Universidad
Carlos III de Madrid

Circuit Simulation of digital circuits with VHDL

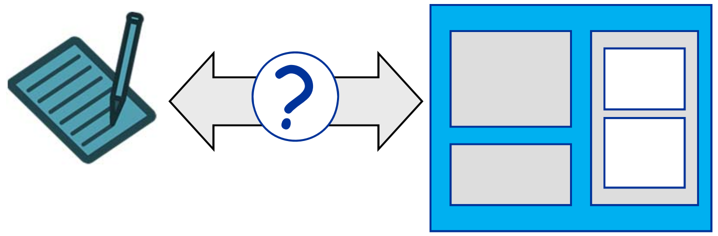
TEST BENCHES AND SIMULATION TOOLS

Outline

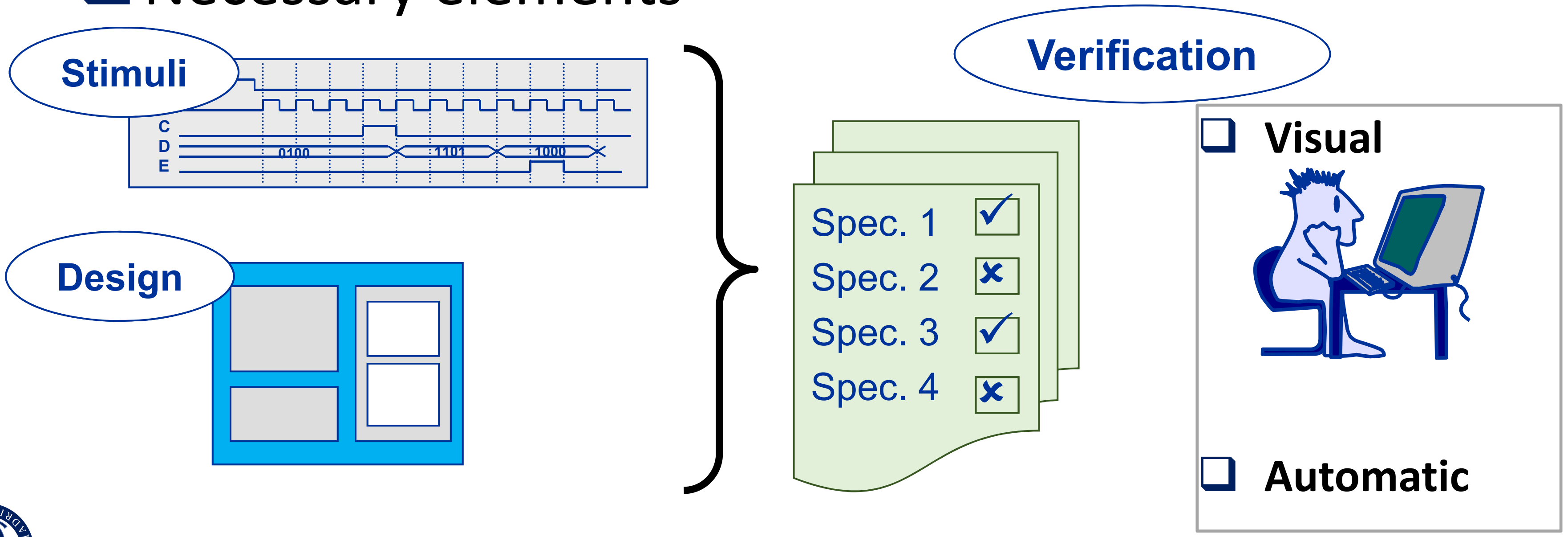
- ❑ Digital circuit simulation
- ❑ Simulating with VHDL
- ❑ Test bench design
  - ❖ Generating stimuli
  - ❖ Automatic checking
- ❑ Commercial tools: Modelsim

Digital circuit simulation

- ❑ Functional validation: Verify that the design behaves according to the specifications.

