      #20 p = 0;
      #20 p = 1;
      #40;
end
      initial begin
            $monitor($time, " p=%1b,q=%1b, r = %1b",p,q,r);
      end
endmodule
```



Signal and its driver

- The relation on the values in the driver and the values the signal assumes:
  1. Current value update
  2. Driver filtering
  3. Driver erasing

Current value update

- After a sequential signal assignment with no delay (zero-delay assignment), the current value of the target signal is not the result of the right expression. The value is placed in the driver.
- The current value of the signal is only updated on synchronization points: wait statements.
- When a sensitivity list is used, the current values of the target signals are updated just before the end process keywords.

Current value update

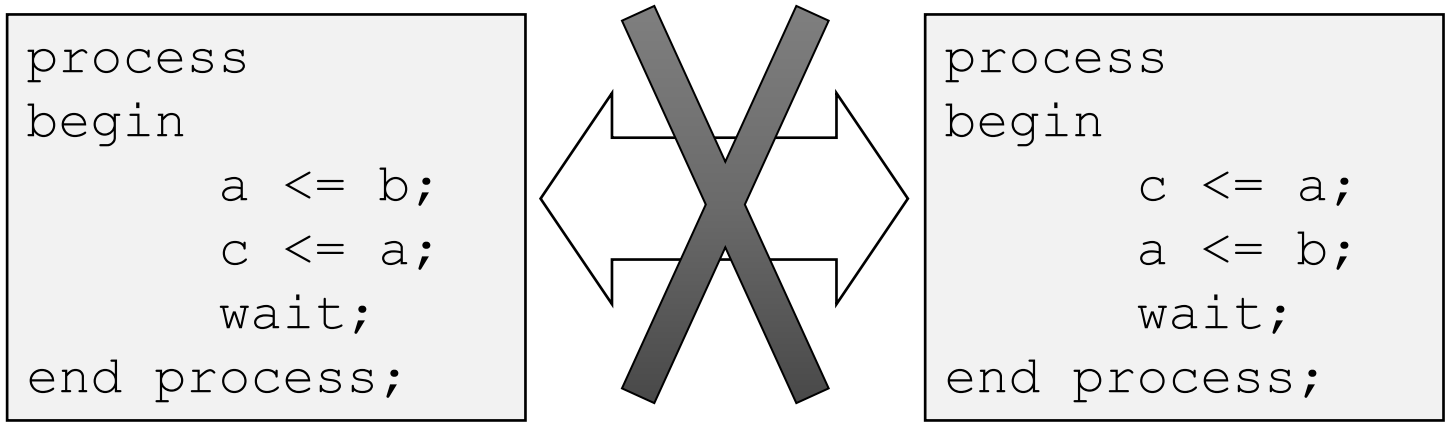
- When no wait statement is necessary but updated values are required, it is possible to create dummy synchronization points
- Goal: force the updating of the current values of the zero-delay assignment target signals
- Possible forms of synchronization points:
  - `wait for 0ns;`
  - `… <= … AFTER 0ns;`

Current value update

- Variables are immediately updated. If target signals have no interprocesses communication functionality, a variable should be used
- The use of a signal (instead of a local variable) to store only an intermediate result of a process is very expensive

Current value update

- The mechanism of signal current value updating provides the concurrency among signal assignment statements
- Signal assignment is not variable assignment even in the sequential domain!



Driver filtering

```
entity assign is
end assign;

