

Rodrigo A. Melo

VHDL for FPGA Simulation

Virtual | Feb | 2021



**Joint ICTP-IAEA School on FPGA-based
SoC and its Applications for Nuclear and**



Related Instrumentation | (smr 3562)



Ministerio de
Desarrollo Productivo
Argentina

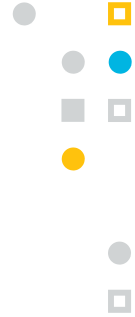
Outline

1 Introduction

2 VHDL for simulation

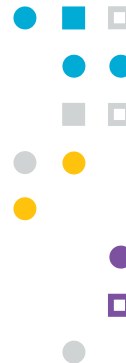
3 Simulators

4 Conclusions

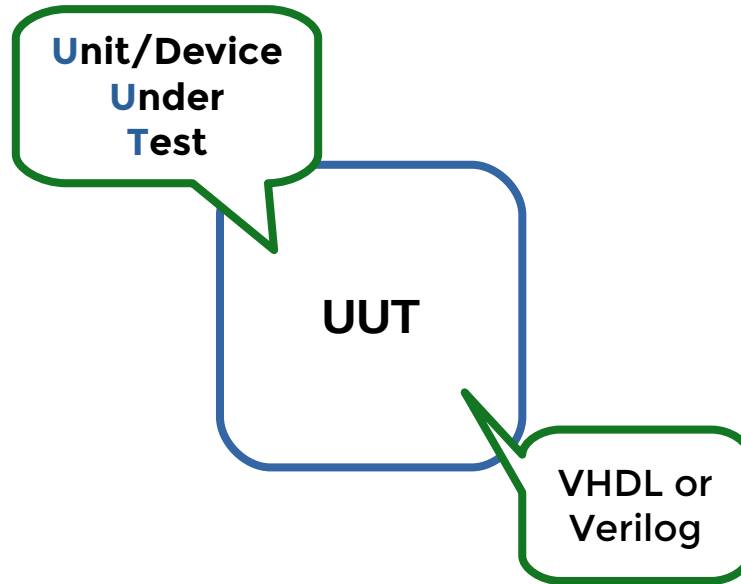
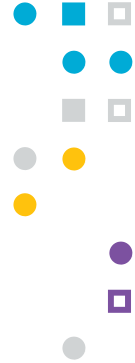