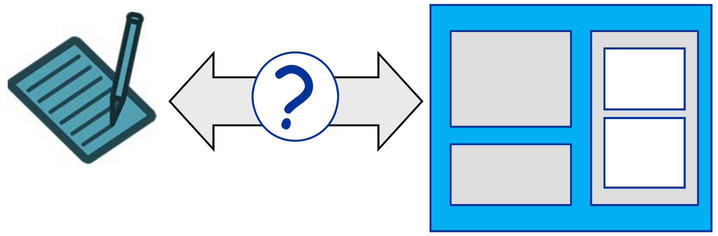


- ❑ Necessary elements

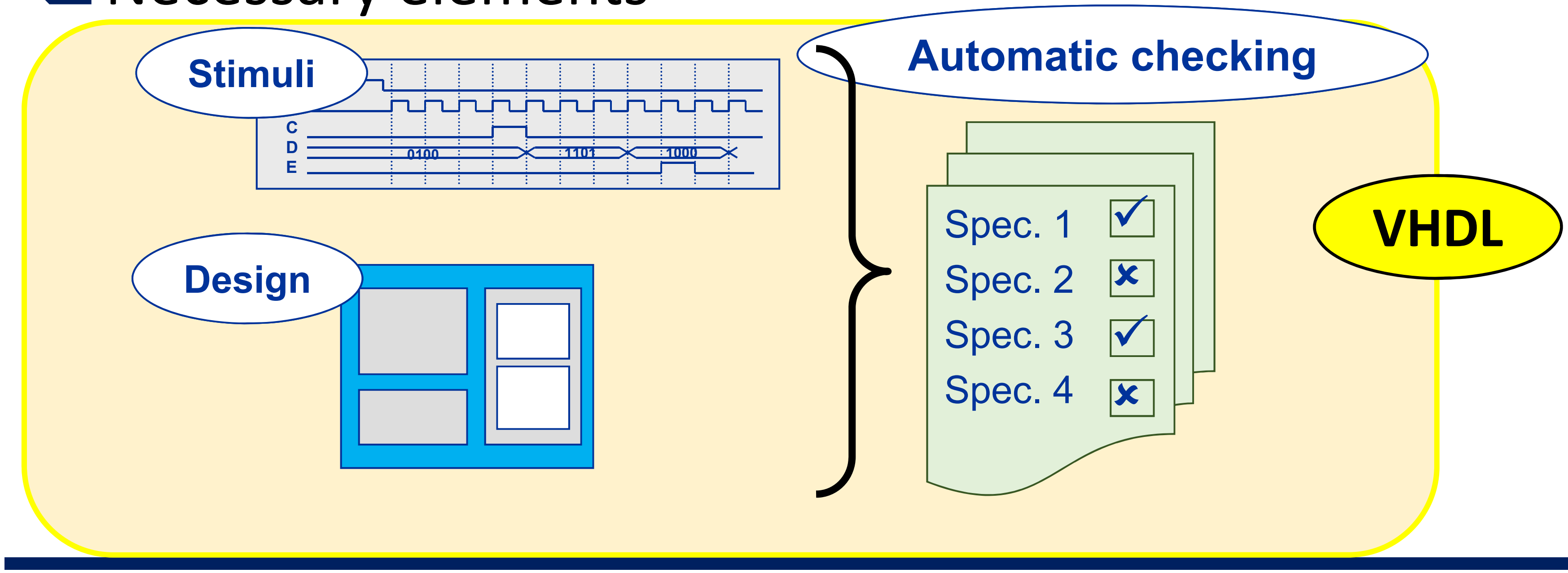


Digital circuit simulation

- Functional validation: Verify that the design behaves according to the specifications.



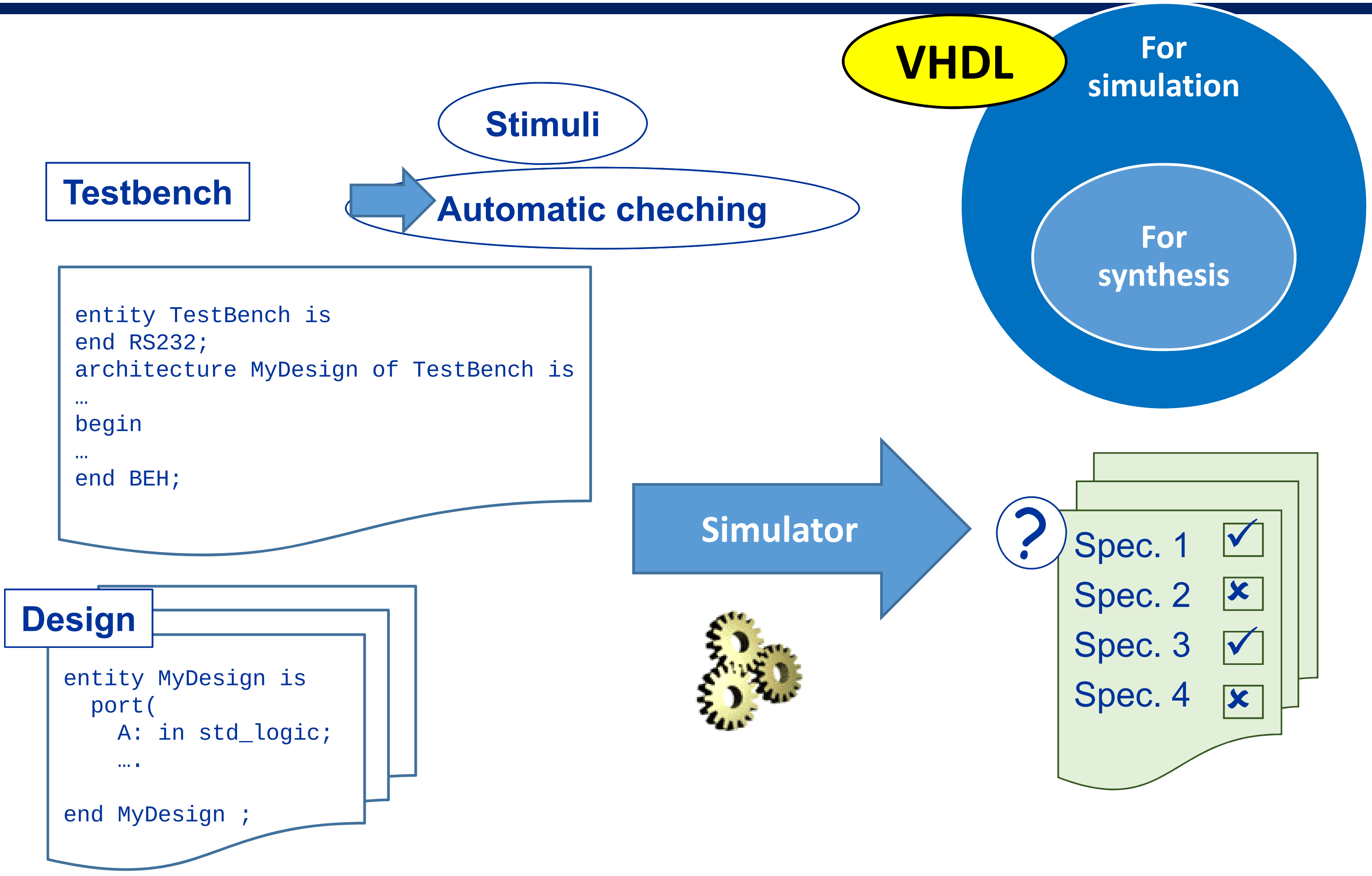
- Necessary elements



Digital circuit simulation: General issues

- Generate a set of inputs (stimuli) as much complete as possible
  - Checking all the specifications to meet by the circuit.
  - In case of FSMs (Finite State Machines), the circuit must pass through every possible state and cover any possible transition.
- Asynchronous initialization of the full system at the beginning of the circuit simulation
- Inputs should change at **inactive** clock edge.

