


# *Designing Complex FPGAs*

-do it right, the first time-



Bert CUZEAU  
Technical Manager - ALSE  
ASIC / FPGA Design Expert  
Doulos HDL Instructor (Verilog-VHDL)

info@alse-fr.com

<http://www.alse-fr.com>

☎ : 33.(0)1 45 82 64 01

# *What is a “Complex” FPGA ?*

Note : Complex  $\neq$  Dense :

- A 1 Meg gates FPGA may be very simple to design...

Some aggravating factors :

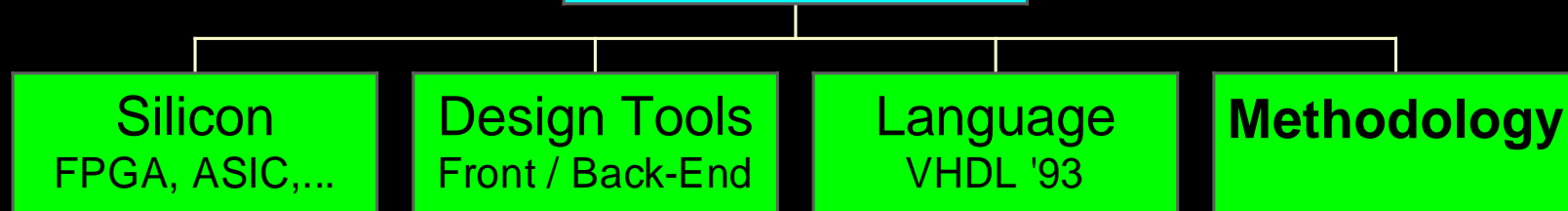
- Asynchronism !
- Poor specs...
- External Environment complex or without model.
- Close to Technological Limits.
- Complex algorithms within few clock cycles.

Hint : look at the Test Plan.