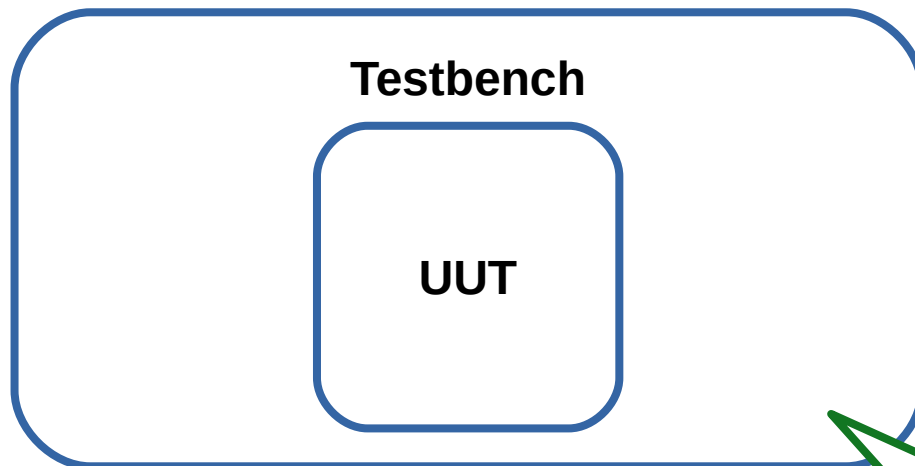
Introduction



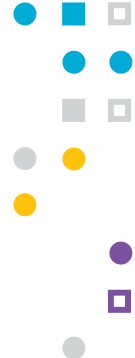
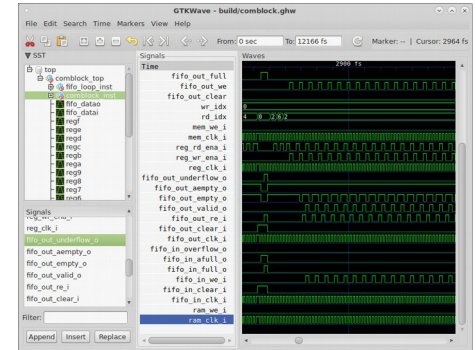
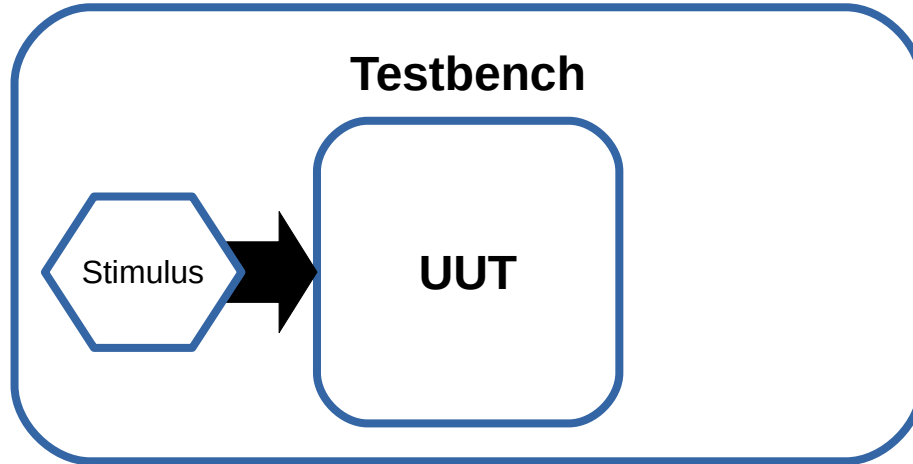


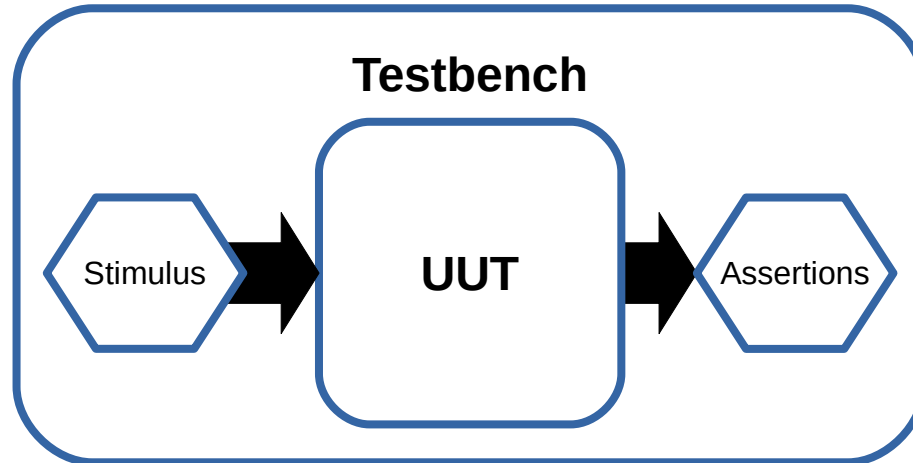
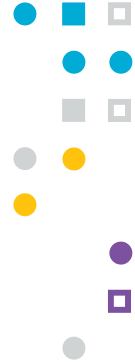
- **Verification:** to check if a design works as expected.
- **In-Circuit Debug:**
 - Performed in a test bench, with a Scope (simple design with a few signals), Logic Analyzer (**\$\$\$**) or Embedded Logic Analyzer (modify your design).
- **Functional verification:**
 - Stimulus are provided and the output compared with expected values, according to a specification.
 - It implies a testbench (generally an HDL) and a simulator
- **Formal verification:**
 - Generally speaking, you need to specify formal properties and a tool checks your design against formal methods (mathematical proof). Ex: **SymbiYosys** (FOSS).

