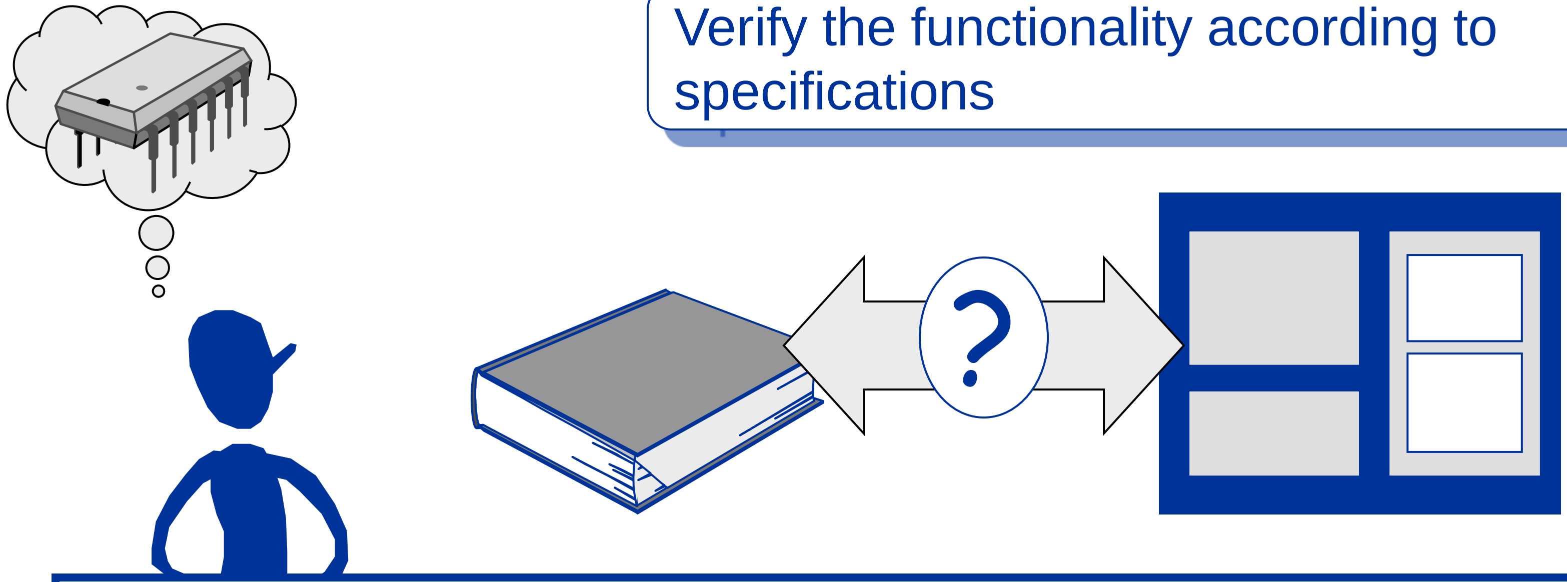
Circuit Simulation with VHDL

Authors: Celia López, Luis Entrena Arrontes, Mario García, Enrique San Millán, Marta Portela, Almudena Lindoso