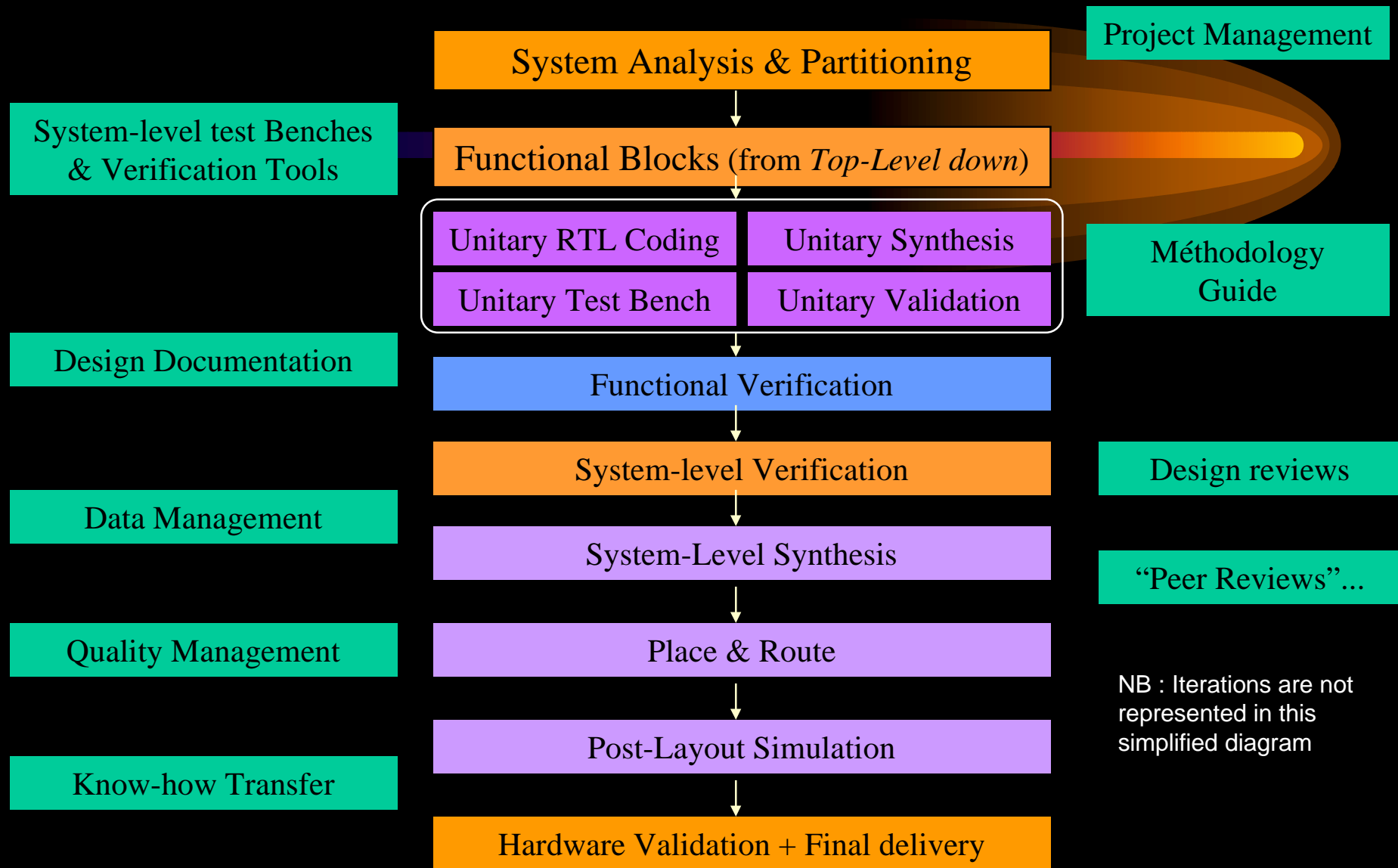
# *Ground for Success*



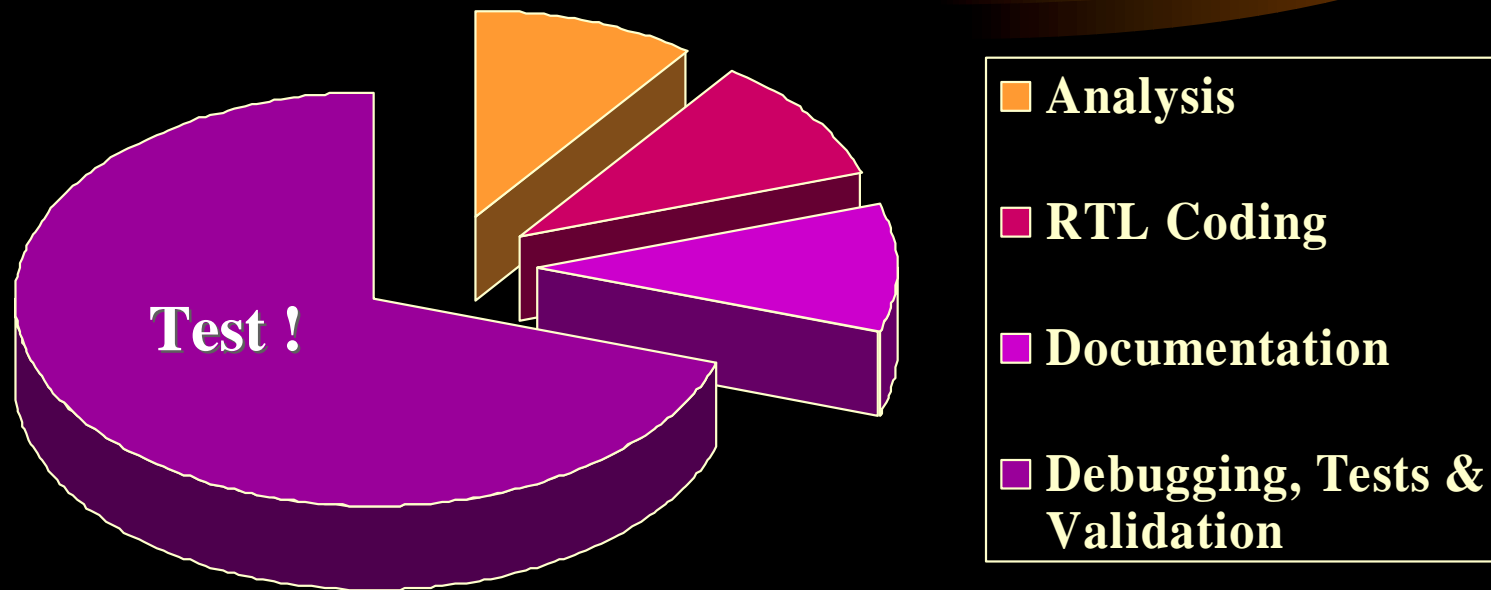
## **Know-How**



# FPGA Design Flow



# *Time to Design !*



Morale : *Coding time* does not need to be accelerated !  
(Quality is more important than speed)  
Debug + Test + Verification is the bottleneck.

# *Simulation is the Key*



## **Advanced Functional Simulation is one Key to success :**

- Shorten, secure and simplify the validation phase.
- Allow for non-regression tests
- Simplify the delivery and acceptance phase.

A clever simulation methodology can have an incredible impact (3 x gains are common).  
=> delays & costs can easily be divided by 2 !