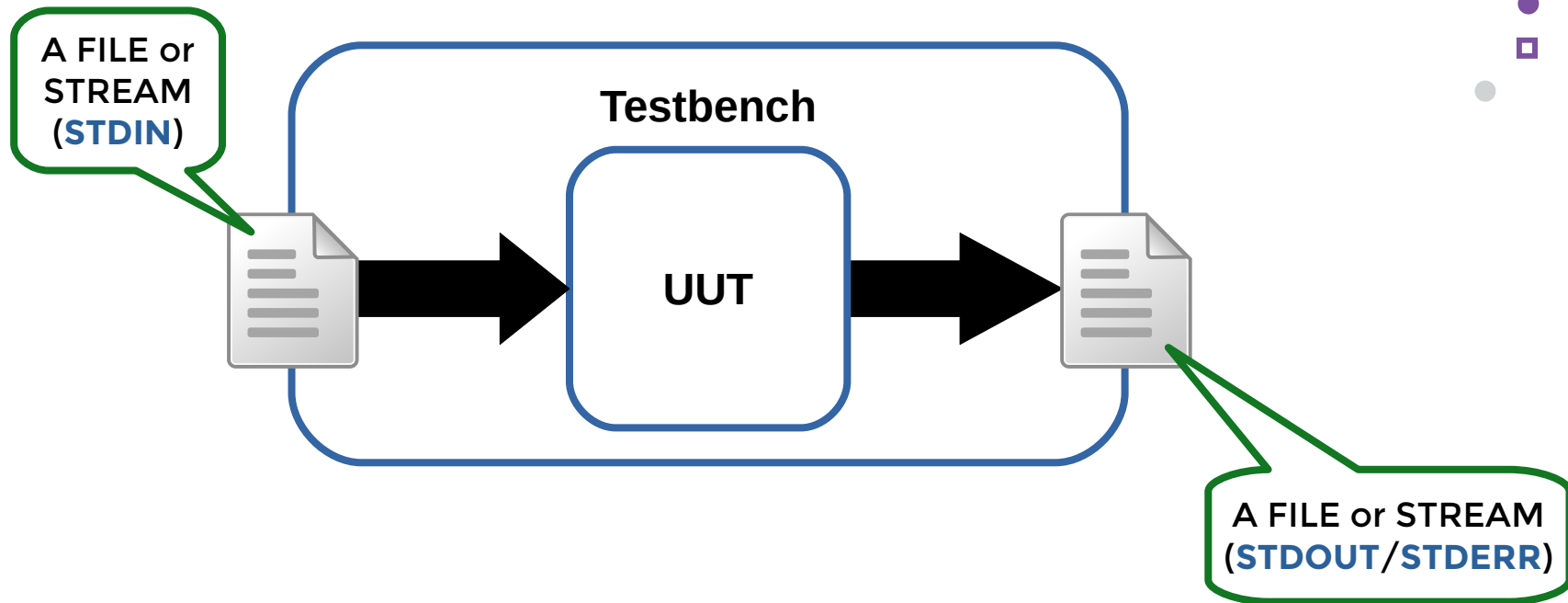


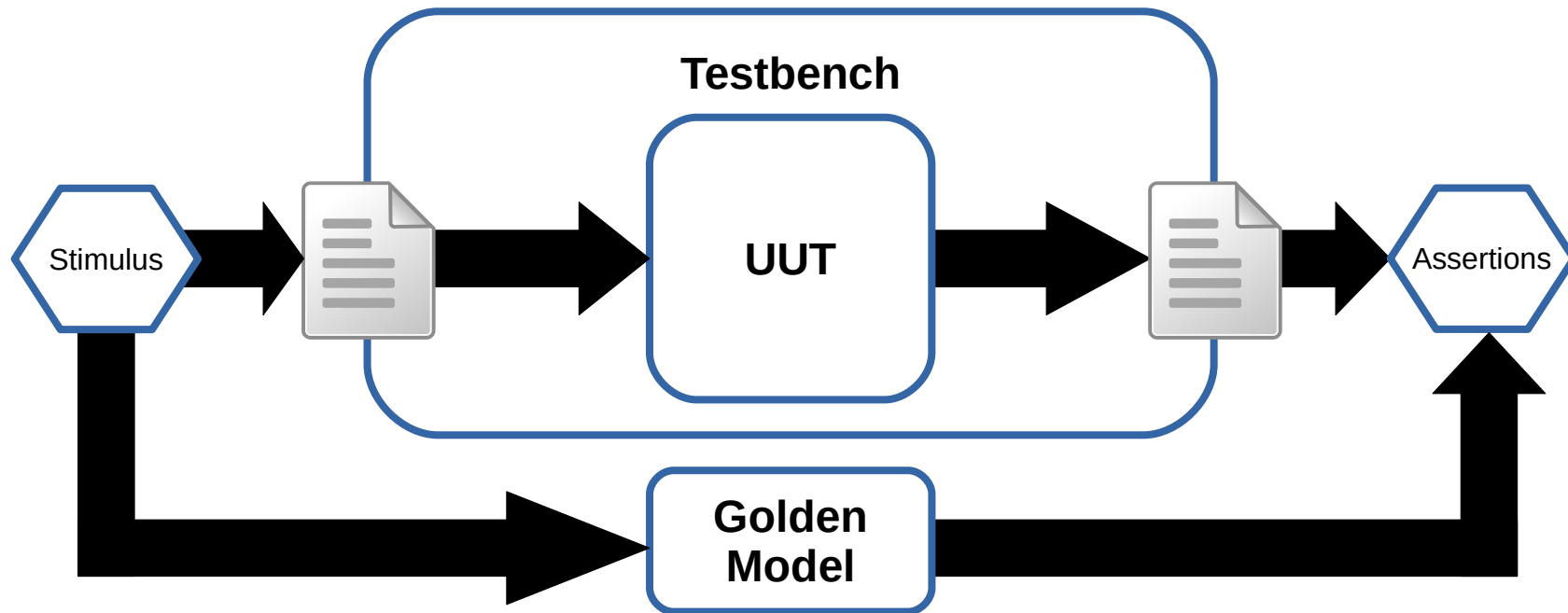


