AARHUS
UNIVERSITY
SCHOOL OF ENGINEERING

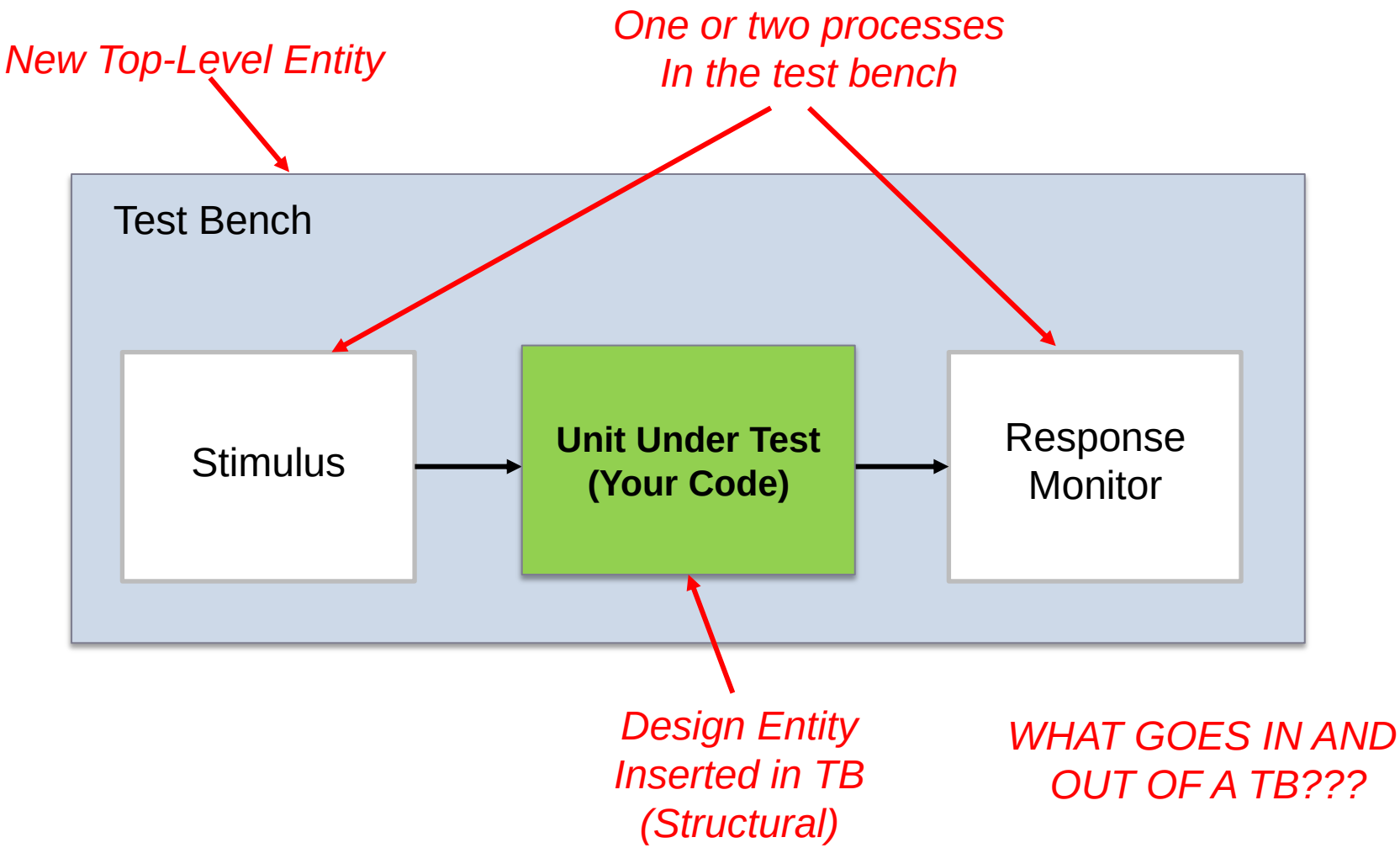
PETER HØGH MIKKELSEN
29. JANUARY, 2015

COMBINATORIAL TEST BENCHES

TEST DRIVEN DEVELOPMENT

› What method makes sense when?

