Universidad Carlos III de Madrid

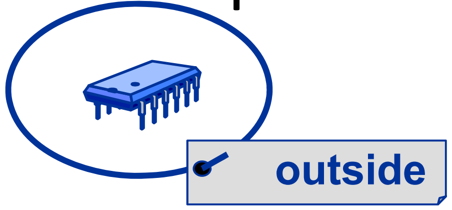
Circuit Simulation of digital circuits with VHDL

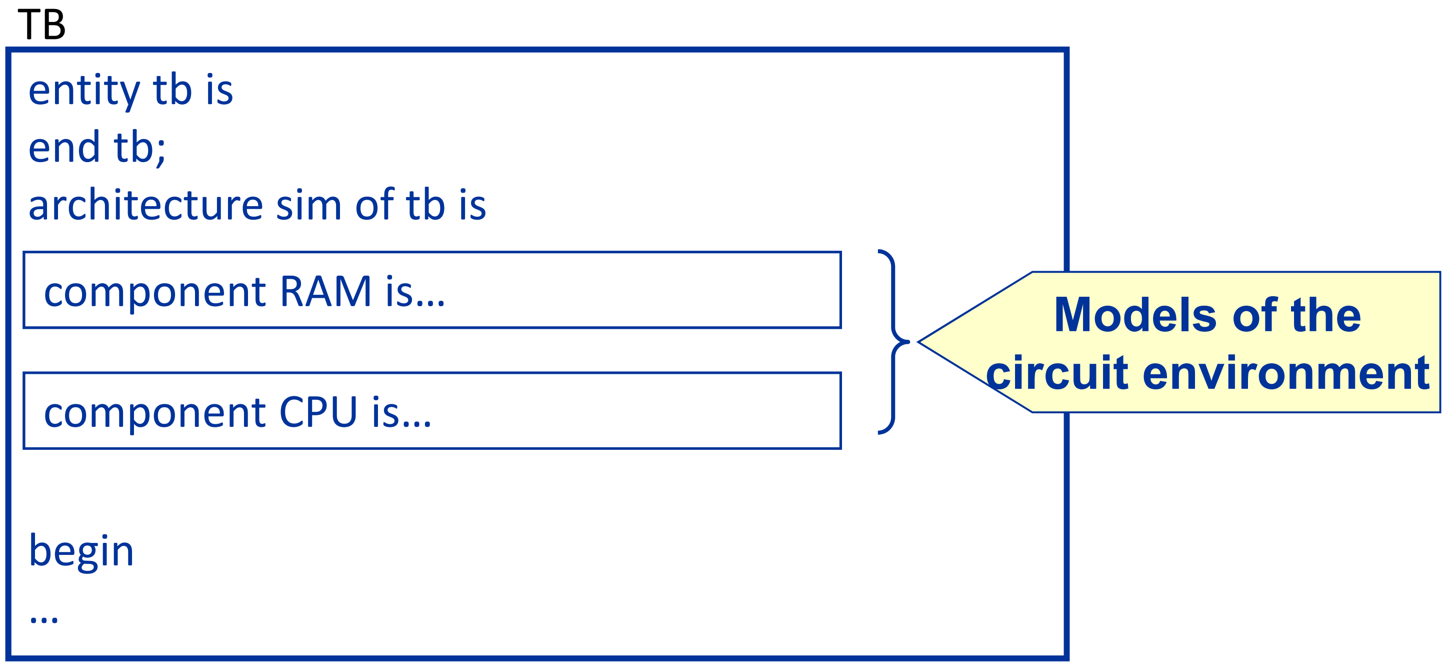
SIMULATION MODELS

Outline

Introduction

- Description or simulation environments in VHDL





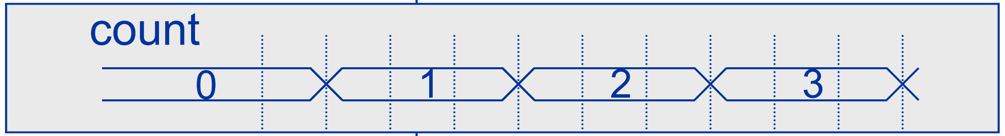
```
entity tb is
end tb;
architecture sim of tb is
component RAM is…
component CPU is…

begin
…
```