Digital circuit simulation with VHDL



```
entity TestBench is
end RS232;
architecture MyDesign of TestBench is
…
begin
…
end BEH;
```

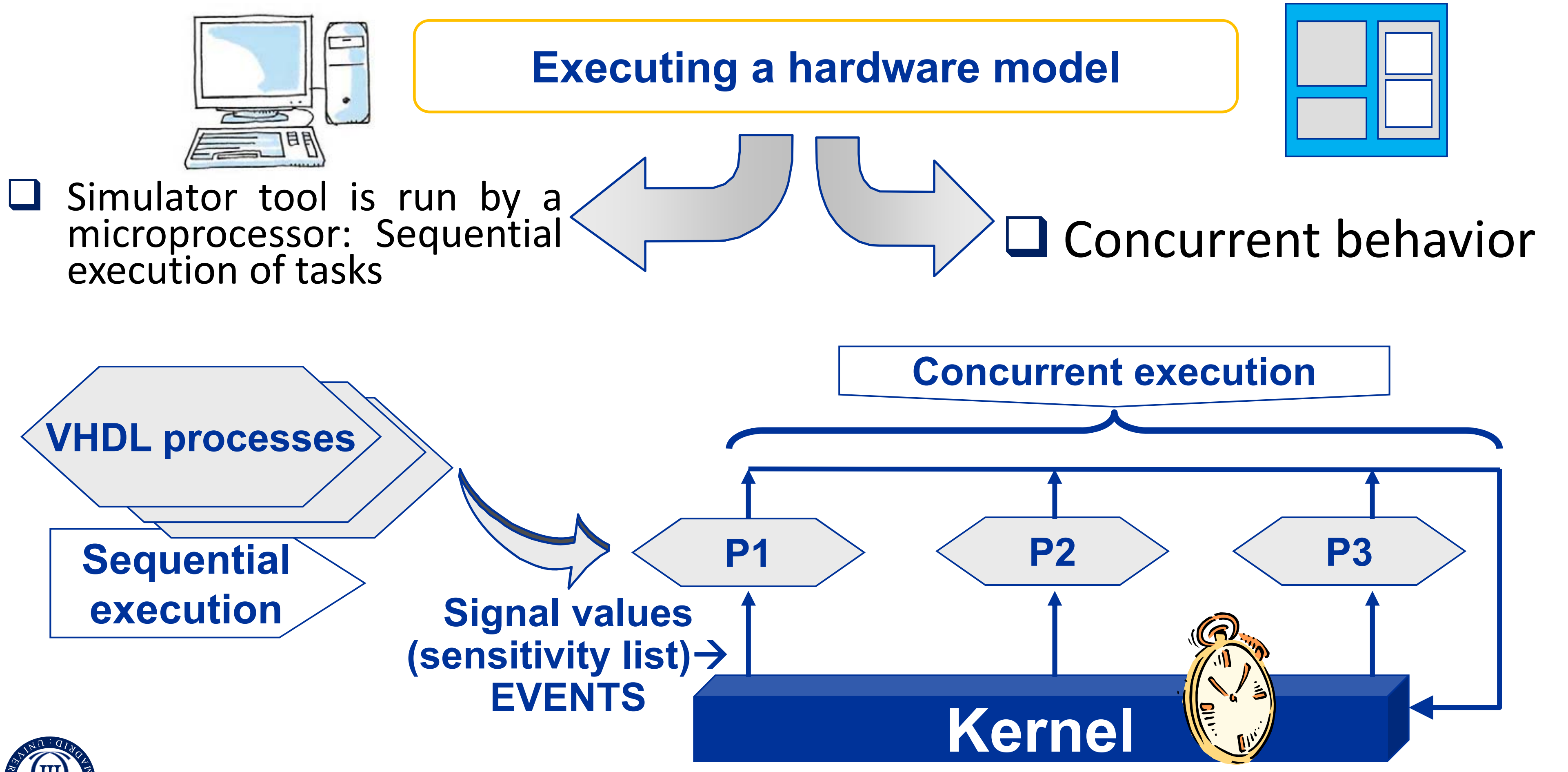
```
entity MyDesign is
  port(
    A: in std_logic;
    ….

end MyDesign ;
```

Digital circuit simulation with VHDL

- Advantages of using VHDL for simulation
  - VHDL was devised for simulating and specifying designs.
  - It supports automatic checking and sending information to the designer during the simulation (interactive checking)
  - It allows designers to model external interfaces at high level: modelling the external environment
  - Using files
  - Using delays and timings