architecture arch of assign is
   signal A: BIT := '0';
   signal B: BIT := '0';
begin
   Wave_gen: process
   begin
    A <= '1' after 5 ns, '0' after 12ns;
    wait;
   end process Wave_gen;
   C <= A or B after 40 ns;
   X <= transport A or B after 40 ns;
end arch;
```

↳keep track of the signal over the time

Driver filtering

- Driver Analysis

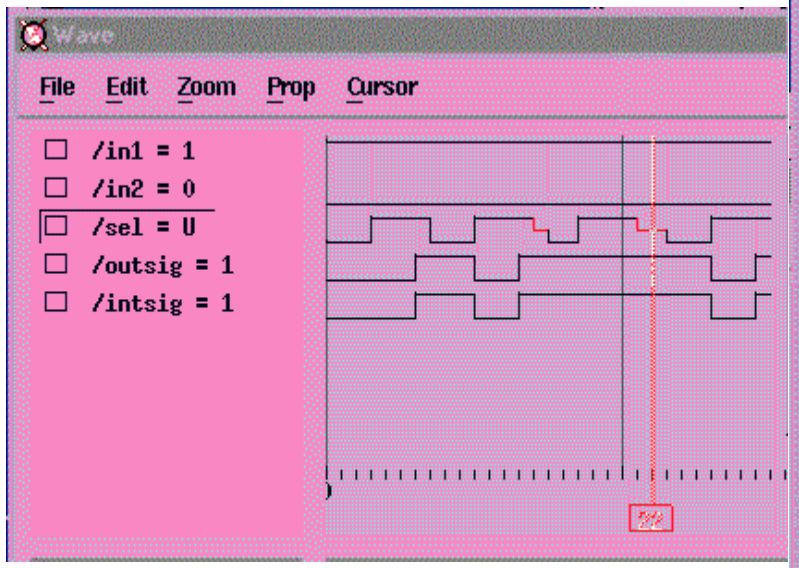
TIME	Events	C	Driving delay	X	Scheduled events
ns					
0	A = '0'	40:'0'		40:'0'	
5	A = '1'	~~40:'0'~~,45:'1'			40:'0',45:'1'
12	A = '0'	~~45:'1'~~,52:'0'			40:'0',45:'1',52:'0'
40	**X <='0'**	52:'0'	----	45:'1',52:'0'	
45	**X <='1'**	52:'0'	----		52:'0'
52	**C <='0'**				
	X <='0'				

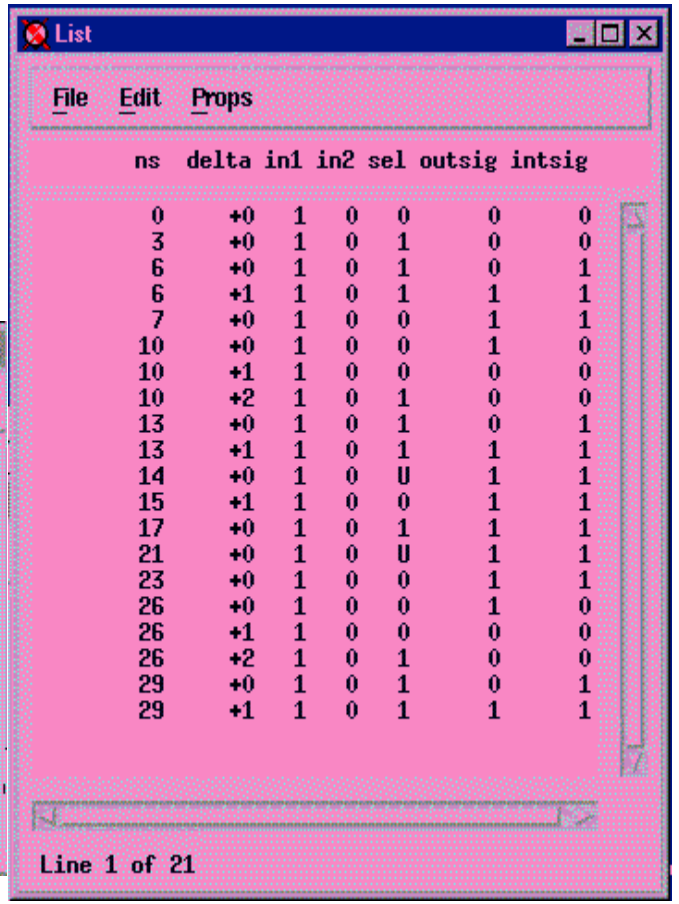
inertial

transport

Driver erasing