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

Write line in the file

Write every clock cycle

DATUM <index> : <value>

Packages for constants and data types

```
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

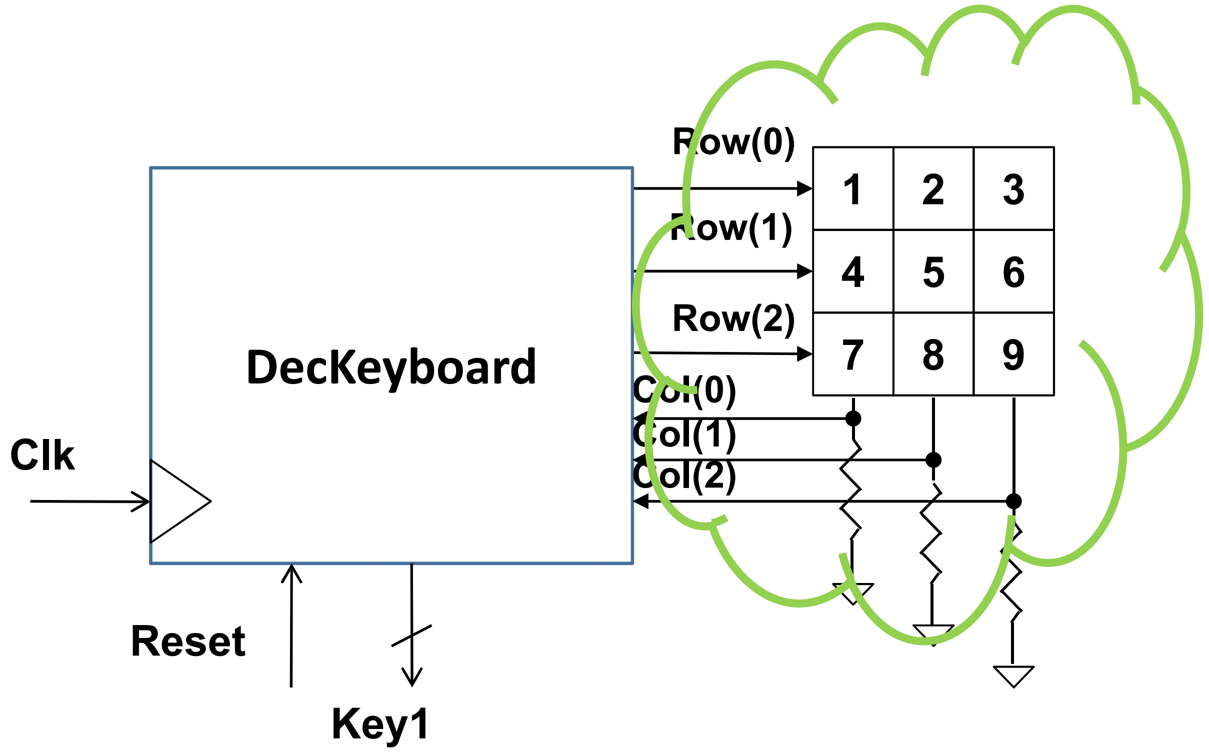
```
entity RAM is
generic(
  gAddrBusWidth : natural := cAddressWidth;
  gMaxSize      : natural := cRAMSize;
  gDataBusWidth : natural := cBusWidth);
port(
    AddrBus   : in    std_logic_vector(gAddrBusWidth-1 downto 0);
    DataBus   : inout std_logic_vector(gDataBusWidth-1 downto 0);
    CS_N   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