Digital circuit simulation with VHDL

- How does the digital simulator work (HDL)?: **Event simulation**



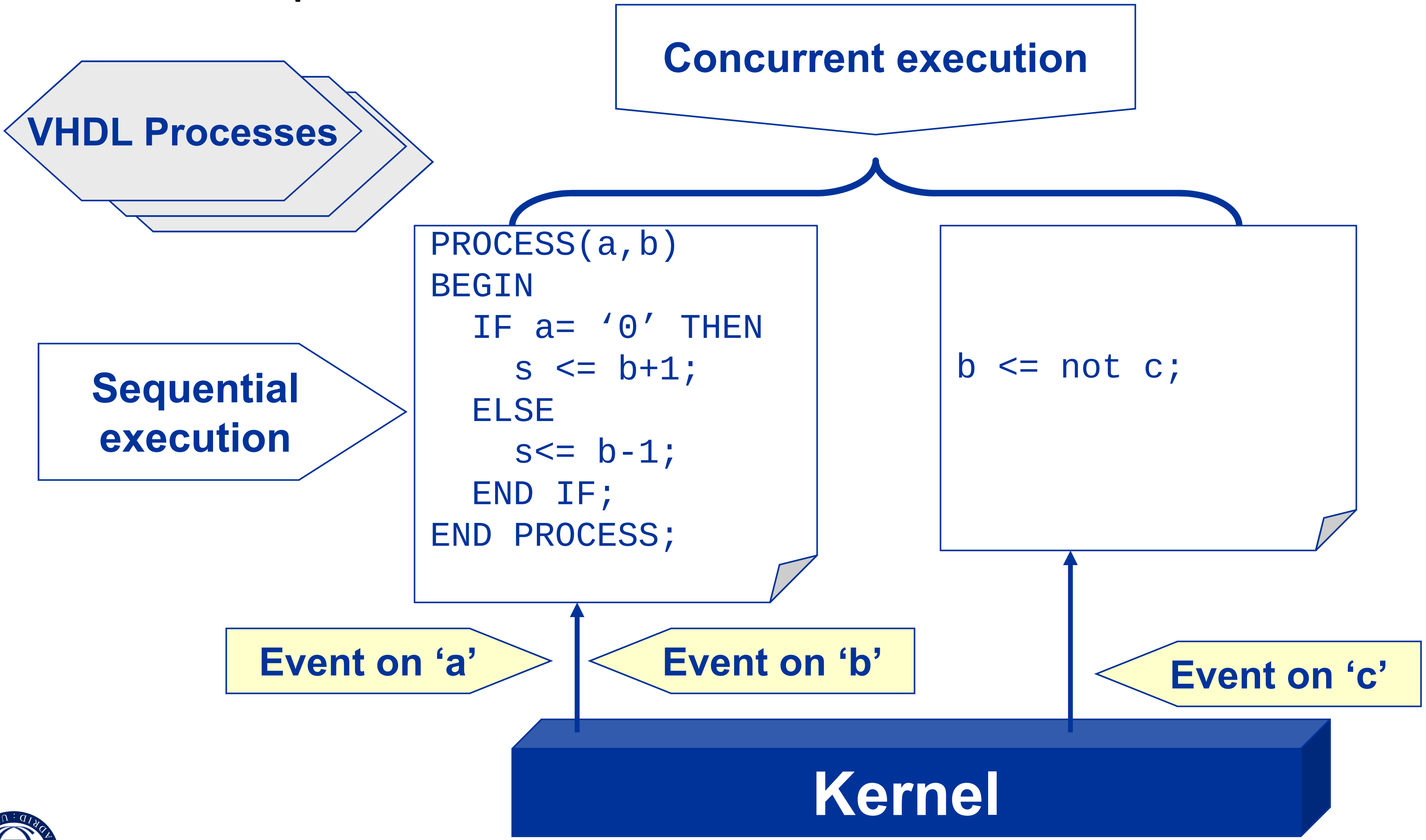
Executing a hardware model
Simulator tool is run by a microprocessor: Sequential execution of tasks
Concurrent behavior
VHDL processes
Sequential execution
Concurrent execution
Signal values (sensitivity list)→ EVENTS
P1
P2
P3
Kernel

Digital circuit simulation with VHDL

- How does the digital simulator work (HDL)?: **Event-driven simulation**
  - When there is a change (event) in any of the signals included in the sensitivity list, the simulator kernel generates a list with pairs (event, instant).
  - For a given simulation instant, the kernel is in charge of:
    1. Accessing the event list and executing the *process* statements with some pending event for the current instant.
    2. Updating the event list with the new events generated during the process execution.
    3. If some of the new events have been generated for the current instant, go again to 1. Otherwise, time simulation is incremented until the next instant with active events.

Digital circuit simulation with VHDL

- Example

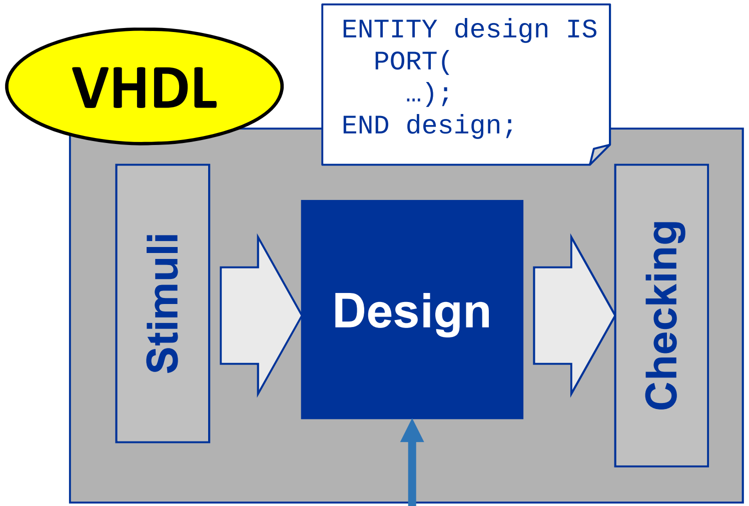


```
PROCESS(a,b)
BEGIN
  IF a= ‘0’ THEN
    s <= b+1;
  ELSE
    s<= b-1;
  END IF;
END PROCESS;
```

```
b <= not c;
```

Test bench design

❑ Test bench: structure



It is implemented as a component, as in hierarchical designs

```
ENTITY tb is
END tb;
ARCHITECTURE sim OF tb is
  COMPONENT design IS
    PORT(
      …);
  END COMPONENT;

SIGNAL s1 : std_logic;
  …
BEGIN

  D: design
  PORT MAP(
      …);
  …

  -- Stimuli generation!!!
  -- Checking!!!
END sim;
```

Empty entity

Component declaration

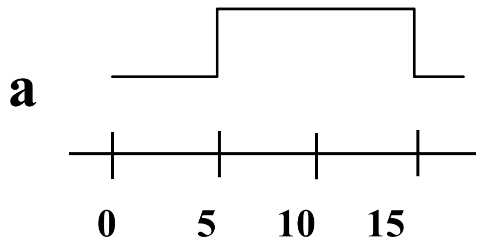
Declaration of signals and constants:
- I/O connections
- …

Component (instance and usage)

Stimuli

Automatic checking

Waveform generation using concurrent statements



```
a <= ‘0’, ‘1’ AFTER 5 NS, ‘0’ AFTER 15 NS;
```

- Concurrent assignment of a waveform
  - Sequence of clauses *<value> AFTER <delay>*
  - Delays are absolute (counted from the execution of the statement, typically at the beginning of the simulation)

WAIT statement

- Used to generate waveforms in sequential style (within a process)
- A WAIT statement suspends the execution of a process until a condition is met

signals	`WAIT ON a;`	Wait until an event happens on a
time	`WAIT FOR 10 ns;`	Wait for 10 ns
conditions	`WAIT UNTIL c1;`	Wait until the condition is true
	`WAIT;`	Never resumes