Outline

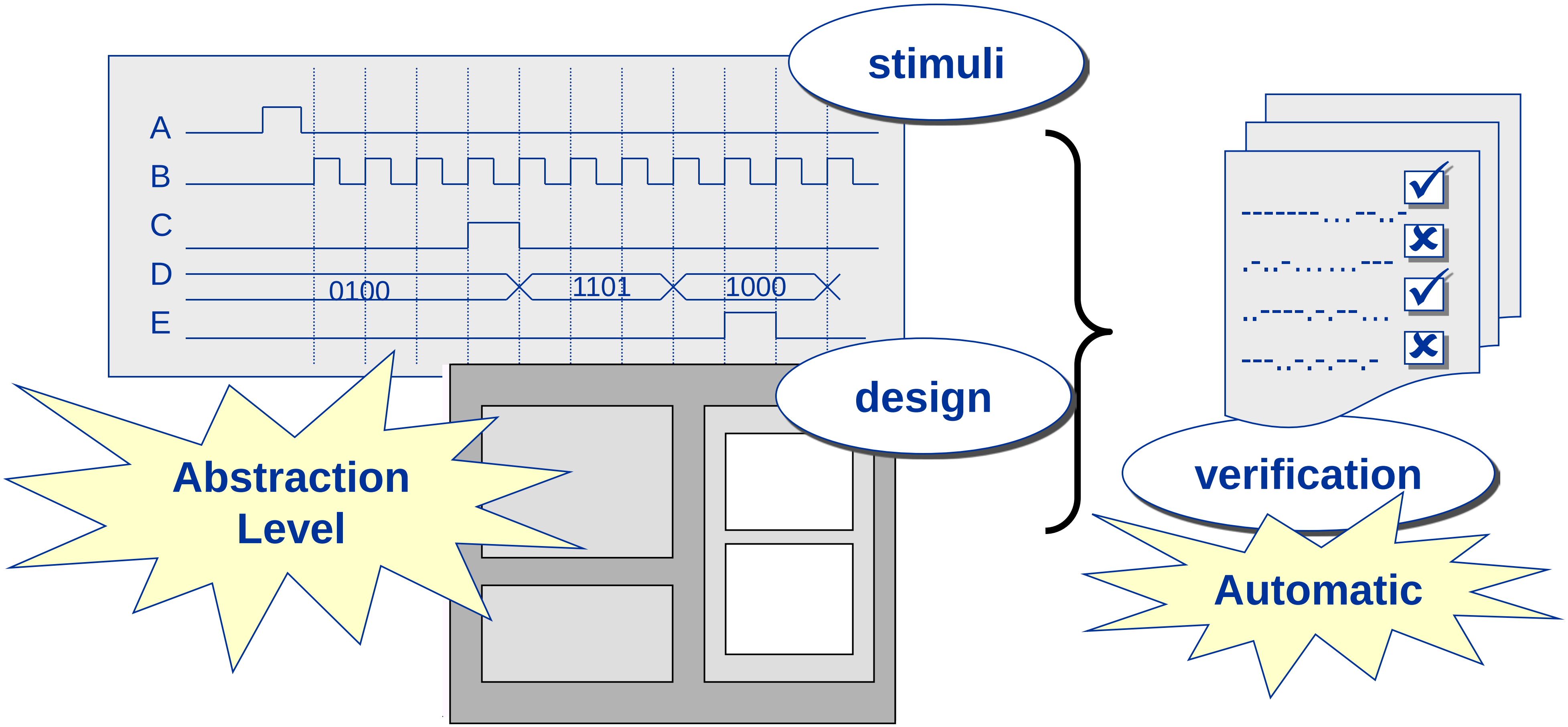
Functional validation

Verify the functionality according to specifications

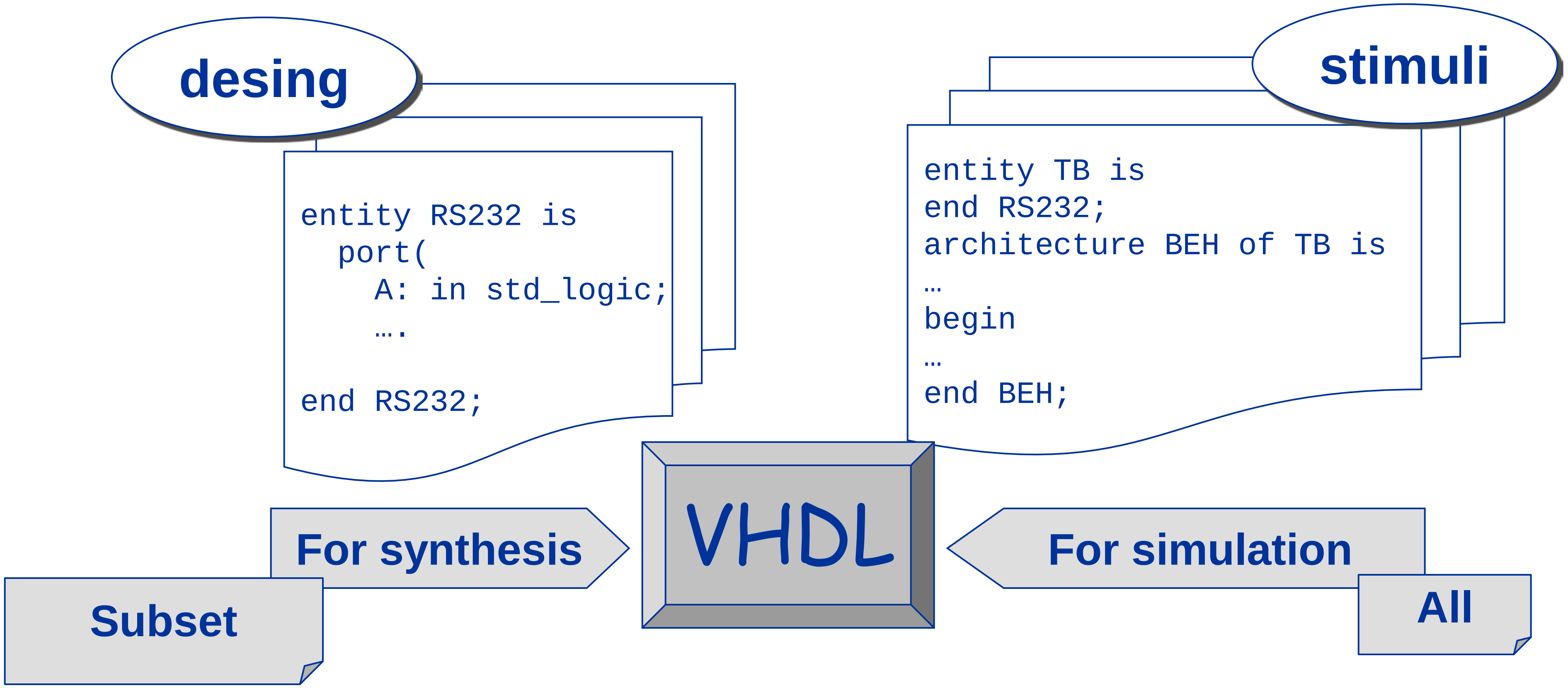


Once the desing has been described, how can we be sure that it works according to specifications?