- When trying to preserve a value of a signal in case some conditions are not met, it may occur that the driver is re-written modifying the current signal driver
- Cause: inappropriate use of else clause in selected/conditional state assignments





Simulation: variables

- Variables within subprograms are initialized each time the subprogram is called
- Variable assignments cannot be delayed

- What's a variable transformed into? It depends on how I write the code

Statement analysis and comparison

- Statements Comparison
  1. Sensitivity List vs. WAIT
  2. ‘TRANSACTION and ‘EVENT
- Statement analysis
  3. Sensitivity list control
  4. Wait until condition

Statements comparison

- Sensitivity list vs. WAIT

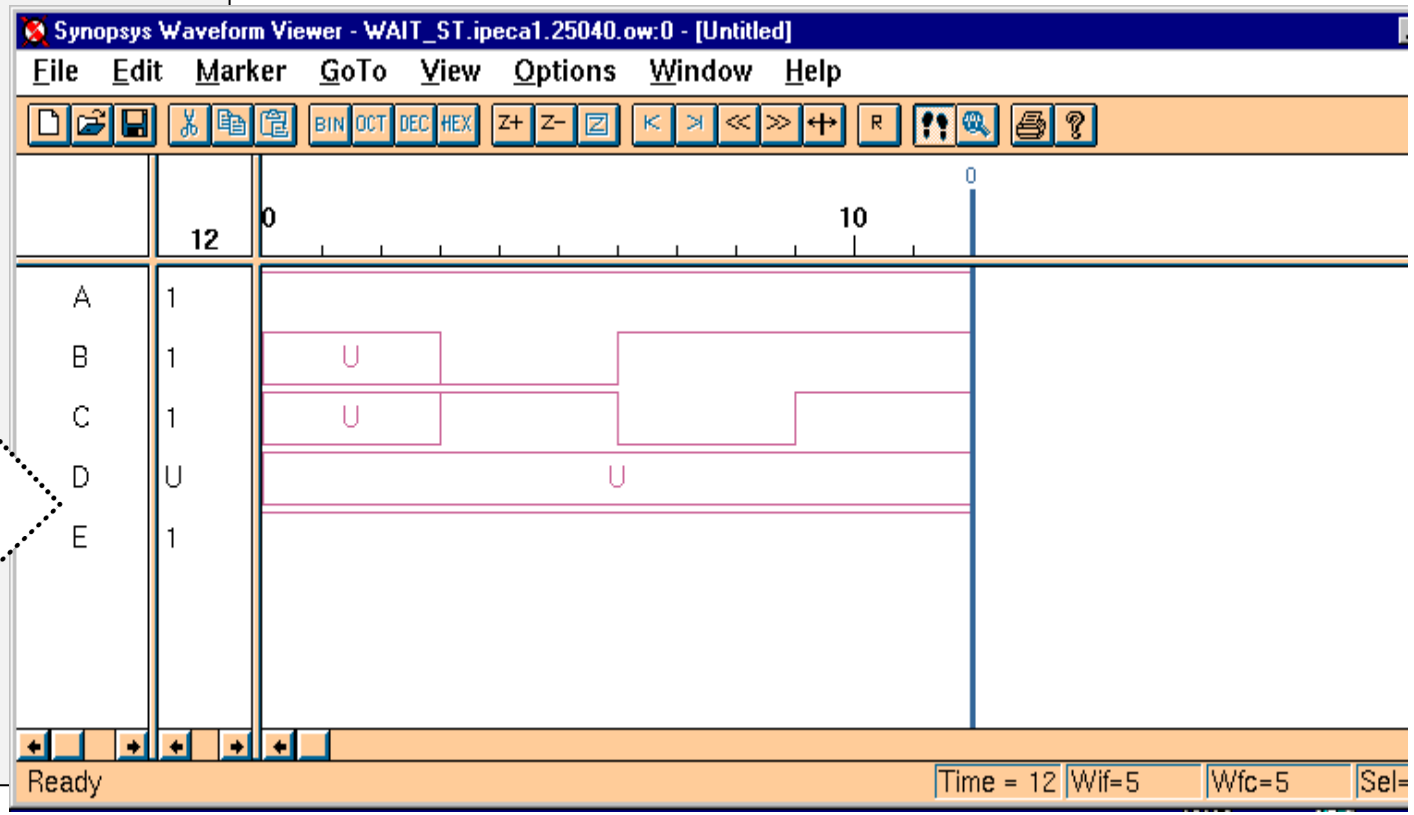
```
P1:process(clk)
begin
 if (clk='1') then
   A <= B and C;
 end if;
end process;
```