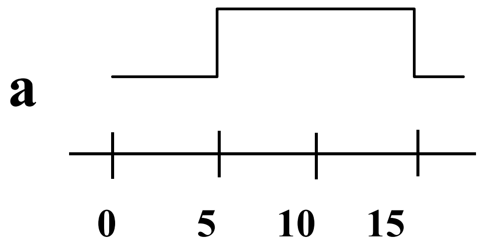
Two types of processes

PROCESSES WITHOUT SENSITIVITY LIST	PROCESSES WITH WAIT STATEMENTS
❖ Execution is suspended at the end of the process ❖ Execution resumes when an event happens in the sensitivity list ❖ WAIT statements are not allowed ❖ They can be synthesized	❖ Execution is cyclic and is only suspended when a WAIT statement is reached (there must be at least one WAIT statement) ❖ Sensitivity list is not allowed ❖ They cannot be synthesized

```
PROCESS (a, b)
BEGIN
        s <= a AND b;
END PROCESS;
```

```
PROCESS
BEGIN
        s <= a AND b;
        WAIT ON a, b;
END PROCESS;
```

Waveform generation using sequential statements



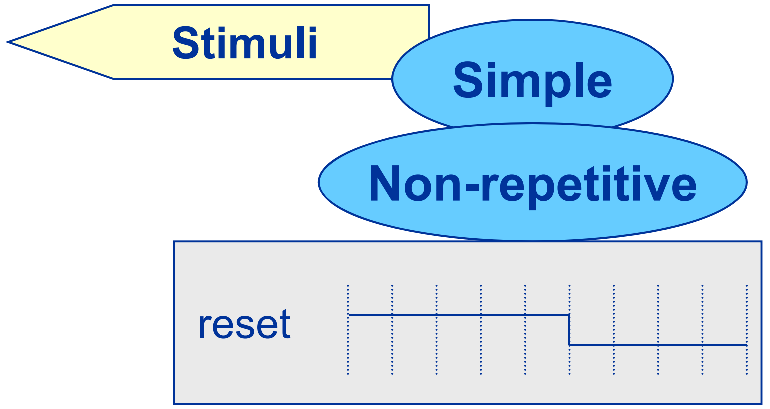
```
PROCESS
BEGIN
        a <= ‘0’;
        WAIT FOR 5 NS;
        a <= ‘1;
        WAIT FOR 10 NS;
        a <= ‘0’;
        WAIT;
END PROCESS;
```

- Sequential assignment of a waveform
  - Assignment followed by WAIT statement
  - Wait times are always relative to the execution time!
  - If a WAIT statement is not included at the end, the waveform is periodic

Stimuli generation: Reset

Sequential style

```
-- Stimuli generation
PROCESS
BEGIN
        reset <= '1';
        WAIT FOR 50 NS;
        reset <= '0';
        WAIT;
END PROCESS;
```



Concurrent style

```
-- Stimuli generation
reset <= '1', '0' after 50 ns;
```

Stimuli generation: clock

- Asymmetric

```
-- Stimuli generation
PROCESS
BEGIN
        clk <= '1';
        WAIT FOR 10 NS;
        clk <= '0';
        WAIT FOR 40 NS;
END PROCESS;
```

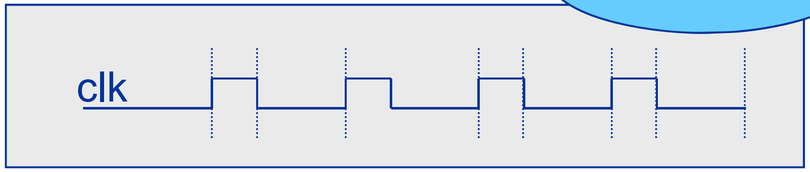
- Symmetric

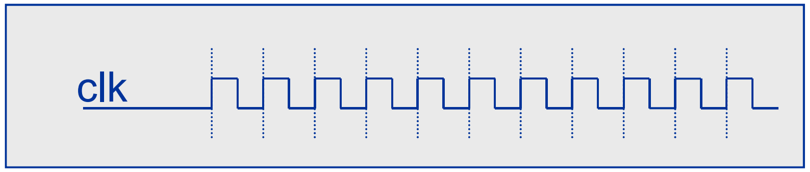
```
-- Stimuli generation
PROCESS
BEGIN
        clk <= NOT clk;
        WAIT FOR 10 NS;
END PROCESS;
```

stimuli

Simple

Periodic



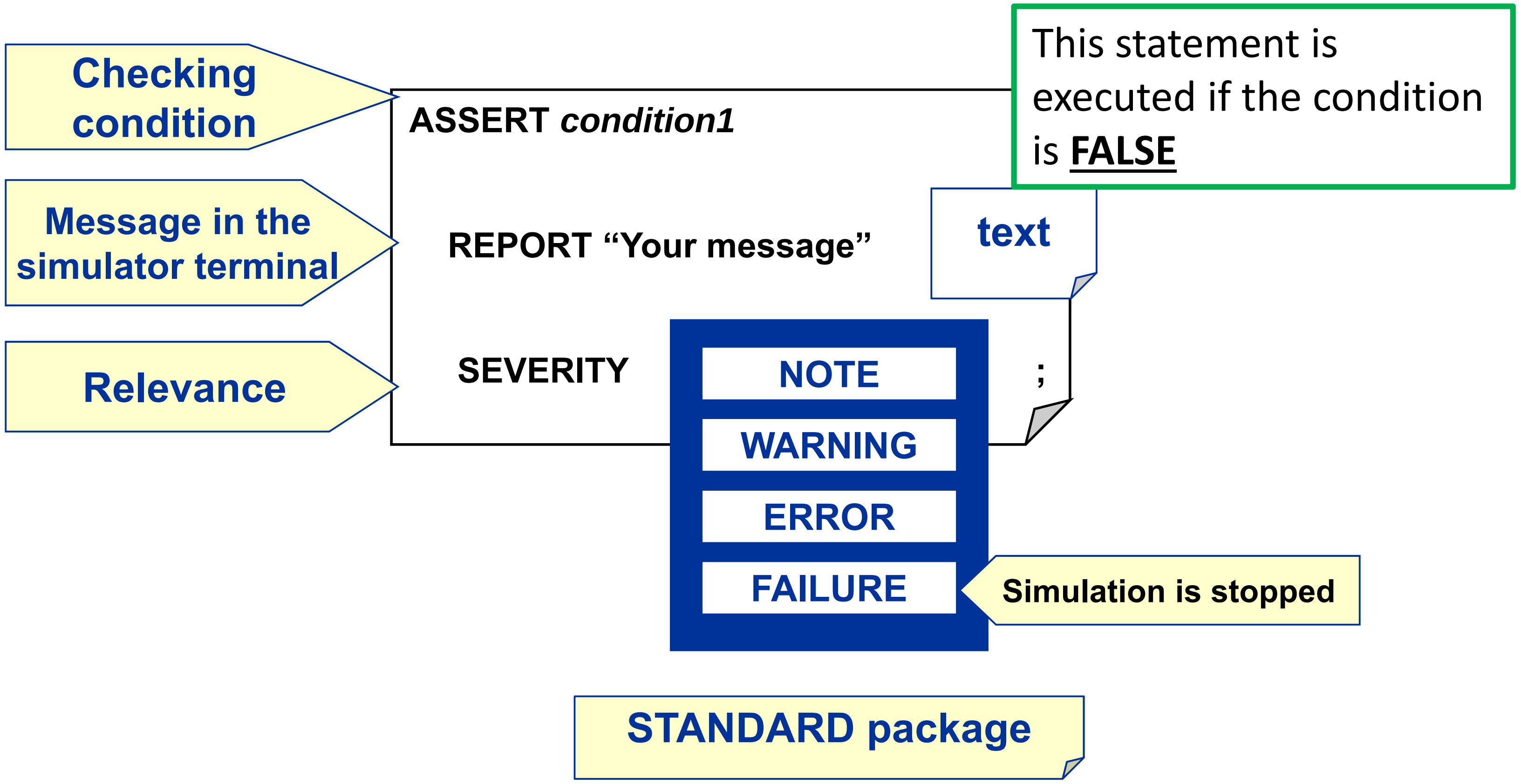


```
-- Initializing the signal is required!
signal clk: std_logic := '0';
```

```
-- Concurrent
clk <= NOT clk AFTER 10 NS;
```