Functional validation: Necessary elements



Functional validation: Necessary elements



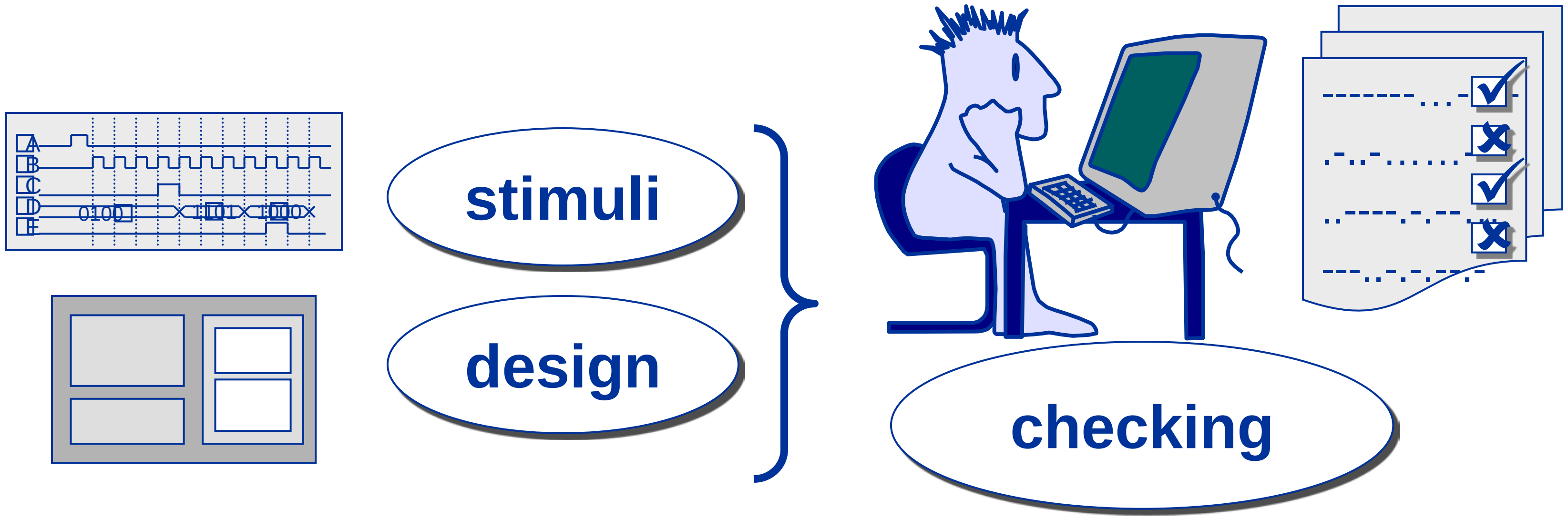
```
entity RS232 is
  port(
    A: in std_logic;
    ….

end RS232;
```

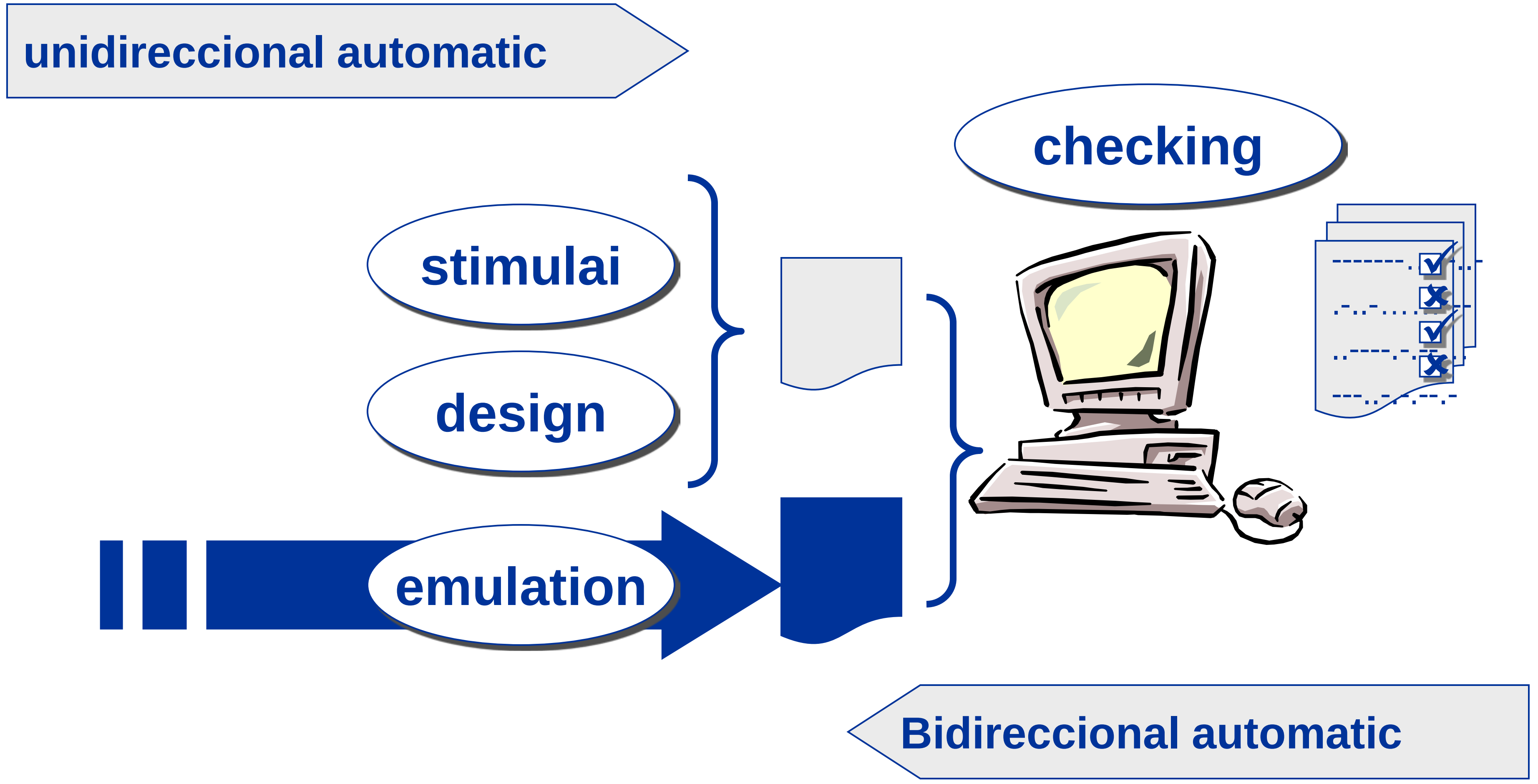
```
entity TB is
end RS232;
architecture BEH of TB is
…
begin
…
end BEH;
```