```
P3:process
begin
 A <= B and C;
 wait until clk='1';
end process;
```

```
P2:process
begin
   wait until clk='1';
   A <= B and C;
end process;
```

Statements comparison

```
architecture arch of wait_st is
    …
begin
    B <= '0' after 3ns, '1' after 6ns;
    C <= '1' after 3ns, '0' after 6ns, '1' after 9ns;
P1: process
begin
    wait until CK = '1';
    A <= B or C;
end process P1;

P2: process
begin
    D <= B or C;
    wait until CK = '1';
end process P2;

P3: process(CK)
begin
    if (CK = '1') then
            E <= B or C;
    end if;
end process P3;
end arch;
```

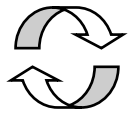


Statements comparison

- 'TRANSACTION and 'EVENT
  - Transaction: re-evaluation of the signal value (does not imply a change in the value) → operation on the signal
  - Event: change of the signal value

```
DIV: process
begin
   R <= A/B;
   M <= A MOD B;
   wait on A, B;
end process DIV;
```

```
MAIN: process
begin
  wait on M;
       A <= M;
    end process MAIN;
```

M = 0, A = 6, B = 2 ⟶ M = 0 ⟶ 

- No event on M.
- Processes are stalled!
- wait on A ⟶ wait until A'TRANSACTION
- wait on M ⟶ wait until M'TRANSACTION

Statement Analysis

- Sensitivity List Control
  - For avoiding possible critical initialization and racing problems the following suggestions should be taken into account while defining the module behavior:
    - a signal must not be read before a wait until statement;
    - signal assignment statements are not allowed before a wait until statement, because the value read from the signal according to the simulation semantics might deviate from the value produced by the corresponding logic network

Statement anaysis

- Sensitivity List Control
  - The more explicit the sensitivity list is, the more efficient is the running of the code, and the easier is the tracking of erroneous behaviors

```
B1: block (S'EVENT and S='1')
begin
     X <= guarded Z1;
     Y <= guarded Z2;
end block B1;
```

  - Assignments of X and Y are supposed to be carried out only on the rising edge of S, in fact they are executed on the high state of S. Each time an event occurs on Z1 or Z2, the value of the guard is checked (not re-evaluated) and if the value of S'EVENT is true (due to a previous change of value) and S='1' the assignments are performed

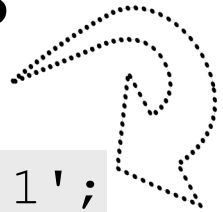
```
GUARD STATEMENT: not S'STABLE and S='1'
```

Statement analysis

- Wait until conditions

1

```
wait until CLK='1';
wait on CLK until CLK='1';
```



- *Implicitly assumed since no explicit sensitivity list was expressed*

2

```
wait on RESET until CLK='1';
```

- *No sensitivity to CLK, condition verified only for an event on RESET*

3

```
wait until (CLK='1' and ENABLE='1')
```

Interpreted as

```
wait on CLK,ENABLE until (CLK='1' and ENABLE='1')
```

Desired to be?