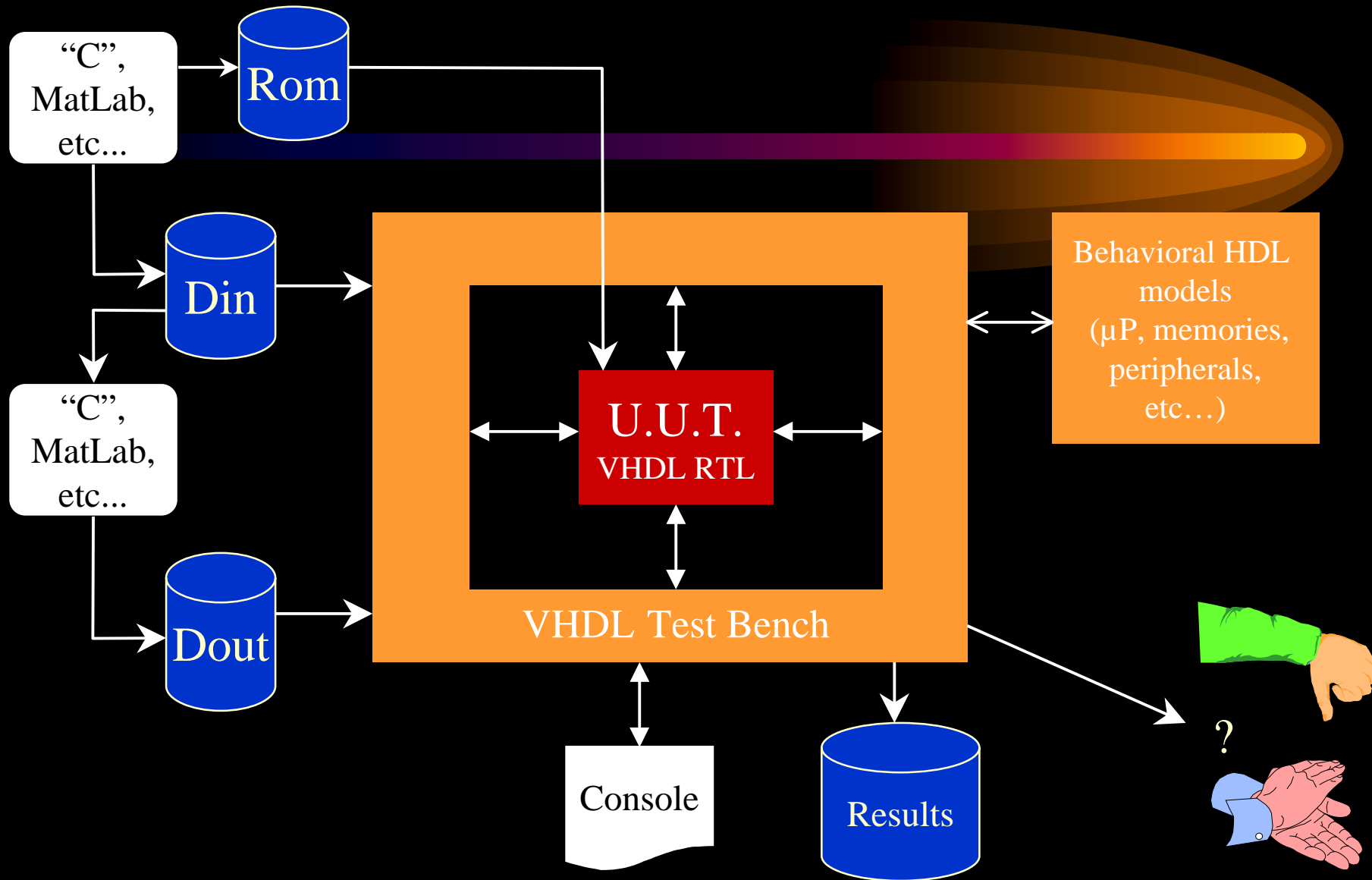
# *Speeding up the Validation phase*



Very simple Techniques can dramatically improve the Simulation efficiency, like :

- Automation (GUI → Scripts),
- Test benches should be self-testing & regressionable,
- Don't eyeball waveforms,
- Use extensively File I/O,
- Use “on-the-fly” comparison,
- Implement interactive tests, ...

# Test Bench Structure Example



## Example : VHDL Code Generator in 'C'

```
fprintf(fdest, "  -- Synthesized Waveform Table\n");
fprintf(fdest, "  -- (c) Bert Cuzeau - info@alse-fr.com\n");
fprintf(fdest, "  subtype SLV8 is std_logic_vector (7 downto 0);\n");
fprintf(fdest, "  type Rom64x8 is array (0 to 63) of SLV8;\n");
fprintf(fdest, "  constant Sinus_Rom : Rom64x8 := (\n  ");

for (n=0; n<n_ech; n++)
{
    slong = sin (twopi*n/(4 * n_ech));
    slong *= 127;  // for 7 bits
    sint = (int) slong;
    fprintf (fdest, "  x\"%02x\"", sint);
    if (n != n_ech-1) fprintf (fdest, ", ");
    if ((n % 8)==7) fprintf(fdest, "\n  ");
}
fprintf(fdest, "  ); ");
fclose(fdest);
```

### Result :

```
subtype SLV8 is std_logic_vector (7 downto 0);
type Rom64x8 is array (0 to 63) of SLV8;
constant Sinus_Rom : Rom64x8 := (
    x"00", x"03", x"06", x"09", x"0c", x"0f", x"12", x"15",
    x"18", x"1b", x"1e", x"21", x"24", x"27", x"2a", x"2d",
    x"30", x"33", x"36", x"39", x"3b", x"3e", x"41", x"43",
    x"46", x"49", x"4b", x"4e", x"50", x"52", x"55", x"57",
    x"59", x"5b", x"5e", x"60", x"62", x"64", x"66", x"67",
    x"69", x"6b", x"6c", x"6e", x"70", x"71", x"72", x"74",
    x"75", x"76", x"77", x"78", x"79", x"7a", x"7b", x"7b",
    x"7c", x"7d", x"7d", x"7e", x"7e", x"7e", x"7e", x"7e" );
```

# *RTL Coding*

**Your HDL coding style must take advantage of :**

- **new tools (HDL synthesis has progressed !)**
- **new components (embedded blocks)**
- **~~Abandon the discrete FFs & logic style !~~**
- **~~Give up on complex logic Equations !~~**
- **~~No more TTL Macros !!!~~**
- **+ Complex Finite State Machines**
- **+ Behavioral RTL (more inference)**

# *Take advantage of the Silicon*



You may find that some families are better suited to such or such kind of application :

- Look at complex macro-functions : pipelined Multipliers, Rams / Roms / Dual Ports, special operators, etc...
- Predictability & stability of performances,
- Synthesis & Simulation tools,
- Efficiency & Ease of use of the Place & Route tools,
- etc...

# *Kow-How*



- Designers should go through an excellent HDL Training Course.
- The synthesis & FPGA technology know-how should be updated.
- Sound Digital Design principles must be understood & enforced !
- A Solid Experience is required (Design & Tools)

Note : Excellent Designer  $\Rightarrow$  Very Simple code...