TEST BENCH BASICS

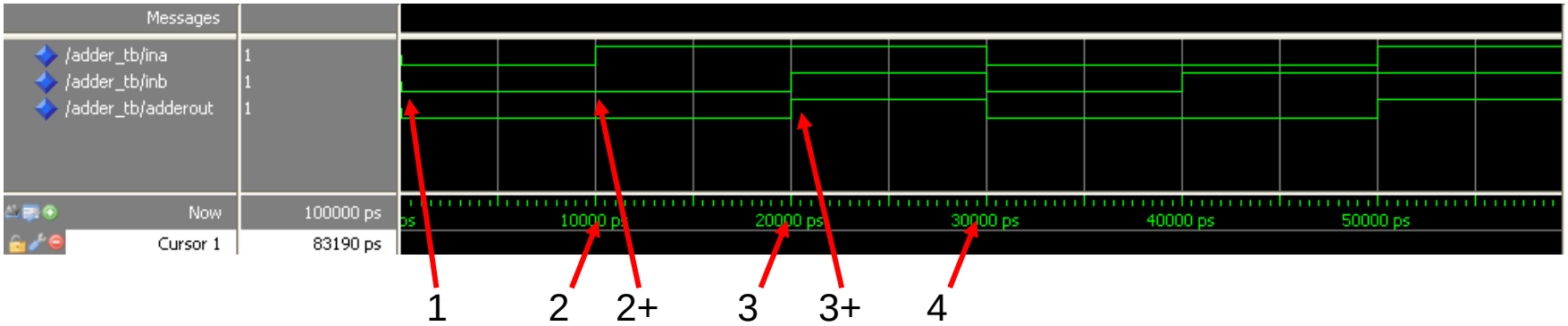


COMBINATIONAL TESTBENCH

```
library ieee;
use ieee.std_logic_1164.all;

entity ander is

  port (
    inA      : in  std_logic;
    inB      : in  std_logic;
    anderOut : out std_logic);

end ander;

architecture ander of ander is

begin  -- ander

  anderOut <= inA and inB;

end ander;
```

```
entity ander_tb is     -- TB has no ports
end ander_tb;

architecture waveform of ander_tb is
  signal inA      : std_logic;
  signal inB      : std_logic;
  signal anderOut : std_logic;

begin  -- waveform
  UUT: entity work.ander
    port map (inA => inA, inB => inB,
      anderOut => anderOut);

  WaveGen_Proc: process
  begin
    inA <= '0';     -- Assign values
    inB <= ’1';
    wait for 10 ns; -- Wait 10 ns
    ...
    wait;
  end process WaveGen_Proc;
end waveform;
```

COMB. TB WAVEFORM




```
 WaveGen_Proc: process
  begin

1)   inA <= '0';
     inB <= '0';
2)   wait for 10 ns;

2+)  inA <= '1';
     inB <= '0';
3)   wait for 10 ns;

3+)  inA <= '1';
     inB <= '1';
     wait for 10 ns;
```

WAIT

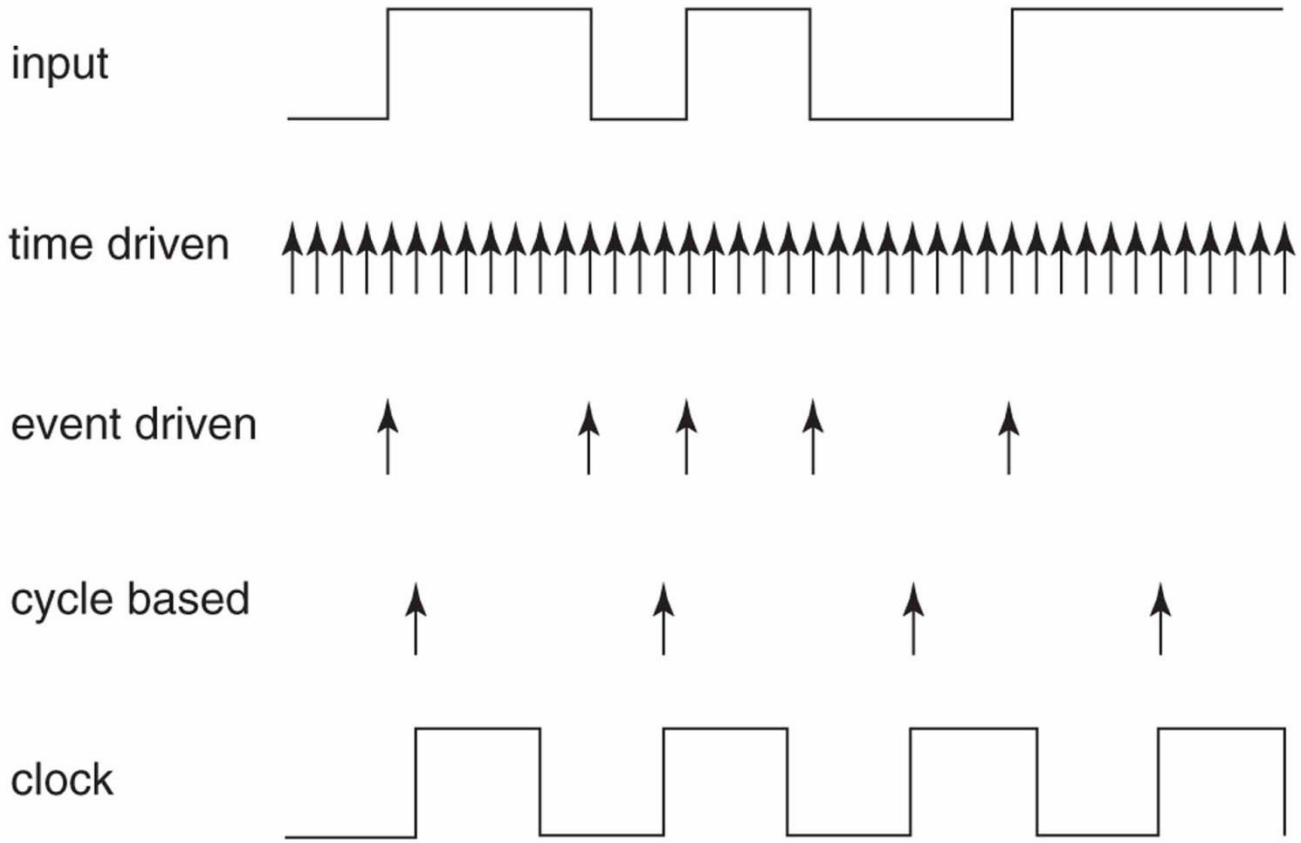
- Wait on <signal>
- Wait until <boolean>
- Wait for <time>
- Wait