```
wait on CLK until (CLK='1' and ENABLE='1')
```

Source code optimization

- Guidelines for writing a VHDL specification that can be easily simulated
- Not necessarily these suggestions hold when dealing with the synthesis task

Source code optimization

- VHDL offers a wide range of data types
  - When a complex algorithm has to be used, performance depends on the selected datatypes
  - Arithmetic operation will be more efficient when working on integers than on bit vectors
- A signal is expensive
  - The simulator needs to build a driver and to schedule future values
  - Whenever a variable can fit, the use of a signal should be avoided
  - Signals should be used only for interprocess communication

Source code optimization

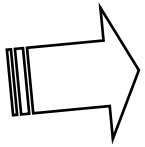
- File operations:
  - I/O is always time consuming but …
  - if a complex test bench needs to be created, the final system Device Under test and Test Bench, may require a too big amount of memory
  - It’s a more flexible solution

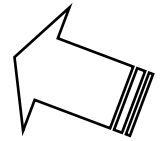
Source code optimization

- Sensitivity lists:
  - Split the VHDL into processes sensitive to a *minimal* number of signals
    - Conditional statements try to know which signal is active. This may imply the storing of previous values for a comparison
    - For activating a suspended process is necessary to verify a high number of signals to detect if events have occurred
- Wait statement
  - prefer **wait on** and **wait for** statements to the **wait until** form
    - the former two are cheaper because they are static and most of the work is done (once) at compilation time

Simulating …

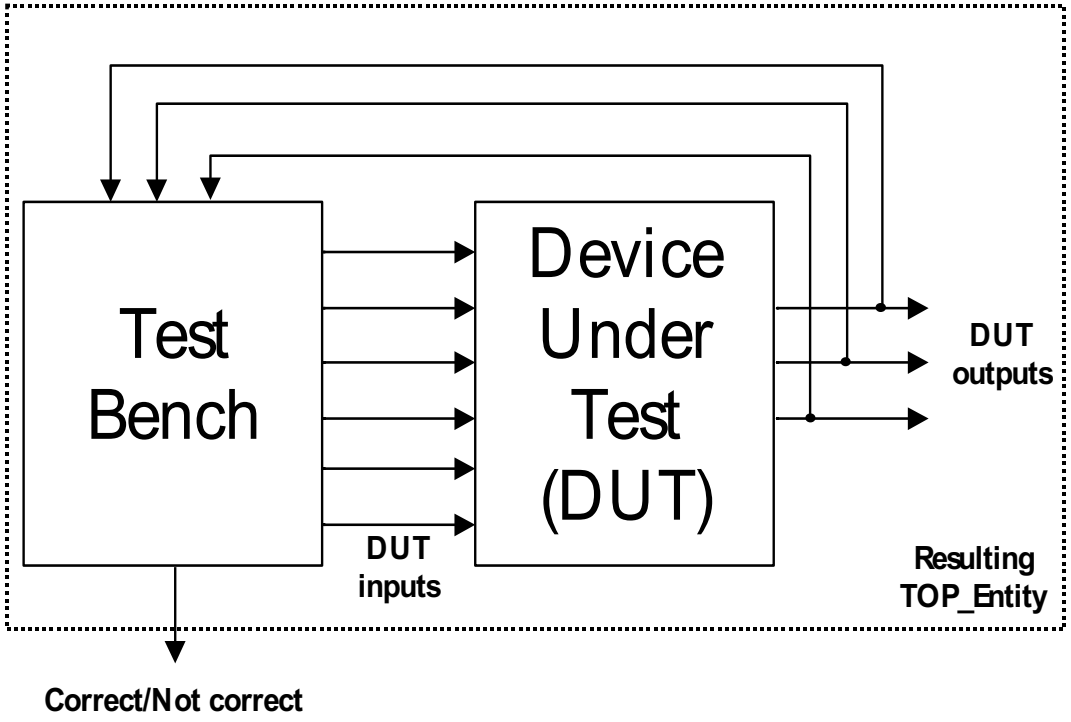
- Stimuli generation: possible approaches
  - Creation of a test bench which autonomously generates stimuli and controls the output produced by the Device Under Test
  - Definition of the stimuli directly in the simulator and manual control of the output traces
  - Creation of a Test Bench for producing stimuli and manual evaluation of the results





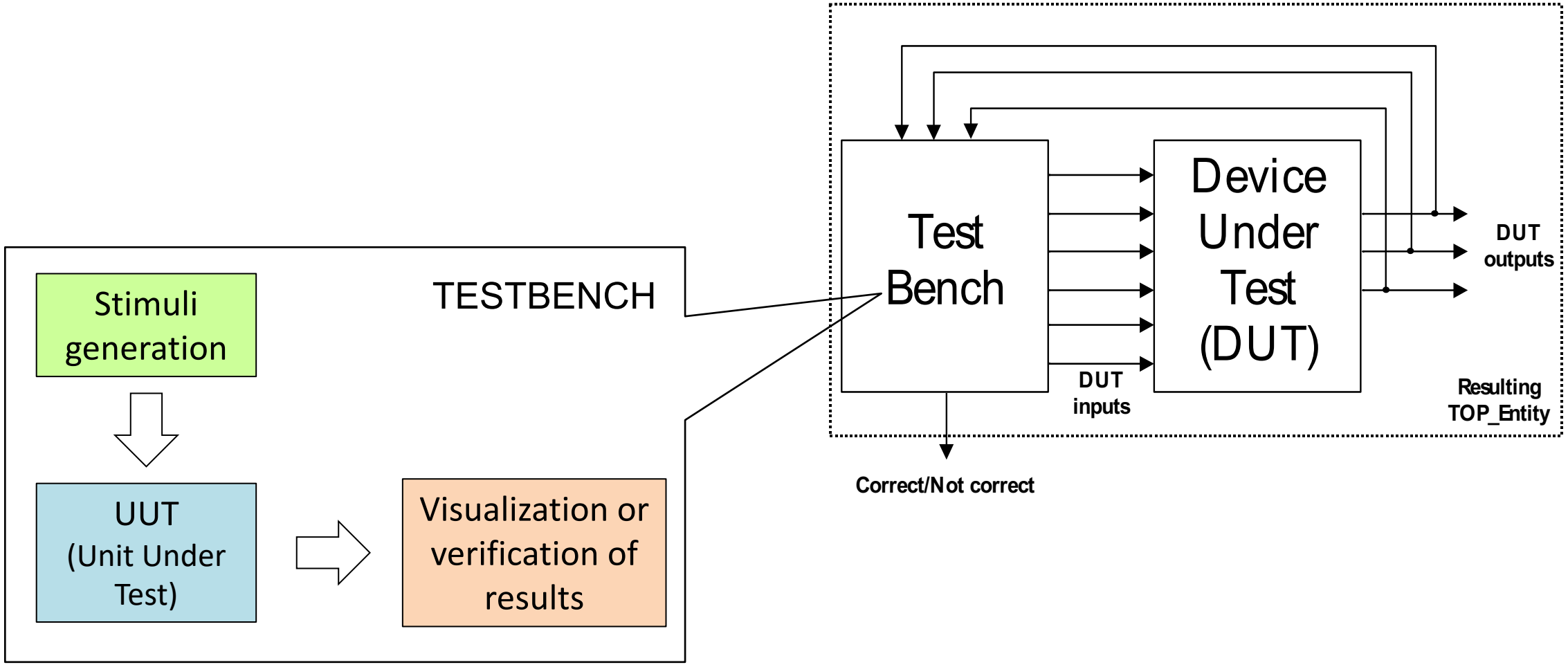
Test Bench

- Structure:



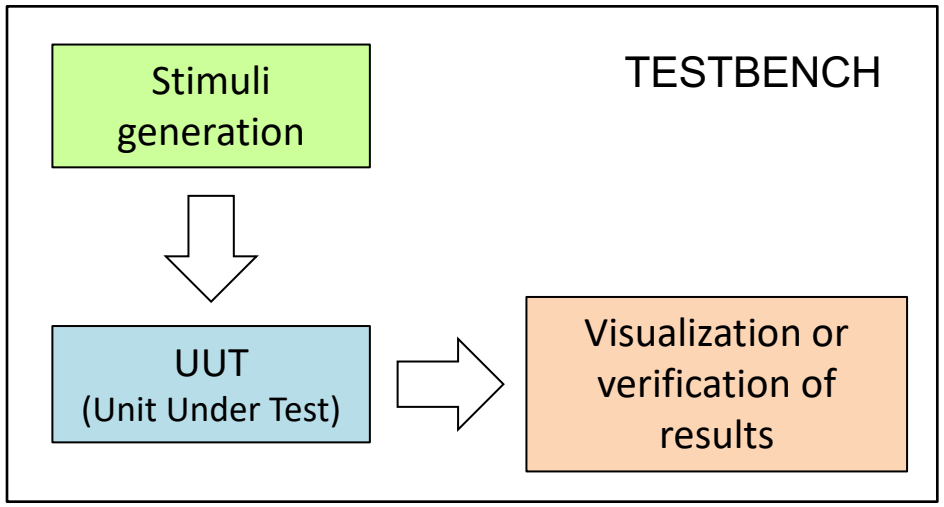
Test Bench

- Structure:



Test Bench: example in Verilog

- Structure:




```
module tb_and;
reg A,B;
wire Z;

my_and UUT(A,B,Z);
Initial
   begin
      A=0;
      B=0;
      #5 A=1;
      #5 A=0;
      B=1;
      #5 A=1;
      #5 $stop;
   end
Initial
      $monitor($time,, A,B,,Z);

endmodule
```

Instantiation of the module to be tested

Stimuli generation

Displaying outputs

Test Bench in Verilog

- In Verilog expressions starting with `$` are system functions
- Display and control functions
  - `$monitor("any string",,var1,"str2",var2)`
    Used to display the results
  - `$stop`
    Used to interrupt the simulation after a certain time
- Accessing files
  - `$readmemb ("filename",memory_name[, initial_address [,final_address]])`
    to read data in binary form from files
  - `$readmemh ("filename",memory_name[, initial_address [,final_address]])`
    to read data in hexadecimal form from files

Test Bench: TEXT I/O example in Verilog

- Reading vectors from a data file

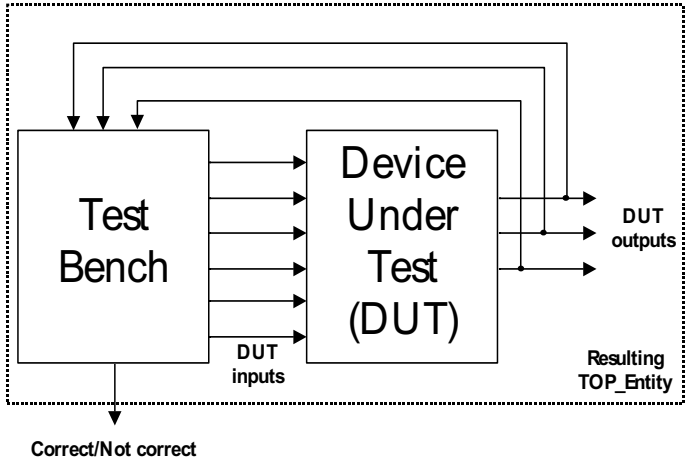
```
"prova.txt"
0 0 00000000
1 0 00000000
1 0 00000000
0 1 00000000
0 1 00000000
0 0 00000001
1 0 00000001
1 0 00000001
0 1 00000001
0 1 00000001
0 0 00000001
```

```
module testbench;
reg [7:0] mem [0:9];
reg [7:0] outbus;
Integer kk;

...

Initial
    $readmemb("prova.txt",mem);

...

endmodule
```

In the testbench mem is an array of 10 positions, each of 8 bits. The values to be assigned to each bit are taken from the file prova.txt which is a simple text file

Test Bench

Module in my project



- For the generation of the waveforms constituting the DUT inputs, two strategies can be adopted:
  - Architecture explicitly generating waveforms
    - *standard* architecture with signal assignments (concurrent or process)
  - Text I/O
    - access to an external file defining the signal assignments, through procedures
  - The Test Bench entity (or the Top entity) has no outputs in case it does not control the correct behavior of the Device Under Test

Test Bench: example in VHDL

- The resulting system to be simulated is constituted by the two components: DUT and Test_DUT

```
ENTITY DUT IS
 PORT(DUT_ouputs: OUT DUT_type;
      DUT_inputs: IN  DUT_type);
END DUT;
```

```
ENTITY Test_DUT IS
 PORT(DUT_ouputs: IN  DUT_type;
      DUT_inputs: OUT DUT_type;
      Correct:    OUT Bit);
END Test_DUT;
```

Test Bench: example in VHDL