## *Beware !*

- Not all Books & Training Courses are good ...
- Existent code (Books, In-house, Internet,...) may be of poor quality or obsolete...

# Conclusion

Today more than ever, success relies upon the Designer's Know-How and Skills, but a bullet-proof methodology is mandatory. ("kludges" don't work any more).

- HDL Languages,

- Synthesis and Simulation Tools and

- Complex Programmable Devices

are not widely available !

Just add the right know-how + a good methodology and attack the most complex projects !

# - Appendix A -

## Simple Behavioral send/receive UART !

### Transmit (N-8-1)

```
TX <= '0'; -- Start bit
wait for BITperiod;

for i in 0 to 7 loop
    TX <= RSDData(i);
    wait for BITperiod;
end loop;

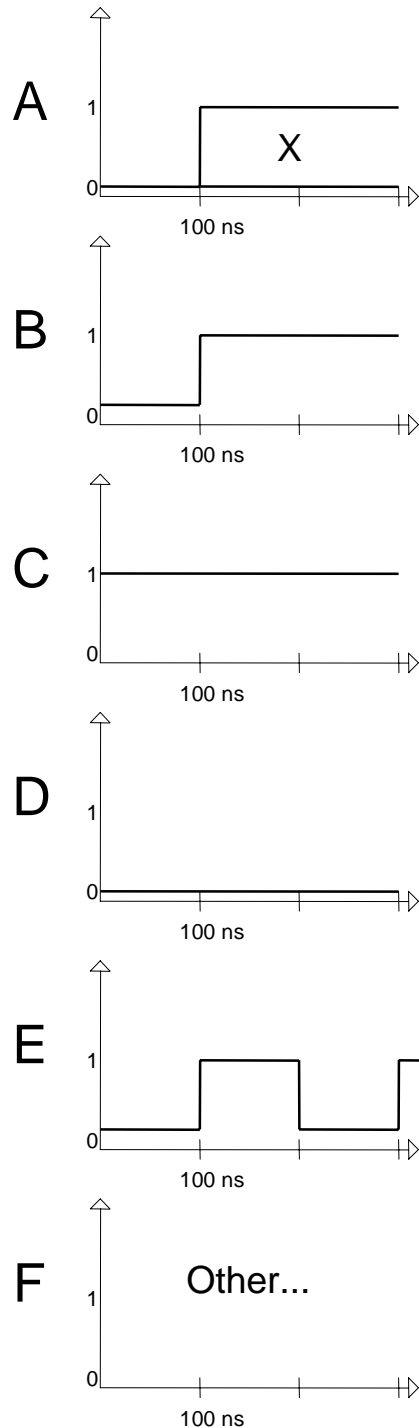
TX <= '1'; -- Stop bit
wait for BITperiod;
```

### Receive (N-8-1)

```
loop
    wait until RX='0'; -- Get falling edge
    wait for (0.5 * BITperiod); -- Middle of Start bit
    assert RX='0' report "Start Bit Error ???"
        severity warning;
    wait for BITperiod; -- First Data Bit
    for i in 0 to 7 loop -- Get word
        DATA(i) := RX;
        wait for BITperiod;
    end loop;
    wait for BITperiod; -- Stop bit
    assert RX='1' report "Stop bit Error ???"
        severity warning;
    WRITE (L, ... (DATA)); -- print the char in a file
    WRITELINE (F, L); -- or in the transcript
end loop;
```

Conclusion : An HDL language (Verilog/VHDL) is perfect to easily model external components and allow for system-level simulation.

## - Appendix B - VHDL Quizz



```
library IEEE;
use IEEE.std_logic_1164.all;

entity quizz is
end quizz;

architecture test of quizz is
    signal nRST : std_logic;
begin

    process
    begin
        nRST <= '0';
        nRST <= '1' after 100 ns;
        wait;
    end process;

end test;
```

?

*Hint : answer is "F". Was it obvious ?*

*What happens if the second line is :  
nRST <= '0' after 100 ns; ?*