- NO Sensitivity List!!!

```
WaveGen_Proc: process
  begin

wait until reset = '1';
...
inB <= '1';
wait for 10 ns;
...
inB <= '0';
wait on anderOut for 10 ns;
...
wait;     -- Wait forever
```

Timeout

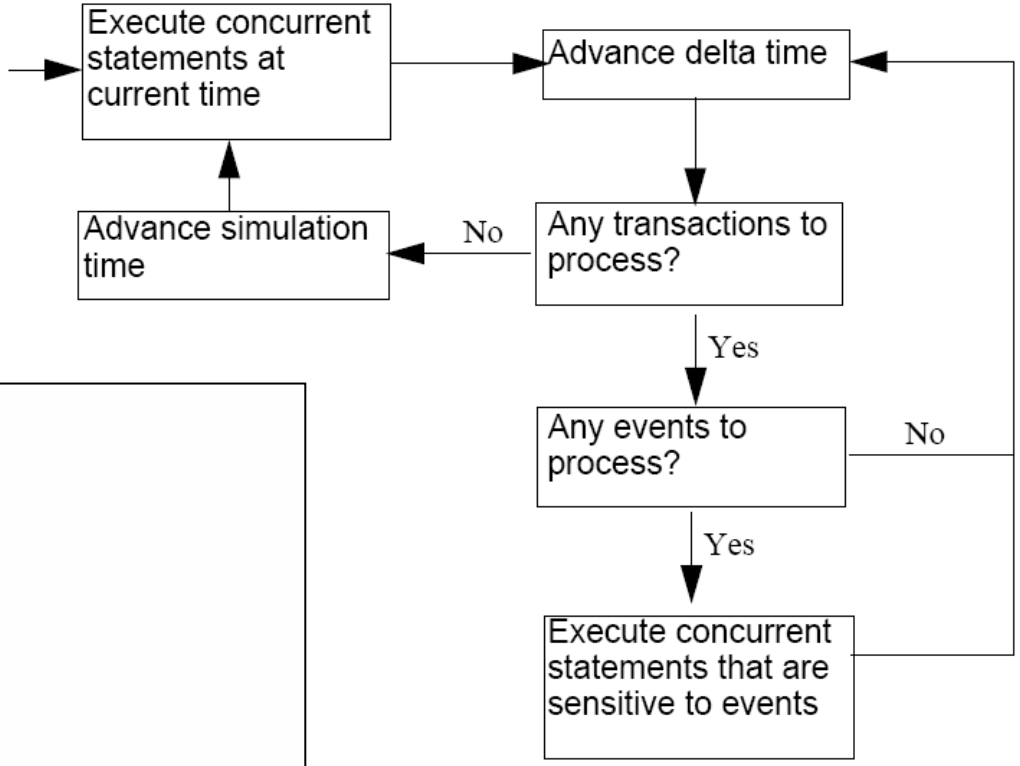
EVENT-DRIVEN SIMULATION

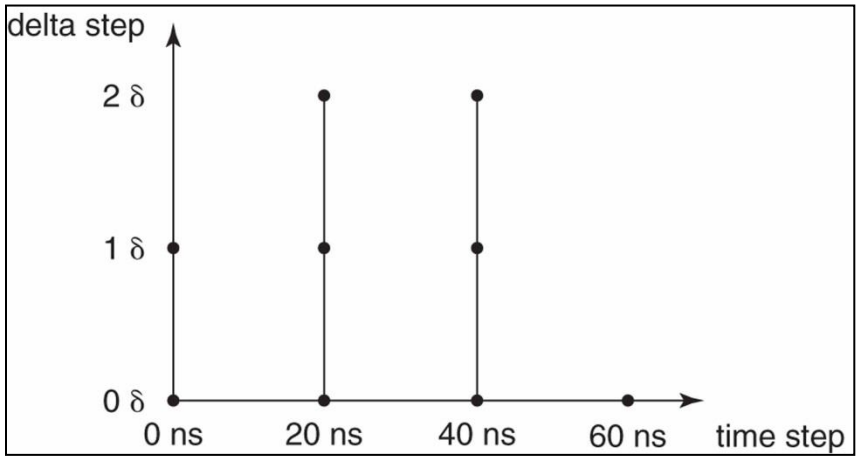


› ModelSim is an event-driven simulator

DELTA DELAYS

Figure 6-1. VHDL Delta Delay Process