Functional Validation Levels

Visual/manual



Functional Validation Levels



Functional Validation with VHDL

Language conceived for simulation and speficication

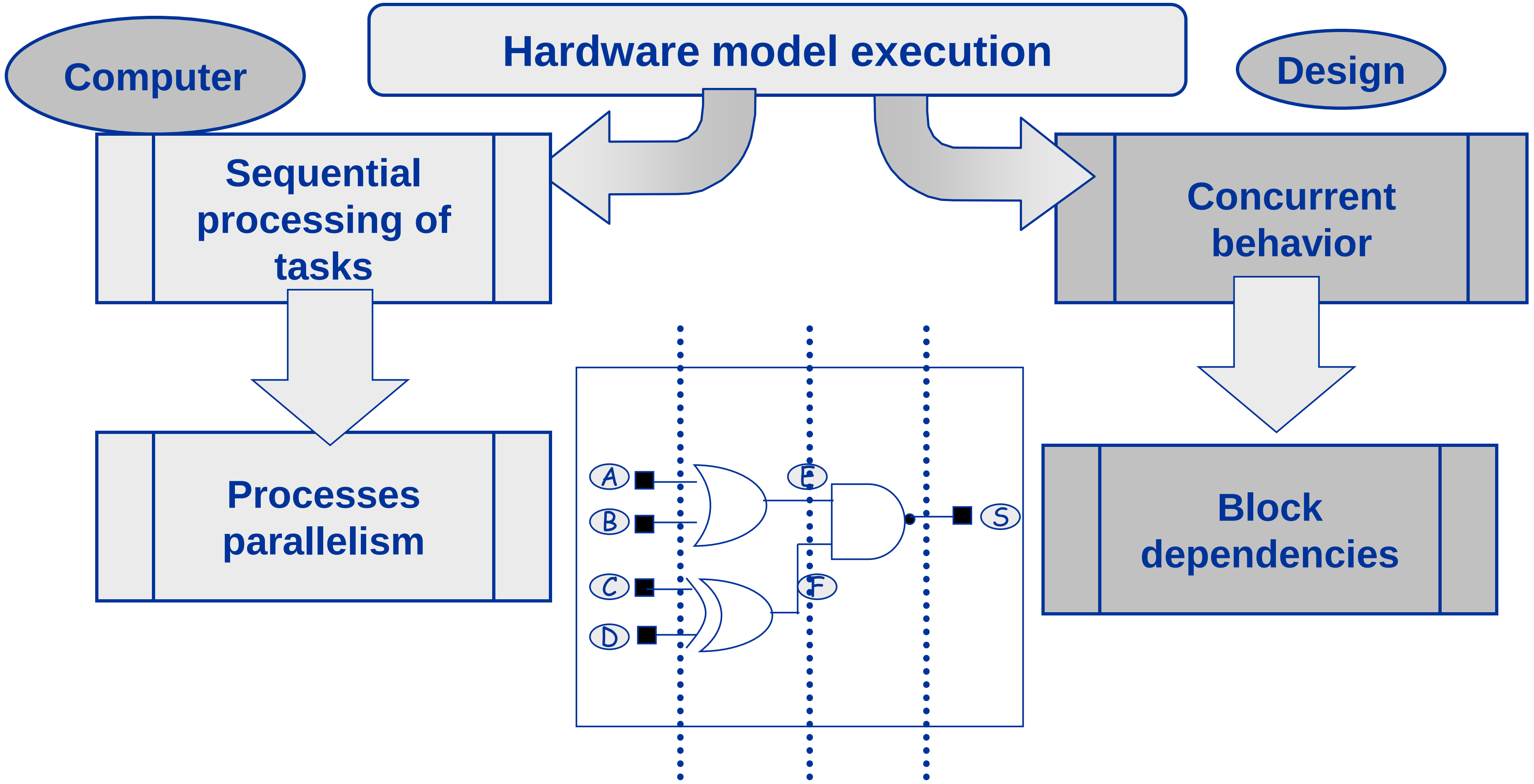
Possibility of automatic and interactive checking