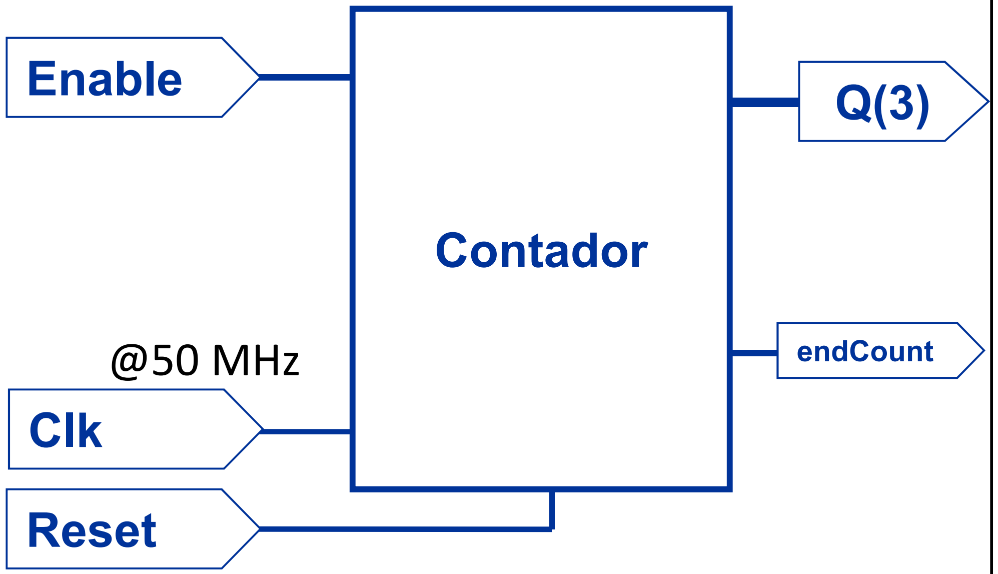
Automatic checking

- Testbench: Automatic checking



Digital circuit simulation with VHDL

- Testbench: Example counter from 0 to 5



```
entity counter is
 port(
  clk         : in std_logic;
  reset       : in std_logic;
  enable      : in std_logic;
  endCount : out std_logic;
  Q           : out unsigned(2 downto 0)
 );
end contador ;
```

Note: This example shows a simple testbench, where all the specifications are not checked.

Digital circuit simulation with VHDL

❑ Testbench: Example counter from 0 to 5

```
ENTITY tb is

END tb;

ARCHITECTURE sim OF tb is
--Component declaration
  component counter
  port(
    clk         : in std_logic;
    reset       : in std_logic;
    enable      : in std_logic;
    endCount : out std_logic;
    Q           : out unsigned(2 downto 0)
  );
end component;
--Signal and constant declarations
signal clk      : std_logic:= '0';
signal reset   : std_logic;
signal enable: std_logic;
signal endCount : std_logic;
signal Q         : unsigned(2 downto 0);
constant T     : time := 20 ns;
```

```
begin
--Component instantation
  MAP_CUT: counter
  port map(
    clk          => clk,
    reset        => reset,
    enable      => enable,
    endCount => endCount ,
    Q             => Q
  );
--Stimuli generation and automatic checkings
clk <= not clk after T/2;

process
begin
  reset    <= '1'; --At the begining
  enable <= '0';
  wait for T;
  reset    <= '0'; --Disable reset
  wait for T; --Count has not started yet
  enable <= '1';
  wait until endCount = '1'; --count has finished
  assert Q = to_unsigned(5,3)
    report "Error: Last value is not 5"
    severity error;
  assert false
    report "End of simulation"
    severity failure;
end process;
```

Stop simulation

Commercial tools

- Modelsim (Mentor Graphics®) → www.mentor.com/
- VCS (Synopsys®) → www.synopsys.com
- Incisive Enterprise Simulator (Cadence®) → www.cadence.com
- ISIM (Xilinx™) → www.xilinx.com

Universidad
Carlos III de Madrid

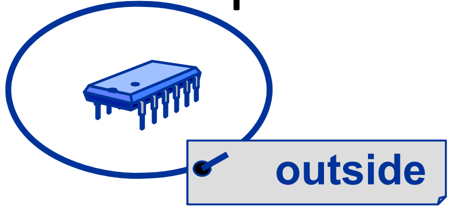
Circuit Simulation of digital circuits with VHDL