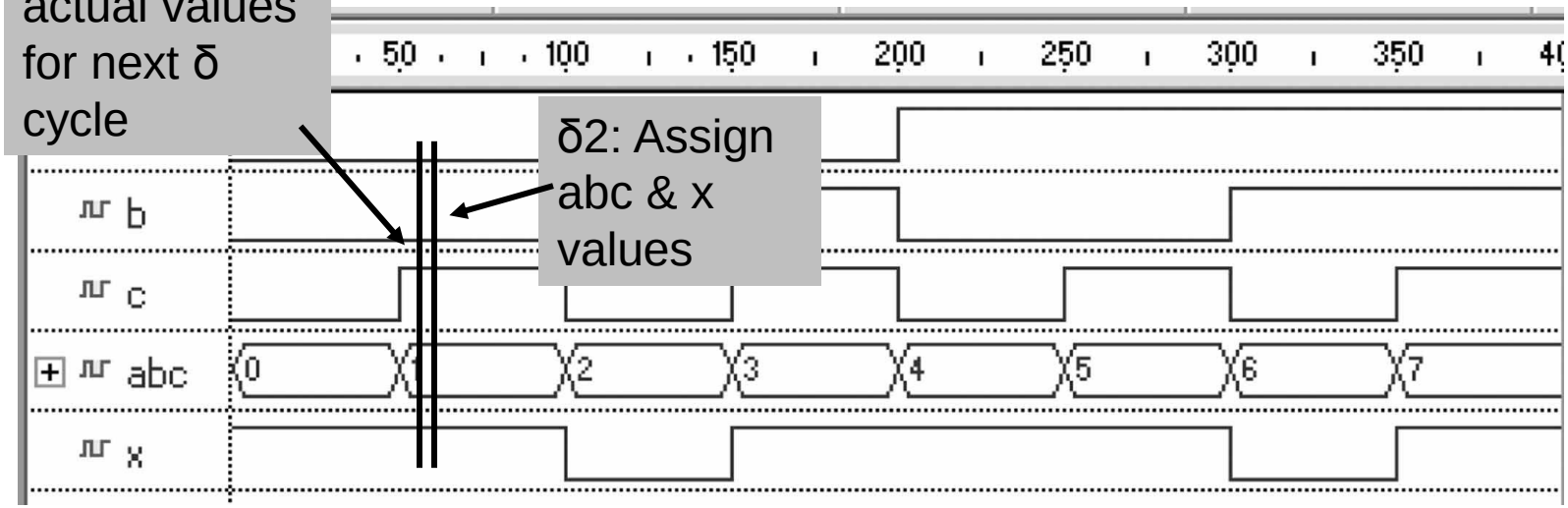
ModelSim
Users Manual

SENSITIVITY LIST

```
tt: process(a, b, c)
begin
  abc <= (a, b, c);
  case abc is
    when "001" | "101" => x <= '0';
    when others => x <= '1';
    ...
```

δ1:Schedule abc update & x update based on actual values for next δ cycle

δ2: Assign abc & x values



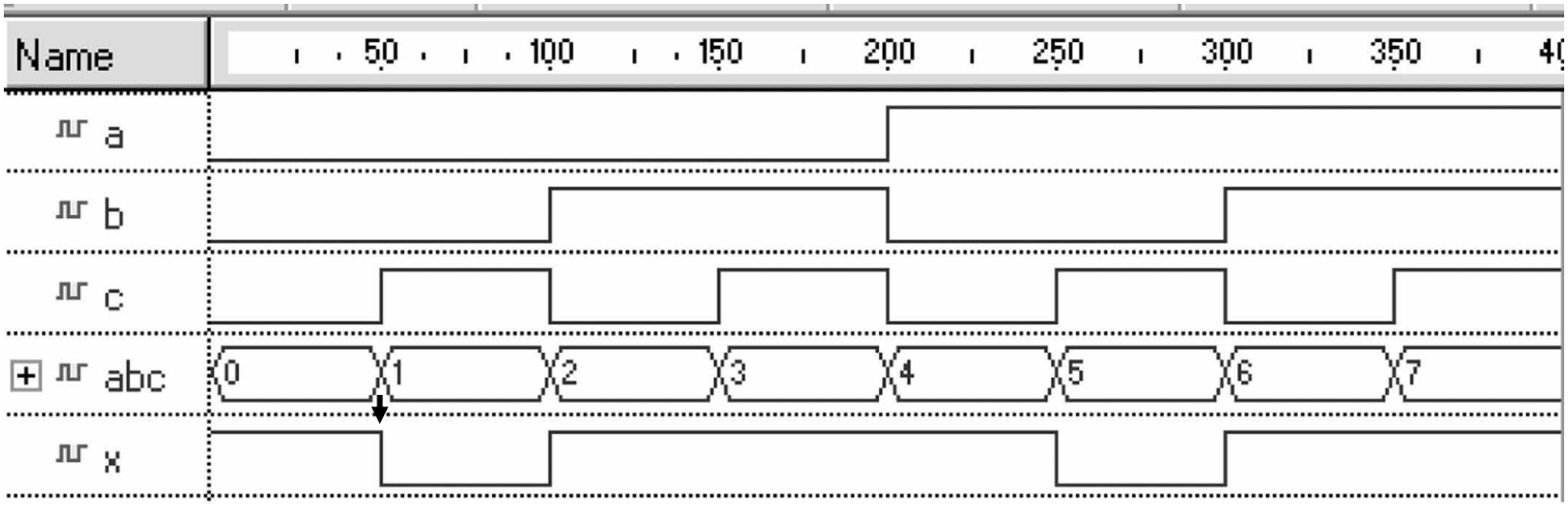
› Why this behavior?

SENSITIVITY LIST CONT.