High level modelling of external interfaces

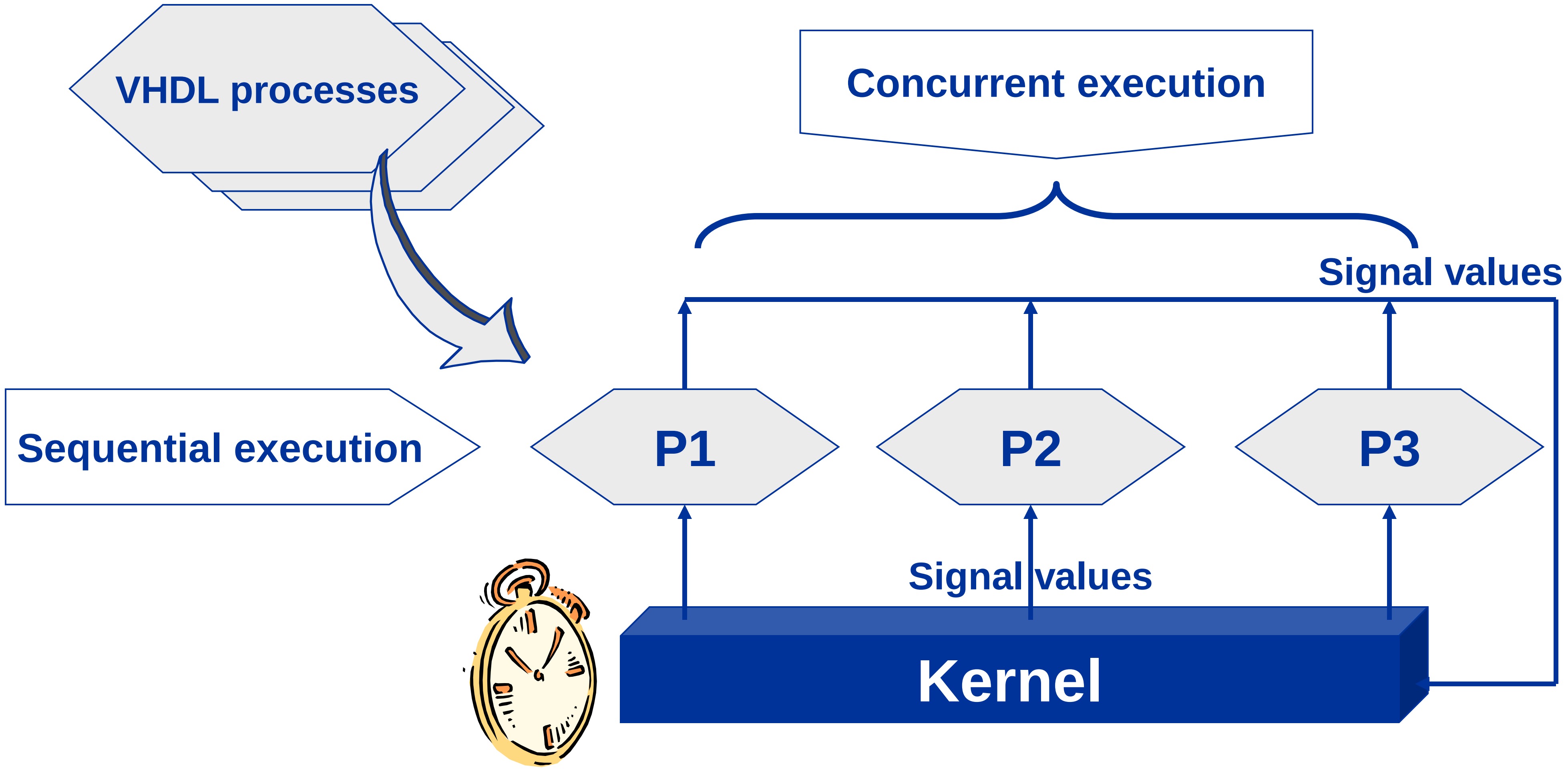
Mecanisms for writing/reading files

Specification of timings and delays

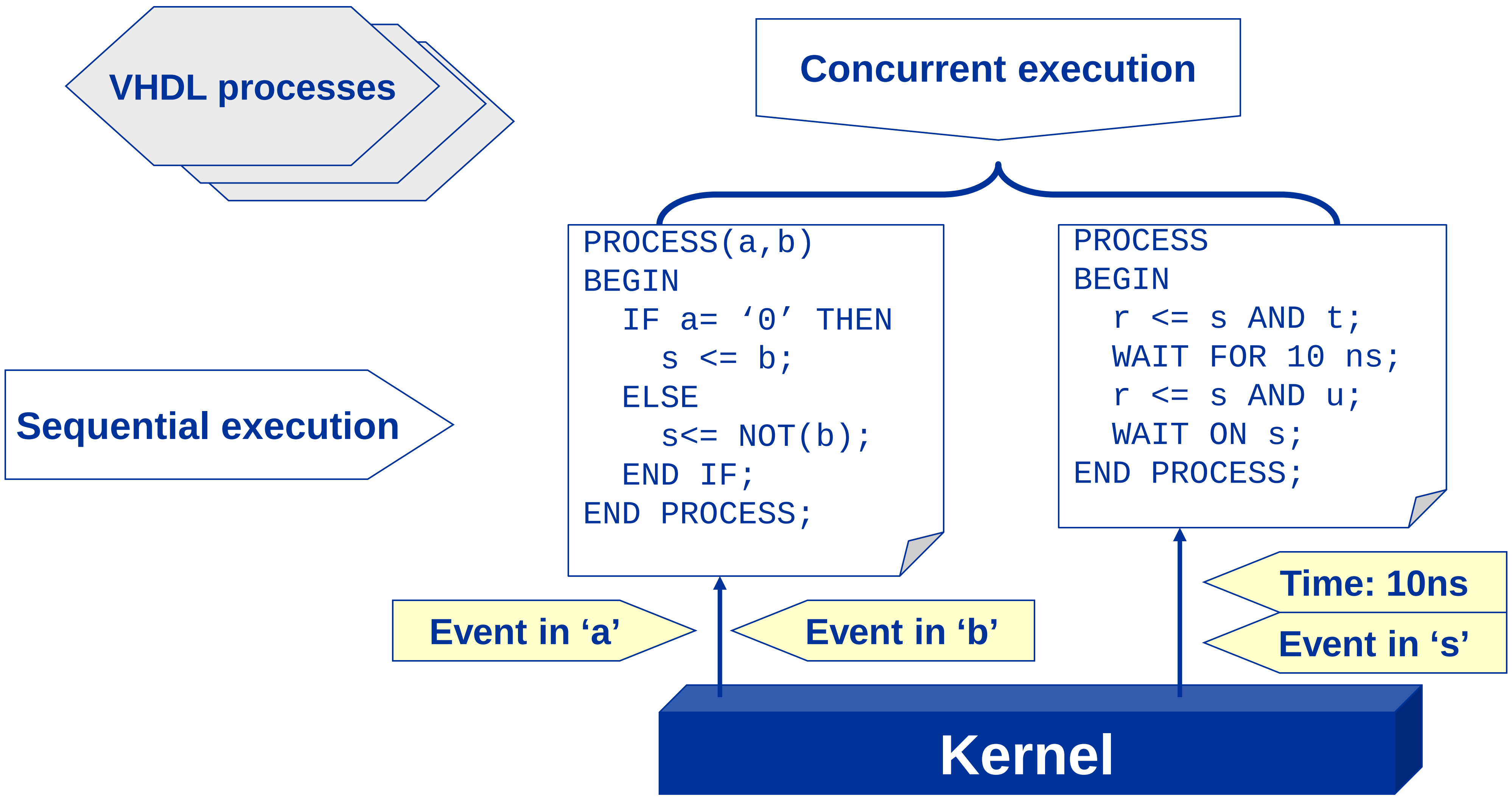
Digital circuit simulation



Digital circuit simulation



Digital Circuit Simulation



```
PROCESS(a,b)
BEGIN
  IF a= '0' THEN
    s <= b;
  ELSE
    s<= NOT(b);
  END IF;
END PROCESS;
```

```
PROCESS
BEGIN
  r <= s AND t;
  WAIT FOR 10 ns;
  r <= s AND u;
  WAIT ON s;
END PROCESS;
```

Digital circuit simulation

VHDL processes

Sequential sentences

after → **A <= B after 20 ns;**

wait →

- **wait on A;** — **signals**
- **wait for 10 ns;** — **time**
- **wait until C1;** — **conditions**
- **wait ;**

```
TYPE time IS RANGE implementation_defined
  UNITS
    fs
    ps = 1000fs;
    ns = 1000ps;
    us = 1000ns;
    ms = 1000us;
    sec = 1000ms;
    min = 60sec;
    hor = 60min;
  END UNITS;
```