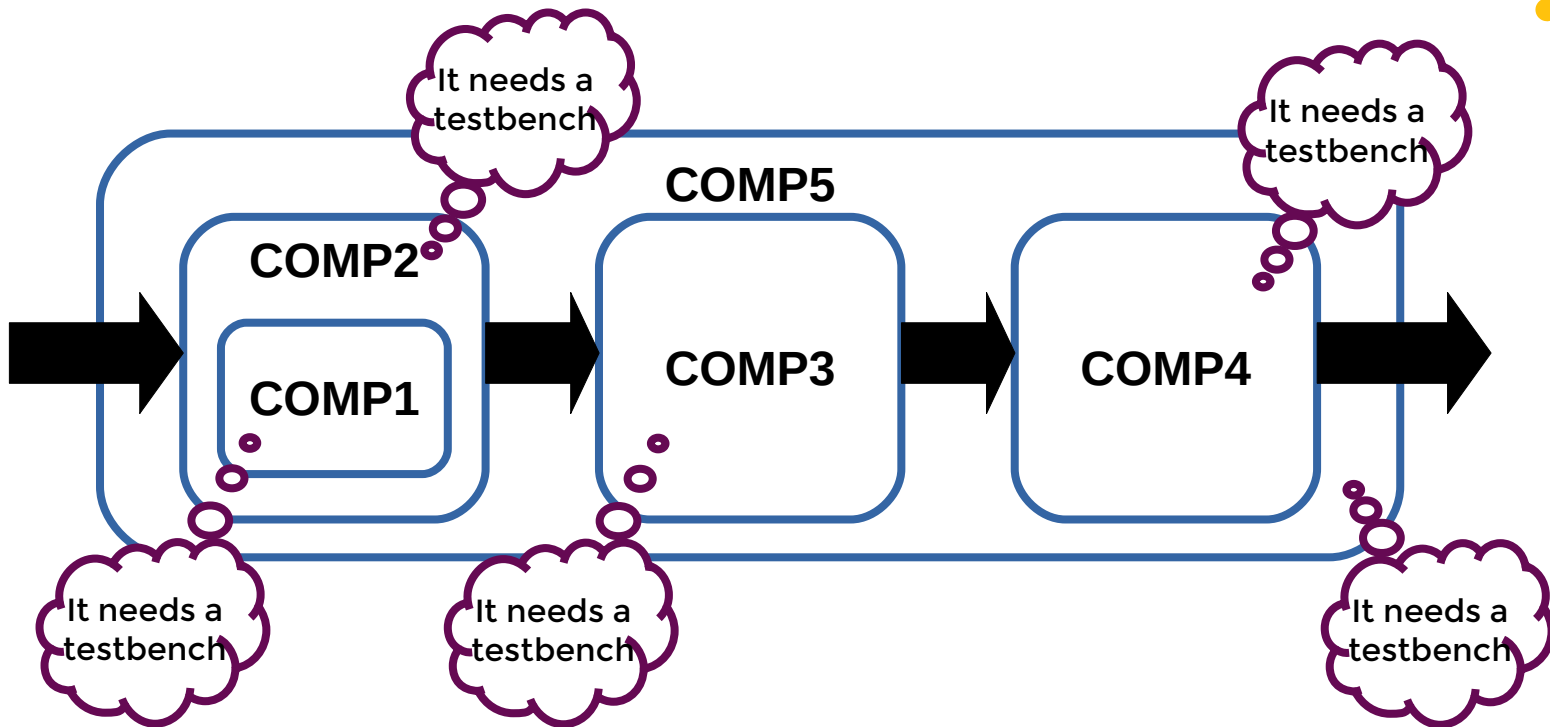
VHDL, Verilog or
Python (**cocotb**)