Abc now behaves as a wire

```
abc <= (a, b, c); -- Concurrent Region
tt: process(abc)
begin
  case abc is
     when "001" | "101" => x <= '0';
     when others => x <= '1';
     ...
```

"case" is now run, when input is changed



SIGNAL ATTRIBUTES

Create New signals

Return Boolean

Return Time

Table 6.11.1
Predefined signal attributes.

Attribute	Result
S'Transaction	Implicit bit signal whose value is changed in each simulation cycle in which a transaction occurs on S (signal S becomes active).
S'Stable(t)	Implicit boolean signal. True when no event has occurred on S for t time units up to the current time, False otherwise.
S'Quiet(t)	Implicit boolean signal. True when no transaction has occurred on S for t time units up to the current time, False otherwise.
S'Delayed(t)	Implicit signal equivalent to S, but delayed t units of time.
S'Event	A boolean value. True if an event has occurred on S in the current simulation cycle, False otherwise.
S'Active	A boolean value. True if a transaction occurred on S in the current simulation cycle, False otherwise.
S'Last_event	Amount of elapsed time since last event on S, if no event has yet occurred it returns TIME'HIGH.
S'Last_active	Amount of time elapsed since last transaction on S, if no transaction has yet occurred it returns TIME'HIGH.
S'Last_value	Previous value of S immediately before last event on S.

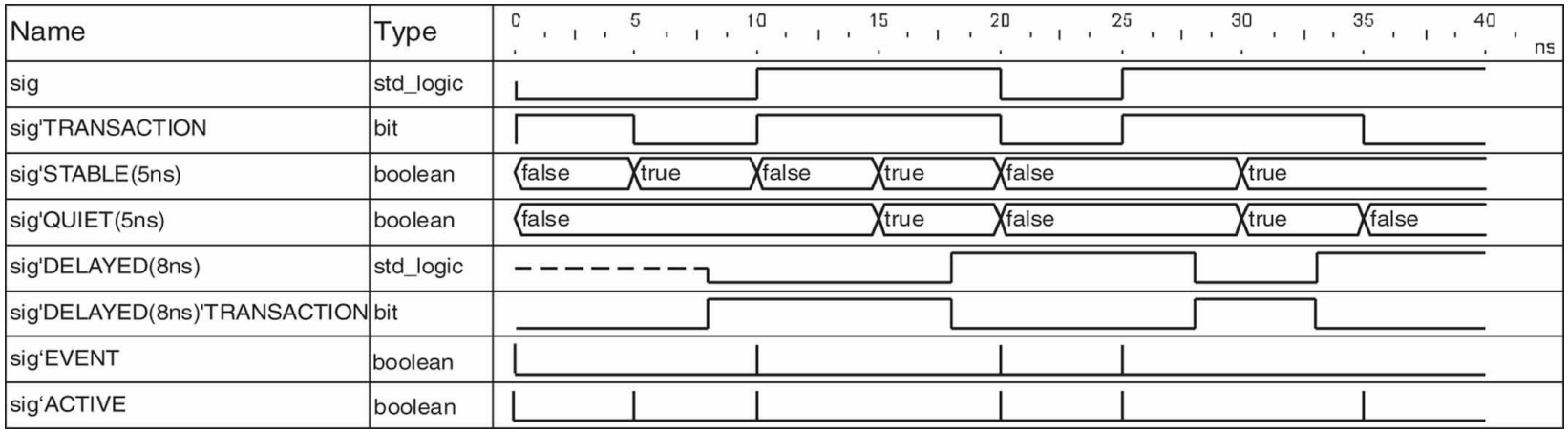
(Cont.)

SIGNAL ATTRIBUTES CONT.

Bus Control

Table 6.11.1 *(Cont.)*

Attribute	Result
S'Driving	True if the process is driving S or every element of a composite S, or False if the current value of the driver for S or any element of S in the process is determined by the null transaction.
S'Driving_value	Current value of the driver for S in the process containing the assignment statement to S.



ASSERTIONS

› "In computer programming, an assertion is a predicate (i.e., a true–false statement) placed in a program to indicate that the developer thinks that the predicate is always true at that place" (Wikipedia)

assert <*true condition*> report <*err string*> severity <*severity level*>;

- <true condition> The expected true boolean expression
- <err string> String returned to simulator std io if condition is false

ASSERTIONS CONT

- <Severity Level>:
- ***Failure***: Errors in the model itself (e.g. if a statement believed to be non-executables is actually executed);
- ***Error***: Timing violations and invalid data affecting the state of the model, including illegal combinations of mode signals and of control signals (e.g. unknown data on a mode input or too short reset time);
- ***Warning***: Timing violations and invalid data not affecting the state, but which could affect the simulation behavior of the model (e.g. if data to be sent out from an interface is invalid);
- ***Note***: Essential information that is not classified in the other severity levels, such as reporting from which text file data is read, which testbench is executed, if an event is detected on an input signal whose function has not been implemented (e.g. activation of production test) etc.
  (ESA VHDL Modeling Guidelines)