Package: STANDARD

Testbench

stimuli

As much complete as possible

~~¿Every possible combination?~~

Highest possible abstraction

Automatic translation

As much exhaustive as possible

To avoid external factors

Basic rules for testbenchs

Asynchronus initialization of the full system

Every operation mode / FSMs states

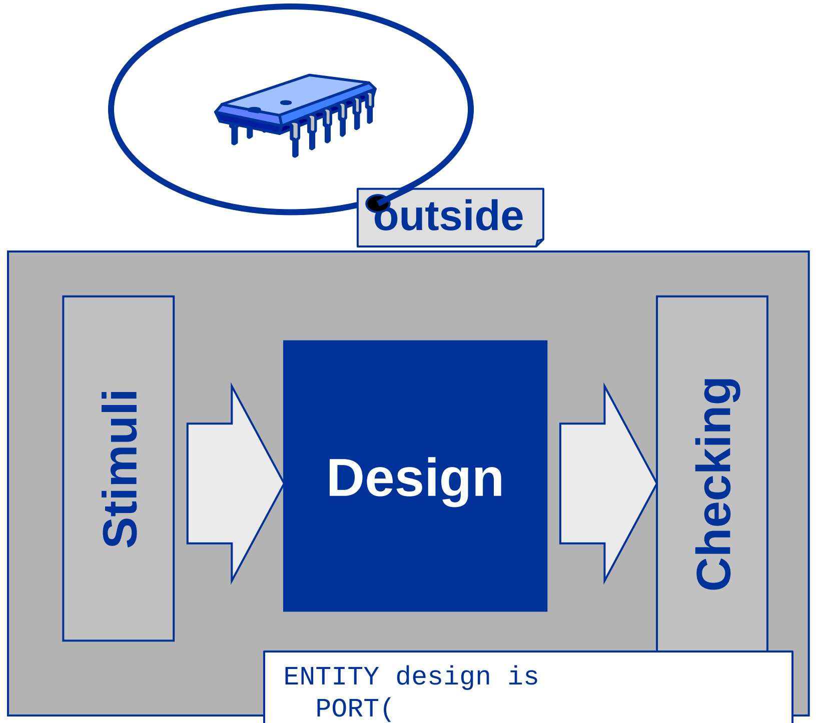
Read and write every register

Every operation mode in buses

Code coverage

Effect in the outputs

Basic structure of testbenchs



```
ENTITY design is
  PORT(
    …);
END design;

ARCHITECTURE beh OF design is
  SIGNAL S1a : std_logic;
  …
BEGIN
…
END beh;
```

```
ENTITY tb is
END tb;
ARCHITECTURE mixed OF tb is
  COMPONENT design IS
    PORT(
      …);
  END COMPONENT;
  FOR D: design
    USE entity WORK.beh(design)
  SIGNAL s1 : std_logic;
  …
BEGIN
  D: design
  PORT MAP(
      …);
  …
  -- STIMULI !!!
  -- CHECKING !!!
END mixed;
```

- empty entity
- component
- specification
- I/O and more
- mapping
- stimuli
- checkings

Stimuli generation: Clocks

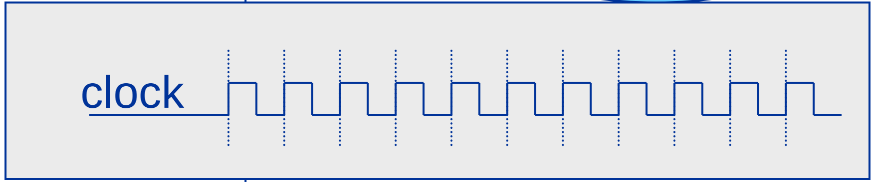
```
ENTITY tb is
END tb;

ARCHITECTURE mixed OF tb is
  SIGNAL clock: std_logic := ‘0’;
  CONSTANT semi_period : natural := 20;
BEGIN
…
  -- STIMULI !!!
  clock <= NOT(clock) after semi_period ns;
END mixed;
```

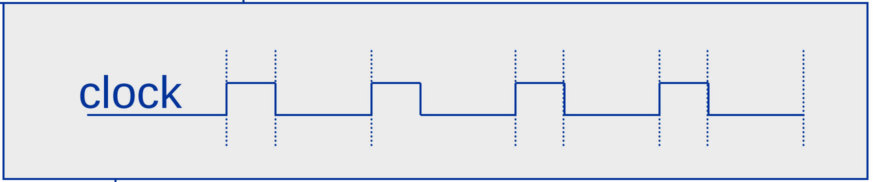
stimuli

Simple

Periodic



```
PROCESS
BEGIN
  clock <= ‘0’;
  WAIT FOR semi_period_L ns;
  clock <= ‘1’;
  WAIT FOR semi_period_H ns;
END PROCESS;
```



Stimuli generation: Reset

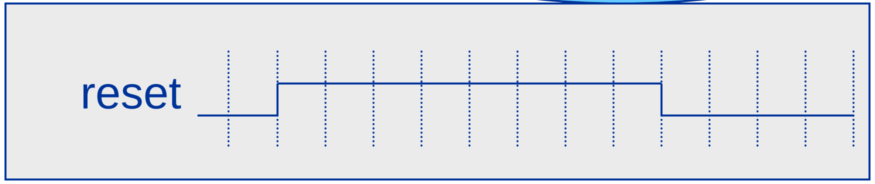
```
ENTITY tb is
END tb;

ARCHITECTURE mixed OF tb is
  SIGNAL reset: std_logic;
  CONSTANT reset_on : natural := 1000;
BEGIN
…
  -- STIMULI !!!
  PROCESS
  BEGIN
    reset <= ‘0’;
    WAIT FOR 150 ns;
    reset <= ‘1’;
    WAIT FOR reset_on ns;
    reset <= ‘0’;
    WAIT;
  END PROCESS;
END mixed;
```

stimuli

Simple

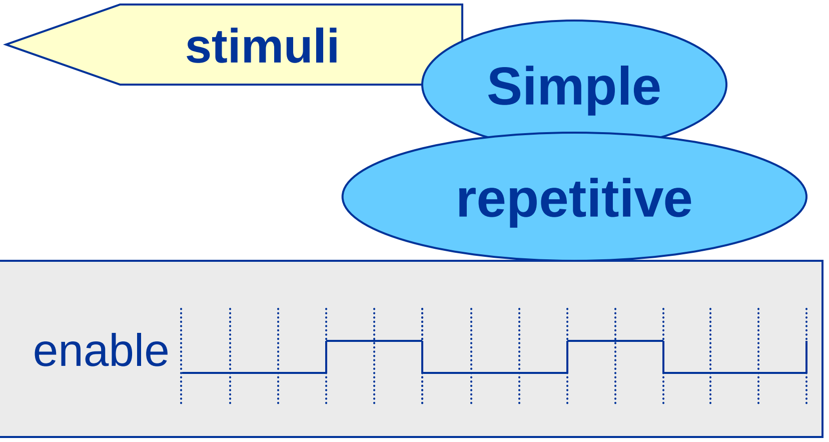
Non periodic



Stimuli generation: Enable

```
ENTITY tb is
END tb;

ARCHITECTURE mixed OF tb is
  SIGNAL enable: std_logic;
BEGIN
…
  -- STIMULI !!!
  PROCESS
  BEGIN
    enable <= ‘0’;
    WAIT FOR 150 ns;
    enable <= ‘1’;
    WAIT FOR 100 ns;
    enable <= ‘0’;
  END PROCESS;
END mixed;
```