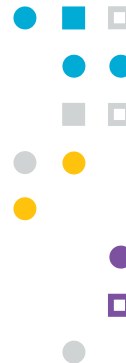












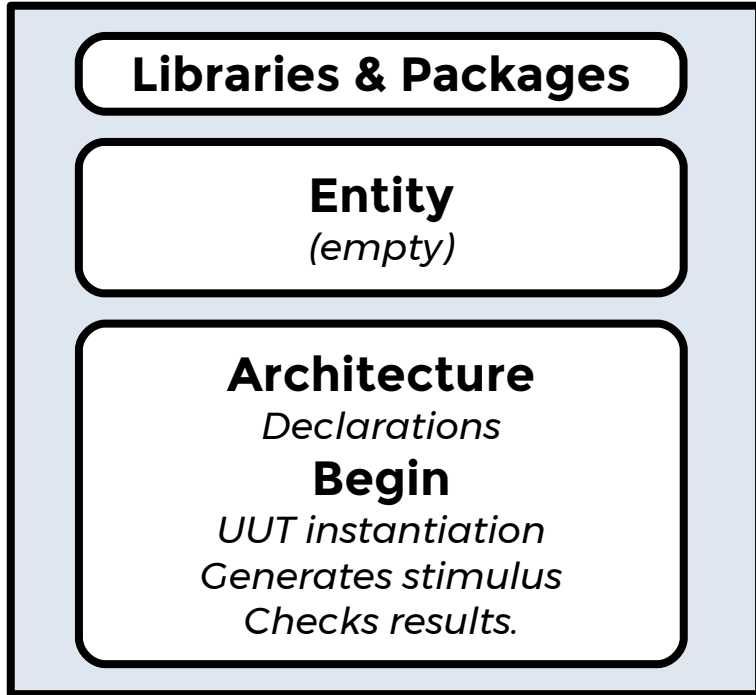
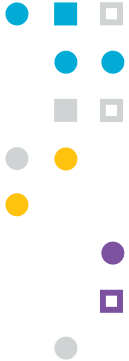
- **UVM:** Universal Verification Methodology
 - Open Verification Methodology
 - Functional verification using SystemVerilog
 - Supported by a few very expensive simulators (and Vivado?)
- **FOSS:**
 - **OSVVM:** Open Source VHDL Verification Methodology
 - **UVVM:** Universal VHDL Verification Methodology
 - **VUnit:** unit testing framework for VHDL/SystemVerilog
 - **SVUnit:** unit testing framework for Verilog/SystemVerilog
 - **Cocotb:** Python based testbenches

But we will learn how to
create a basic VHDL
testbench from scratch.

VHDL for simulation



Structure of a VHDL testbench



- The **Entity** is employed only to define the name of the testbench.
- Only one **Architecture**.



```
architecture Simul of counter_tb is
    constant PERIOD : time := 20 ns; -- 50 MHz
    signal clk      : std_logic := '1';
    signal rst      : std_logic;
    ...
    signal stop     : boolean := FALSE;
begin

    do_clock: process
    begin
        while not stop loop
            wait for PERIOD/2;
            clk <= not clk;
        end loop;
        wait; -- Event Starvation
    end process do_clock;

    rst <= '1', '0' after 3*PERIOD;
```

- time is a pre-defined physical type. It allows you to specify fs, ps, ns, us, ms, sec, min and hr.
- You need to produce **Event Starvation** (no more events) to finish the simulation. It is achieved by a **wait** without options.

```
wait [on signals] [until condition] [for time];
```

```
-- wait on signals;  
wait on s1, s2; -- wait for an event  
-- wait until condition;  
wait until clk_i = '1'; -- wait for an event  
-- wait for time;  
wait for 10 ns;  
-- wait;  
wait; -- wait forever (event starvation)
```

- Is a sequential statement.
- Used by processes **without** a sensitivity list.



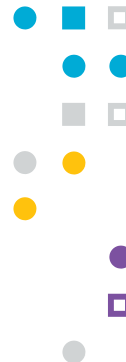


```
report <message_string> [severity <severity_level>];  
assert <condition> [severity <severity_level>];  
assert <condition> report <message_string> [severity <severity_level>];
```

- Report is a sequential statement (only inside of a process).
- The <severity_level> can be note (default for report), warning, error (default for assert) or failure.
- Assert can be either, a sequential or a concurrent statement.
- <condition> is a boolean value which must be TRUE to avoid the report.
- You can only report a string. The value of a **signal** or **variable** is not a string. You need to know the data **type** and use the image attribute:

```
report "unexpected value. i = " & integer'image(i);
```

Example - Component



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

entity counter is
    port (
        clk_i : in  std_logic;
        rst_i : in  std_logic;
        cnt_o : out std_logic_vector(3 downto 0)
    );
end entity counter;
```