ASSERTION EXAMPLE

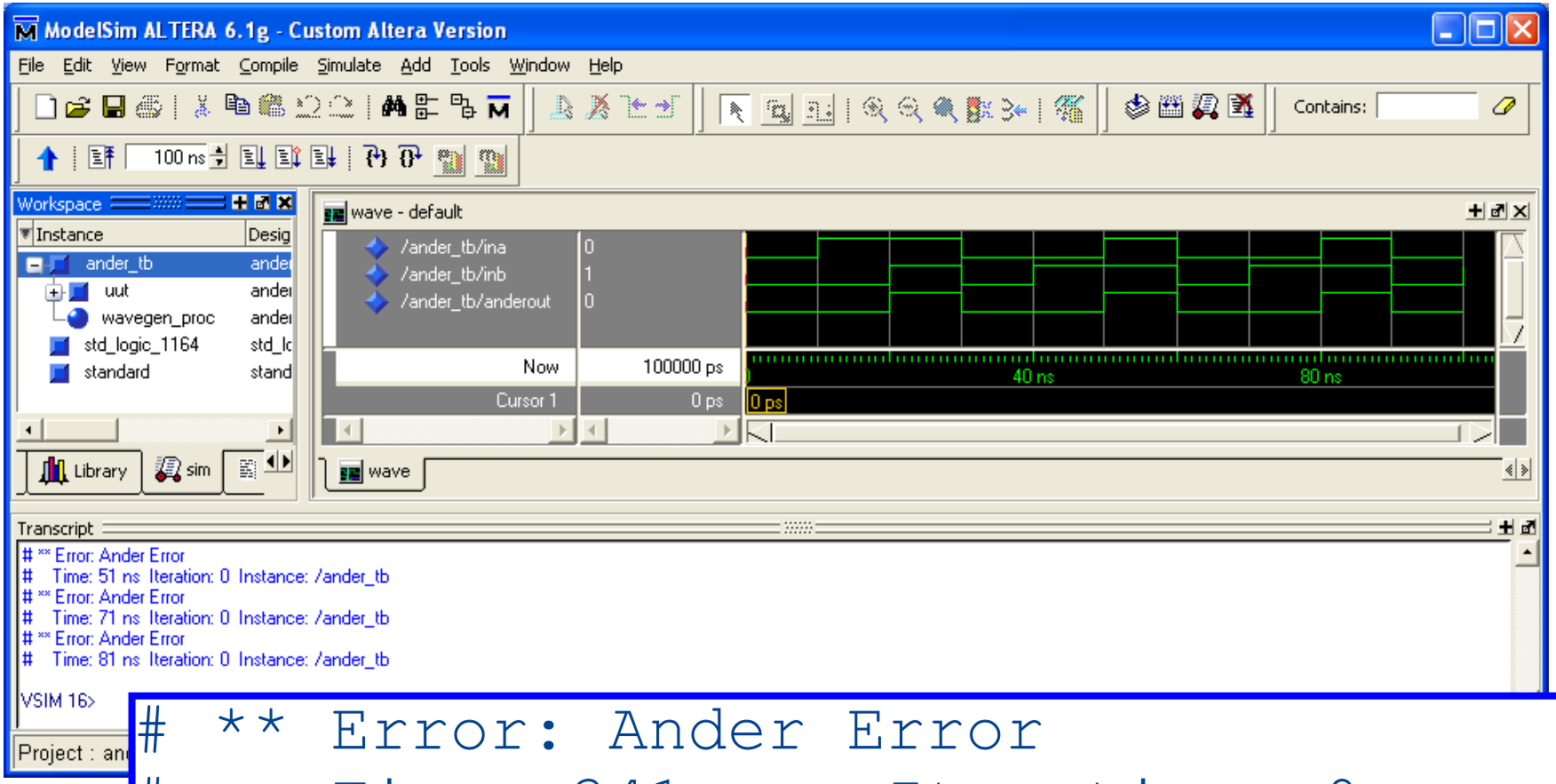
```
WaveGen_Proc: process
 begin
   inA <= '0';
   inB <= '0';
   wait for 10 ns;

   inA <= '1';
   inB <= '1';
   wait for 10 ns;
   assert anderOut = '1'
    report "And Error" severity error;

   wait;
 end process WaveGen_Proc;
```

Expected Behaviour

Response if assertion is NOT true

THINK: Reverse IF statement, ex. if !(anderOut = 1)