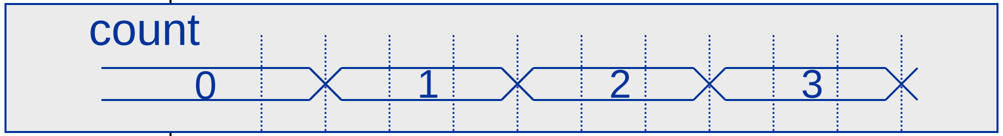
Stimuli generation: Input data

```
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;
```

estimuli

Complex counters



Stimuli generation: Memories

```
…
ARCHITECTURE mixed OF tb is
  TYPE TableType is array (natural range <>) of
                    std_logic_vector(3 downto 0);
  SIGNAL ValueTable: TableType (0 to 20)
                   := (“0011”, “0110”, “0111”,
                      “1011”, “1110”, “1111”,
                      “1001”, “0111”, “0101”,
                      “1010”, “0111”, “0011”,
                      “0000”, “1110”, “0110”,
                      “0100”, “0001”, “0101”,
                      “0010”, “0100”);
BEGIN
  PROCESS(Index)
  BEGIN
   Value <= ValueTable(Index);
END PROCESS;
END mixed;
```

estimuli

Complex

tables

Testbench checking

```
ENTITY tb is
END tb;

ARCHITECTURE mixed OF tb is
  -- Count from 0 to 10
  SIGNAL s1 : std_logic_vector(3 downto 0);
  …
BEGIN
  -- Checking !!!
  -- Check if s1 has not overflowed
  ASSERT CONV_INTEGER(s1) < 11
    REPORT “Overflow in Count”
     SEVERITY warning;
  -- Check if Hold time is met
  ASSERT (Now – LastEvent) >= HoldTime
    REPORT “Hold time violation”
      SEVERITY warning;
END mixed;
```

Checking

assert

condition **false**

report **texto**

severity

- **NOTE**
- **WARNING**
- **ERROR**
- **FAILURE**

Stop the simulation

STANDARD package

Interaction with the outside world

Input and output files

- **The type FILE defines a file data type**
- **A FILE declaration defines an identifier for a file and associates it to a physical file**
- **FILE objects can not be assigned, but they can be read or written through specific subprograms:**
  - **Procedure READ (file_type, datum)**
  - **Procedure WRITE (file_type, datum)**
  - **Function ENDFILE (file_type)**
- **The TEXTIO package, available in all the VHDL environments, defines types, procedures and functions for reading and writing ASCII files**

I/O files. Stimuli reading

```
ARCHITECTURE mixed OF tb is
-- File declaration. TEXT type
  FILE input_file: TEXT IS IN “input.dat”;
BEGIN
  -- STIMULI !!!
  PROCESS
    VARIABLE val1 : char;
    VARIABLE val2 : integer;
    VARIABLE file_line: LINE;
  BEGIN
    IF not(ENDFILE(input_file)) THEN
      -- to read a text line:
      READLINE (input_file, file_line);
      -- The datum is interpreted as char:
      READ (file_line, val1);
      -- The datum is interpreted as integer:
      READ (file_line, val2);
    END IF;
    Dato_Input <= Val2;
    WAIT FOR 100 ns;
  END PROCESS;
END mixed;
```

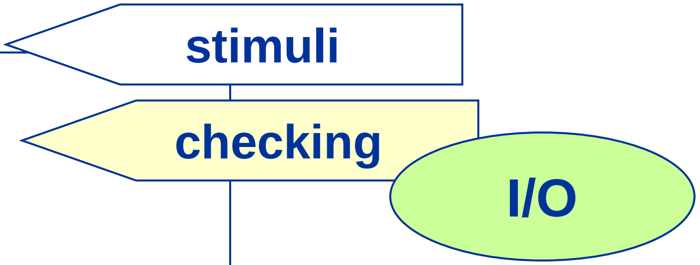
stimuli

checking

I/O

I/O files. Results

```
ARCHITECTURE mixed OF tb is
  -- File declaration
  FILE output_file: TEXT IS OUT “output.dat”;
BEGIN
  -- STIMULI !!!
  PROCESS(Datum)
    VARIABLE file_line: LINE;
  BEGIN
    -- Para escribir una línea
    WRITE (file_line, Datum);
    WRITELINE (output_file, file_line);
    WAIT FOR 100 ns;
  END PROCESS;
END mixed;
```



Commercial tools

ASIC
FPGA

Simulation
Synthesis
Place&Route
Layout

Mentor Graphics
- **ModelSim**

Synopsys
- **VCS**
- **Design Compiler**

Cadence

FPGA manufacturers

ALTERA → Quartus II

XILINX → ISE

ACTEL → Libero