```
architecture RTL of counter is
    constant MODULE : positive := 12;
    signal cnt       : unsigned(3 downto 0);
begin
    do_counter : process (clk_i)
    begin
        if rising_edge(clk_i) then
            if rst_i = '1' then
                cnt <= (others => '0');
            else
                if cnt < MODULE-1 then
                    cnt <= cnt + 1;
                else
                    cnt <= (others => '0');
                end if;
            end if;
        end if;
    end process do_counter;

    cnt_o <= std_logic_vector(cnt);
end architecture RTL;
```

Example – Testbench (part 1)



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
library std; -- Standard library of VHDL
use std.textio.all; -- we will write to STDOUT
library COUNTER_LIB; -- Our LIBRARY
use COUNTER_LIB.COUNTER_PKG.all; -- Our PACKAGE

entity counter_tb is
end entity counter_tb;

architecture Simul of counter_tb is
    constant PERIOD : time := 20 ns; -- 50 MHz
    signal clk       : std_logic := '1';
    signal rst       : std_logic;
    signal cnt       : std_logic_vector(3 downto 0);
    signal stop      : boolean := FALSE;
begin
```

```
do_clock: process
begin
    while not stop loop
        wait for PERIOD/2;
        clk <= not clk;
    end loop;
    wait; -- Event Starvation
end process do_clock;

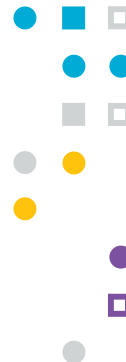
rst <= '1', '0' after 3*PERIOD;

-- UUT instantiation
 uut : counter
port map(
    clk_i => clk,
    rst_i => rst,
    cnt_o => cnt
);
```



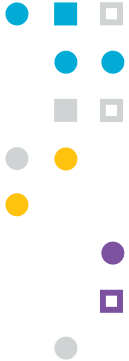
```
testing: process
  variable L : LINE; -- buffer
begin
  -- Print to STDOUT
  write(L,NOW); -- Current simulation time
  write(L,STRING'("--> Start of test"));
  writeline(output,L); -- Write to STDOUT
  -- Test of the initial value
  wait until rising_edge(clk);
  wait until rst='0';
  assert unsigned(cnt)=0
    report "Error! not 0"
    severity failure;
  -- Test of the intermediate values
  for I in 0 to 11 loop
    wait until rising_edge(clk);
    assert unsigned(cnt)=I
      report "Error! cnt = (&integer'image(to_integer(unsigned(cnt))))&"
      severity failure;
  end loop;
  wait until rising_edge(clk);
```

Example – Testbench (part 3)



```
-- Test cycle restart
assert unsigned(cnt)=0
    report "Error! Mod-12 ('&integer'image(to_integer(unsigned(cnt))))&)"
        severity failure;
-- Print to STDOUT
write(L,NOW); -- Current simulation time
write(l,string("-> End of test"));
writeln(output,L); -- Write to STDOUT
-- Clock stop
stop <= true;
wait;
end process testing;

end architecture Simul;
```



- Defines read and write procedures to work with FILES.
- The procedures supports the types `bit`, `bit_vector`, `boolean`, `character` (ex: 'A'), `string` (ex: "ICTP", defined from 1 to 4), `integer`, `real` and `time`.

```
procedure READLINE(file F: TEXT; L: out LINE);
procedure READ(L: inout LINE; VALUE: out <type>);
procedure READ(L: inout LINE; VALUE: out <type>; GOOD : out BOOLEAN);

procedure WRITE(
  L :inout LINE; VALUE : in <type>;
  JUSTIFIED: in SIDE := right;
  FIELD: in WIDTH := 0
);
procedure WRITELINE(file F : TEXT; L : inout LINE);
```



```
stimulus: process
  file F      : TEXT open READ_MODE is "input.dat";
  variable L  : LINE;
  variable tag : string(1 to 3);
  variable int : integer;
  variable ok  : boolean;
begin
  while not endfile(F) loop
    readline(F, L); -- F can be replaced by input (read from STDIN)
    read(L, tag, ok);
    assert ok report "Read ERROR!" severity failure;
    -- Do something with tag (the read value)
    read(L, int, ok);
    assert ok report "Read ERROR!" severity failure;
    -- Do something with int (the read value)
  end loop;
  wait; -- event starvation
end process stimulus;
```



```
checks: process
  file F: TEXT open WRITE_MODE is "output.dat";
  variable L: LINE;
begin
  ...
  WRITE(L, NOW);
  WRITE(L, STRING("Your string")); -- This cast is needed for strings
  WRITELINE(F,L); -- F can be replaced by output (print to STDOUT)
  ...
  wait; -- event starvation
end process checks;
```