ASSERTIONS OUTPUT

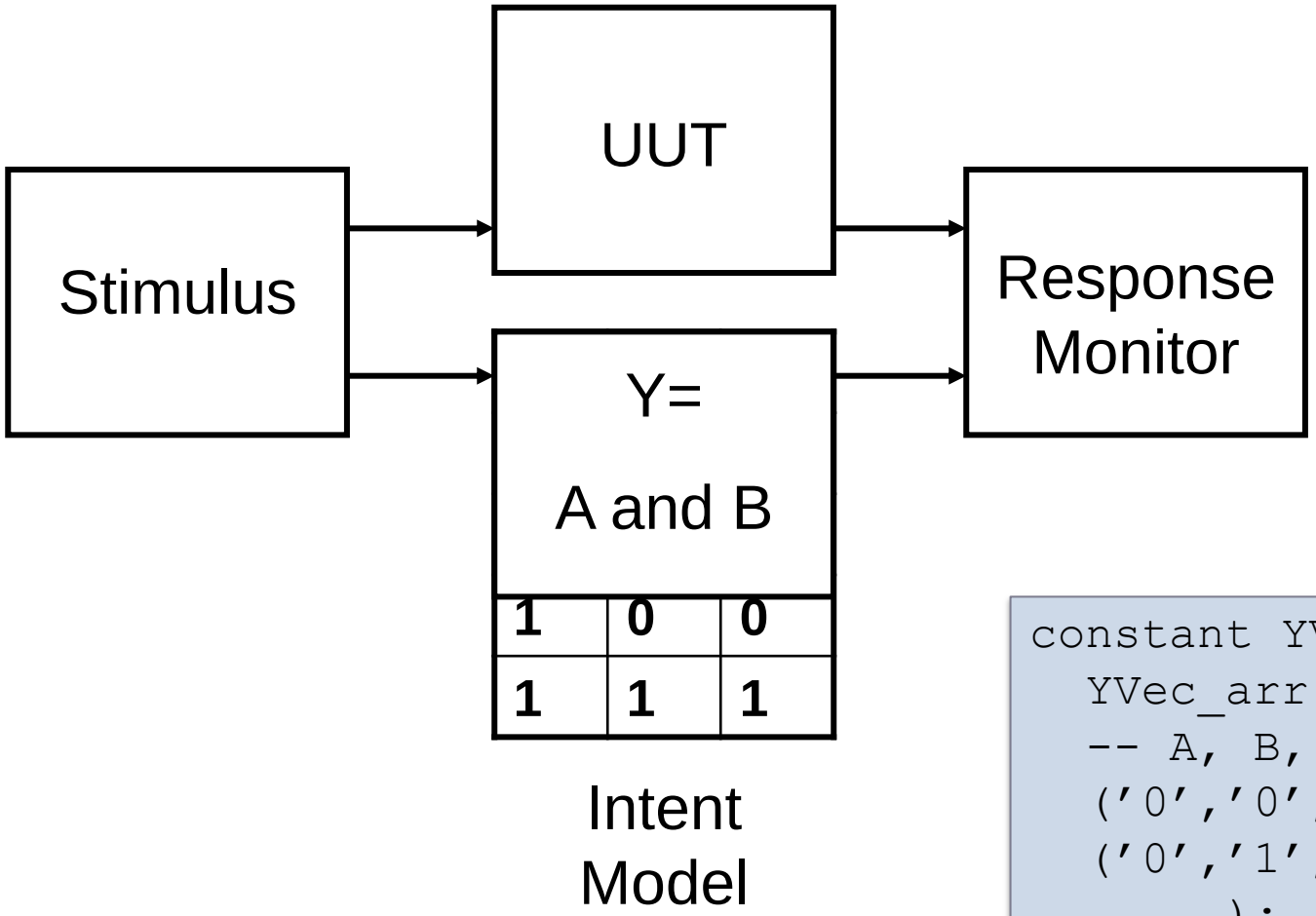



```
# ** Error: Ander Error
#    Time: 341 ns  Iteration: 0
 Instance: /anderclocked_tb
```

TEST BENCH TYPES

- Stimuli -> Waveform
- Self Checking, using look-up tables
- Self Checking, using behavioural models

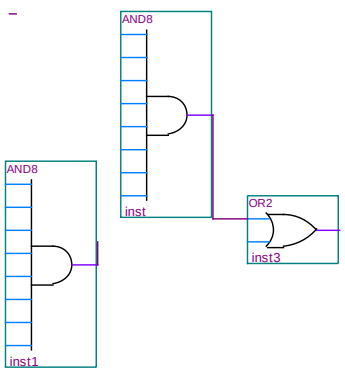
SELF-CHECKING TESTBENCH

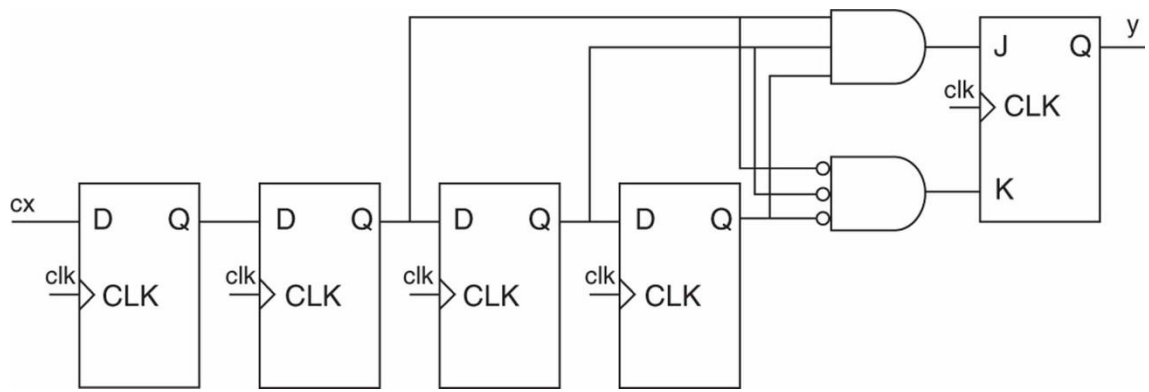



```
constant YVec :
  YVec_arr :=(
  -- A, B, Y
  ('0','0','0'),
  ('0','1','0'),
    ...);
```