```
ENTITY Top_Test_DUT IS
    PORT(Correct:  OUT  Bit);
END Top_Test_DUT;
ARCHITECTURE Top_ST OF Top_Test_DUT IS
    COMPONENT DUT
     PORT(...)
    END COMPONENT;
    COMPONENT Test_DUT
     PORT(...)
    END COMPONENT;
    SIGNAL x, y: DUT_type;
    ...
BEGIN
    u1:   DUT  PORT MAP (x,y);
    u2:   Test_DUT PORT MAP (x,y);
END Top_ST;
```

Test Bench

- Instead of having a specific Test Bench entity and then a Top entity including the two components it is possible to have a Test Bench instantiating as a component and providing the desired stimuli at its inputs
  - Synopsys autonomously produces the “frame”

Test Bench: example in VHDL

```
ENTITY Top_Test_DUT IS
   PORT(Correct:    OUT Bit);
END Top_Test_DUT;
ARCHITECTURE Top_ST2 OF Top_Test_DUT IS
   COMPONENT DUT
    PORT(...)
   END COMPONENT;
   SIGNAL x, y: DUT_type;
   ...
BEGIN
   u1:  DUT PORT MAP (x, y);
   x<= stimuli generation ...
   y<= stimuli generation ...
END Top_ST2;
```

Test Bench: example in VHDL

- DUT: full adder

```
ENTITY FAdd IS
      PORT(A, B: in bit;
           Cin:  in bit;
           Sum:  out bit;
           Cout: out bit);
END FAdd;
```

```
ENTITY Test_FAdd IS
   PORT(Correct:  out bit);
END FAdd;

ARCHITECTURE Test_Arc OF Test_FAdd IS
   COMPONENT FAdd
    PORT(A, B, Cin: in  bit;
         Sum, Cout: out bit);
   END COMPONENT;
   SIGNAL A, B, Sum, Cin, Cout: bit;
BEGIN
   UUT:  FAdd PORT MAP(A,B,Cin,Sum,Cout);
```

Test Bench: example in VHDL

```
stim: process
     type Entry is record
          A,B,Cin:  bit;
          Sum,Cout: bit;
     end record;
     type Tab is array (0 to 7) of Entry;
     constant TTab: Tab :=
               (------A----B---Cin--Sum-Cout----
                ('0', '0',  '0',   '0',   '0'),
                ('0', '0',  '1',   '1',   '0'),
                ('0', '1',  '0',   '1',   '0'),
                ('0', '1',  '1',   '0',   '1'),
                ('1', '0',  '0',   '1',   '0'),
                ('1', '1',  '0',   '0',   '1'),
                ('1', '1',  '0',   '0',   '1'),
                ('1', '1',  '1',   '1',   '1')
          );
```

```
 begin
      for i in TTab'range loop
           A <= TTab(i).A;
           B <=      TTab(i).B;
           Cin <= TTab(i).Cin;
           wait for 1 ns;
           if (Sum = TTab(i).Sum) and
              (Cout = TTAb(i).Cout) then
             Correct <= '1';
           else
             Correct <= '0';
           end if;
      end loop;
      wait;
 end process;
end Test_Arch;
```

Test Bench: TEXT I/O example in VHDL

- Reading vectors from a data file

```
"stimulus_sender.vec"
     0 0 00000000
     1 0 00000000
     1 0 00000000
     0 1 00000000
     0 1 00000000
     0 0 00000001
     1 0 00000001
     1 0 00000001
     0 1 00000001
     0 1 00000001
     0 0 00000001
```

```
process(clk)
   file dt_in: text is in "stimulus_sender.vec";
   variable in_line: line;
   variable ack_v,data_received_v: STD_LOGIC;
   variable dummy_char: character;
 begin
   if(not(endfile(dt_in)) and (clk = '1')) then
     readline (dt_in, in_line);
     read(in_line, ack_v);
     read(in_line, dummy_char);
     read(in_line, data_received_v);
     data_received <= data_received_v;
     ack <= ack_v;
   end if;
 end process;
```

Simulating ...

- The alternative to Test Bench definition is the use of the simulator itself which allows the user to directly force (even randomly) the input values to simulated entities.
- It is possible to save input waveforms for re-using them at different levels of abstractions or for comparing different and, presumably, equivalent implementations.
- It is possible to save output waveforms but there is no mechanism for automatically controlling the correctness of the obtained simulation results.