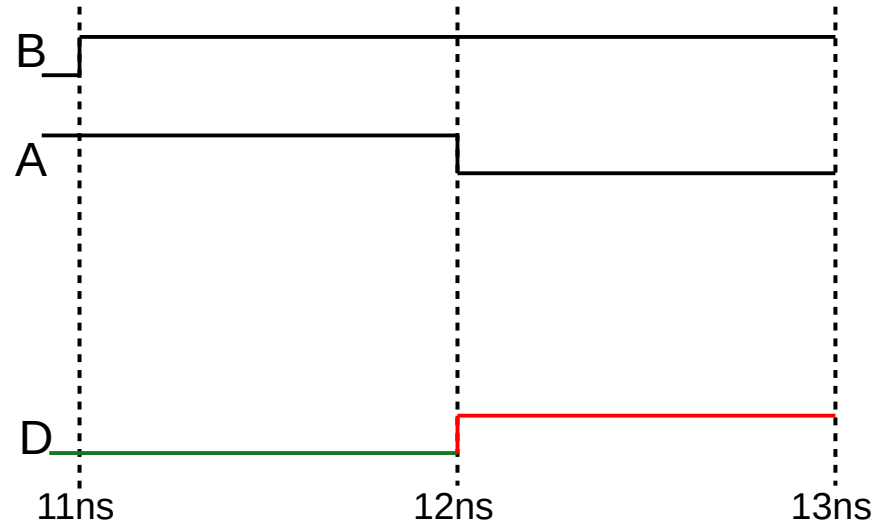
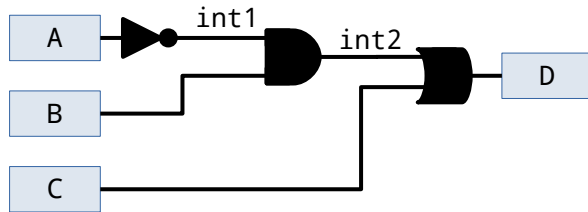
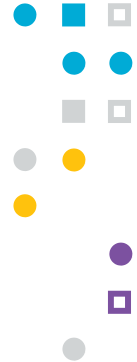


You can read from **STDIN** with a file called **input** and to write to **STDOUT** with a file called **output** (these files are automatically opened when you include **textio**).

Simulators

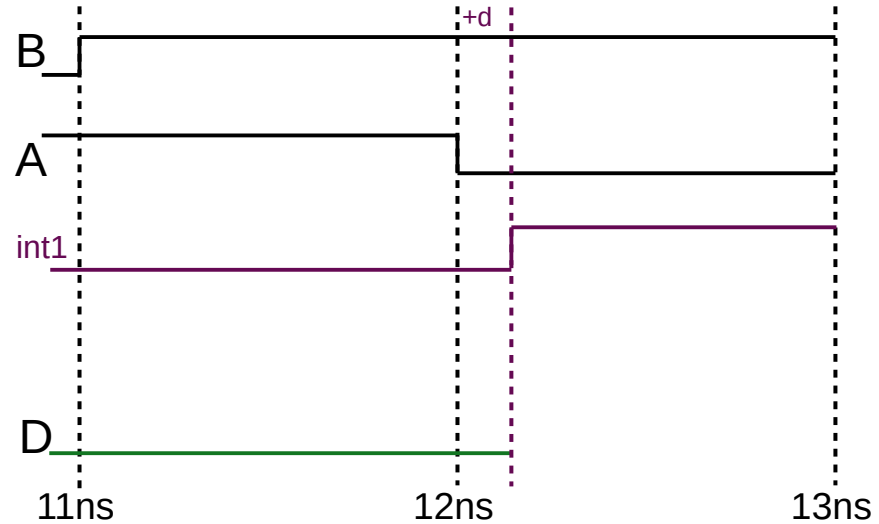
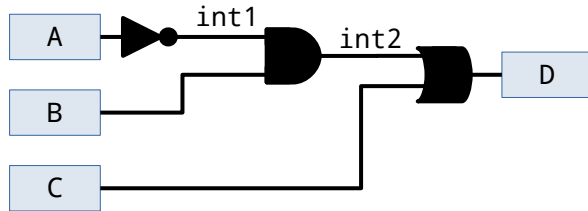
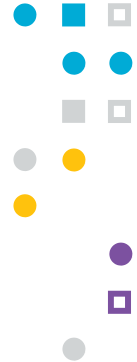


How internally works a simulator?