SIMULATION MODELS

Outline

Introduction

- Description or simulation environments in VHDL



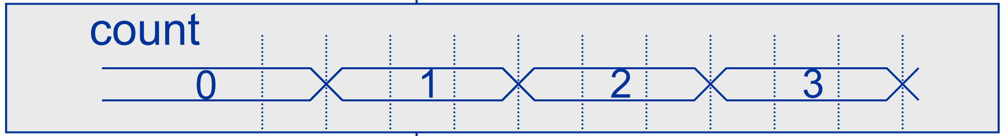
TB

```
entity tb is
end tb;
architecture sim of tb is
  component RAM is…
  component CPU is…

begin
…
```

Models of the circuit environment

Stimuli generation using a counter

```
ENTITY tb is
END tb;

ARCHITECTURE mixed OF tb is
  SIGNAL count: integer range 0 to 3 := 0;
BEGIN
…
  -- STIMULI!!!
  PROCESS
  BEGIN
    IF count = 3 THEN
      count = 0;
    ELSE
      count <= count + 1;
    END IF;
    WAIT FOR 150 ns;
  END PROCESS;
END mixed;
```



Stimuli generation using tables

```
…
ARCHITECTURE a OF tb IS
        TYPE TableType IS array (0 TO 15) of std_logic_vector(3 downto 0);
        CONSTANT Table: TableType :=
                (“0011”, “0110”, “0111”, “1011”, “1110”, “1111”, “1001”, “0111”,
                 “0101”, “1010”, “0111”, “0011”, “0000”, “1110”, “0110”, “0100”);
        SIGNAL index: INTEGER := 0;
                SIGNAL value: std_logic_vector(3 downto 0);
…
BEGIN
…
        PROCESS
        BEGIN
                value <= Table(index);
                IF index = 15 THEN
                        index := 0;
                ELSE
                        index <= index + 1;
                END IF;
                WAIT UNTIL clk = ‘0’;
        END PROCESS;
…
END a;
```

I/O Files

- A FILE object defines an identifier for a system file
- FILE objects cannot be assigned, but can be read from and written to by using special subprograms:
  - Procedure READ
  - Procedure WRITE
  - Function ENDFILE
- The package TEXTIO, that is available in VHDL, defines the necessary types, procedures and functions to read from and write to ASCII files:
  - Procedure READLINE
  - Procedure WRITELINE

Using text files for data input

```
USE STD.TEXTIO.ALL;
…..
PROCESS
        FILE input_file: TEXT OPEN READ_MODE IS “data.txt";
        VARIABLE lin: LINE;
        VARIABLE val_char: CHARACTER;
        VARIABLE val_int: INTEGER;
BEGIN
-- Read a CHARACTER from the console
        READLINE(INPUT, lin);
        READ(lin, val_char);
-- Read an integer for an input file
        IF not(ENDFILE(input_file)) THEN
                READLINE(input_file, lin);
                READ(lin, val_int);
        END IF;
        s <= val_int;
        WAIT UNTIL clk = '1';
END PROCESS;
```

Include package TEXTIO

File declaration

Line declaration

Read line

Read datum from line

Check if the file contains data

Read values are variables

Read every clock cycle

Using text files for data output

```
USE STD.TEXTIO.ALL;
…..
PROCESS
        FILE output_file: TEXT OPEN WRITE_MODE IS “data.txt";
        VARIABLE lin: LINE;
        VARIABLE val: INTEGER;
        CONSTANT label: STRING := "DATUM";
        CONSTANT separator: STRING := " : ";
        VARIABLE index: INTEGER := 0;
BEGIN
        WRITE(lin, label);
        index := index + 1;
        WRITE(lin, index);
        WRITE(lin, separator);
        WRITE(lin, val);
        WRITELINE(output_file, lin);

        WAIT UNTIL clk = '1';
END PROCESS;
```

Include package TEXTIO

File declaration

Line declaration

Compose the line

DATUM <index> : <value>

Write line in the file

Write every clock cycle

Packages for constants and data types

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
package BASIC is
  -- Constants
  constant cBusWidth          : integer := 8;
  constant cAddressWidth      : integer := 16;
  constant cNibbleWidth       : integer := 4;
  constant cWordWidth         : integer := cBusWidth;
  constant cRAMsize           : natural := 2**(cAddressWidth);

  constant cClockSemiperiod   : natural := 15 ns;

  -- Data types
  subtype DataBusType    is std_logic_vector(cBusWidth  -1 downto 0);
  subtype DataWordType   is std_logic_vector(cWordWidth -1 downto 0);
  type    RAMtype        is array (natural range <>) of DataWordType;
  subtype RAMsubtype      : RAMtype(0 to cRAMsize-1);