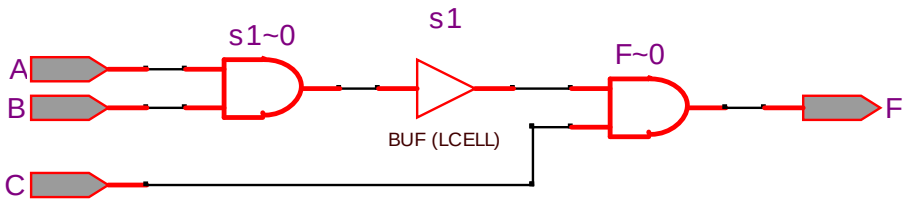
EXHAUSTIVE VERIFICATION

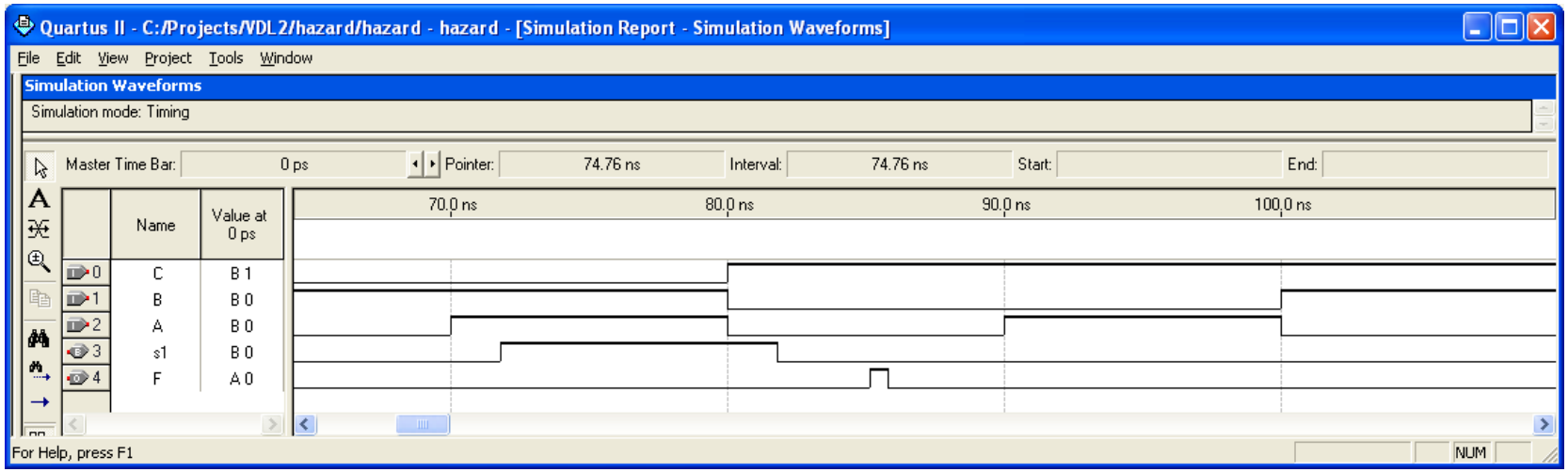
- Possible for combinatorial designs, since output only depends on input.
- Number of possible combinations increase by 2^n
- Impractical for sequential designs, since the number of input combinations is enormous





HAZARDS





›Static- & dynamic Hazards due to signal delays
›Solution: Karnaugh Maps or Synchronous designs

CODE- & BRANCH COVERAGE

Instance Coverage

Instance	Design Unit	Design Unit Type	Stmt Count	Stmt Hits	Stmt %	Stmt Graph
/test_delta/chi...	mode_two...	Module	30	30	100	
/test_delta/chi...	pre_processor	Module	784	616	78	
/test_delta/chi...	micro	Module	26	25	96	
/test_delta	test_delta(rtl1)	Architecture	763	420	55	
/test_delta/chip	delta(rtl)	Architecture	57	55	96	
/test_delta/chi...	tx_process(rtl)	Architecture	520	415	79	
/test_delta/chi...	post_proce...	Architecture	203	190	93	
/test_delta/chi...	arbitrator(rtl)	Architecture	236	211	89	
/test_delta/chi...	fs_add_mux...	Architecture	4	4	100	
/test_delta/chi...	fifocell(rtl)	Architecture	17	17	100	
/test_delta/chi...	fifocell(rtl)	Architecture	17	17	100	

› Code Coverage gives a ratio of how many statements has been covered by the testbench

BRANCH COVERAGE

source - testdel.vhd

File Edit View Tools Window

Hits | BC | In # | C:/CodeCoverage/testdel.vhd

```
285 wait;
286 end process;
287
288 -- Transmit data input
289 LOAD_DATA: process
290 file F:TEXT is in "transmit_data";
291 variable L: LINE ;
292 variable tempstore : STD_LOGIC_VECTOR(7 downto 0);
293 begin
294   wait until FALLING_EDGE (TXC);
295     if TXDATA_S = TRUE then
296       if TXCOUNTER = 7 then
297         TXCOUNTER <= 0 ;
298         if not ENDFILE (F) then
```

testdel.vhd

Ln: 286 Col: 0 By inst

› CC does not cover branches, a separate measure, BC is used.
› BC checks if all branches has been covered by the test

MODELSIM

- ModelSim is a standalone simulator
- Produced by Mentor Graphics
- Altera & Xilinx offers a reduced version, distributed freely with their respective IDEs
- Simulates either pre- or post-synthesis (functional or timing)
- Can be controlled via GUI or command prompt
- Supports TCL scripts

SIMULATION WINDOW