  -- FSM
  type    Statetype    is (Idle, Start, Parity, Stop, Data, Break);
end BASIC;
```

Modelling memories

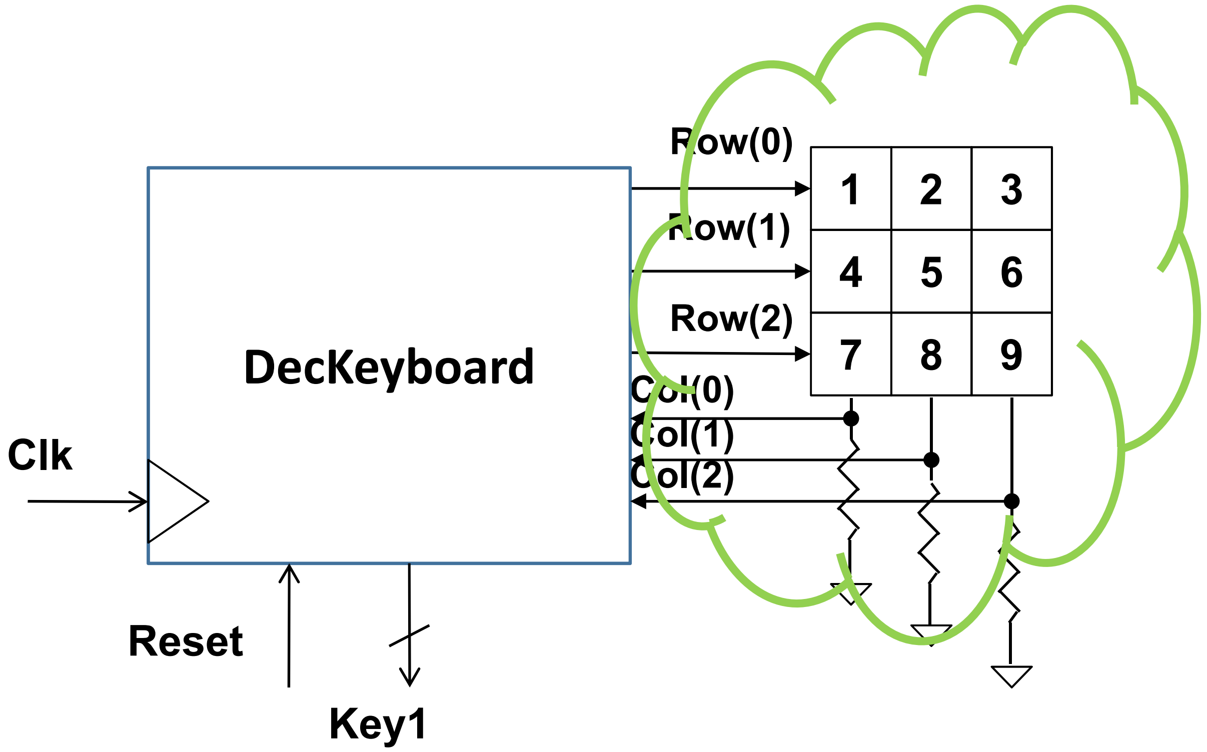
```
entity RAM is
generic(
  gAddrBusWidth : natural := cAddressWidth;
  gMaxSize      : natural := cRAMSize;
  gDataBusWidth : natural := cBusWidth);
port(
    AddrBus   : in    std_logic_vector(gAddrBusWidth-1 downto 0);
    DataBus   : inout std_logic_vector(gDataBusWidth-1 downto 0);
    CS_N      : in    std_logic;
    RD_N      : in    std_logic;
    WR_N      : in    std_logic);
end RAM;
```

Modelling memories

```
architecture BEHAVIORAL of RAM is
  signal  RAM_table    : RAMsubtype := (others => (others => '0'));
  signal  DataBusIn    : DataWordType;
  signal  DataBusOut   : DataWordType;
  signal  EnaWrite     : std_logic;
  signal  EnaRead      : std_logic;
  signal  AuxData      : DataWordType;
Begin
  -- Bidirectional Data Bus
  DataBusIn <= DataBus;
  DataBus   <= DataBusOut when CS_N = '0' AND RD_N = '0' else (others => 'Z');
  -- Managing reading and writing processes
  EnaWrite <= NOT(CS_N OR WR_N);
  EnaRead  <= NOT(CS_N OR RD_N);
  Memory_Access : process
  begin
    wait on AddrBus, EnaWrite, EnaRead, Data;
    -- Read Cycle
    if EnaRead'event and EnaRead= '1' then
      DataBusOut <= RAM_table(CONV_INTEGER(AddrBus));
    elsif AddrBus'event and EnaRead = '1' then
      DataBusOut <= RAM_table(CONV_INTEGER(AddrBus));
    end if;
    -- Write Cycle
    if EnaWrite'event then
      RAM_table(CONV_INTEGER(AddrBus))  <= DataBusIn;
    end if;
  end process Memory_Access;
end BEHAVIORAL;
```

Modelling peripherals

- Matrix Keypad



Modelling peripherals

❑ Matrix Keypad

```
entity keyboard is
  port(
    key: in integer range 0 to 9;
    Row: in std_logic_vector(0 to 2);
    Col: out std_logic_vector(0 to 2)
  );
end keyboard ;
architecture model of keyboard is
begin
  process(key,Row)
    Col <= “000”;
    case  key is
       when 1 =>
         if Row = “100” then – Row 1
           Col <= “100”;
         end if;
       when 2 =>
         if Row = “100” then – Row 1
           Col <= “010”;
         end if;
```

```
       when 3 =>
          if Row= “100” then
           Col <= “001”;
         end if;

        ...

       when others=>
    end case;
  end process;
end model;
```

Modelling peripherals

❑ Matrix Keypad

```
entity TB_Dec is
end TB_Dec;

architecture sim of TB_Dec is
component DecKeyboard
  port(
    Clk    :in std_logic;
    reset  : in std_logic;
    Col    : in std_logic_vector(0 to 2);
    Row    : out std_logic_vector(0 to 2);
    Key1   : out integer range 0 to 9);
end component;

component keyboard is
  port(
    key: in integer range 0 to 9;
    Row: in std_logic_vector(0 to 2);
    Col: out std_logic_vector(0 to 2)
  );
end component ;


signal clk :std_logic:= '0';
signal reset: std_logic;
signal Col: std_logic_vector(0 to 2);
signal Row: std_logic_vector(0 to 2);
signal key1 : integer range 0 to 9;
Signal PushKey: integer range 0 to 9;

constant HalfT: time := 500 ns;
constant ttap: time := 10 ms;
begin
```

Modelling peripherals

- Matrix Keypad

```
clk <= not clk after HalfT;
Reset <= ‘1’, ‘0’ after 3*HalfT;
  process --modelling 3 taps
  begin
  --short key 3
    PushKey <= 3;
    wait for ttap;
  --short key 9
    PushKey <= 9;
    wait for ttap;
  --long key 3
    PushKey <= 3;
    wait for 10*ttap;

    assert false
    report “END"
    severity failure;
  end process;
```

```
  MAP_CUT: DecKeyboard
  port map(
    Clk    => Clk   ,
    reset  => reset ,
    Col    => Col   ,
    Row    => Row   ,
    kwy1   => key1);

  MAP_model: keyboard
  port(
    key => PushKey,
    Row => Row,
    Col => Col
  );

end sim;
```