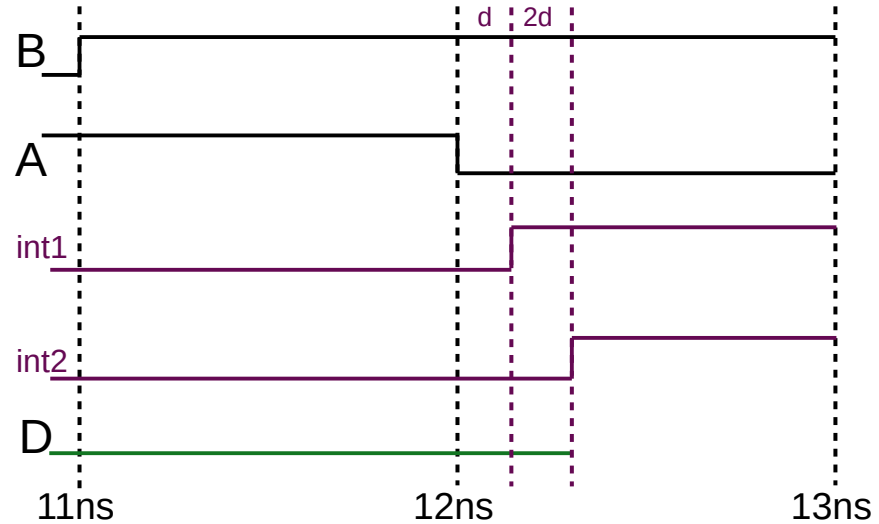
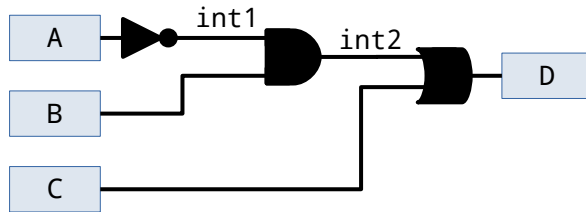
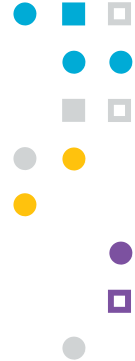
```
architecture rtl of example is
    signal int1, int2 : std_logic;
begin
    int1 <= not A;
    int2 <= int1 and B;
    D    <= int2 or C;
end architecture rtl;
```

How internally works a simulator?



```
architecture rtl of example is
    signal int1, int2 : std_logic;
begin
    int1 <= not A;
    int2 <= int1 and B;
    D    <= int2 or C;
end architecture rtl;
```

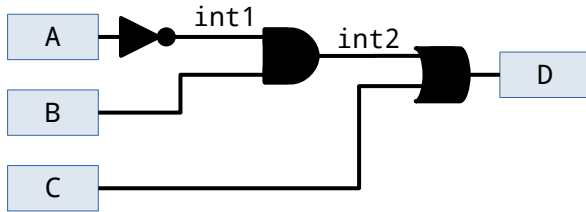
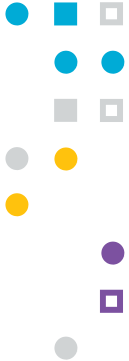
How internally works a simulator?



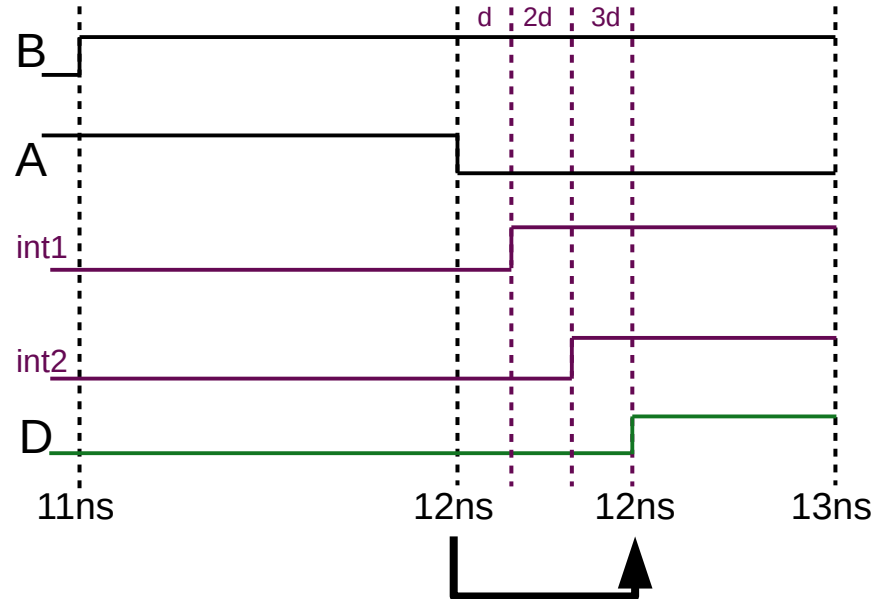
```

architecture rtl of example is
    signal int1, int2 : std_logic;
begin
    int1 <= not A;
    int2 <= int1 and B;
    D    <= int2 or C;
end architecture rtl;
  
```

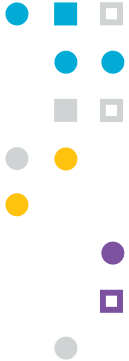
How internally works a simulator?



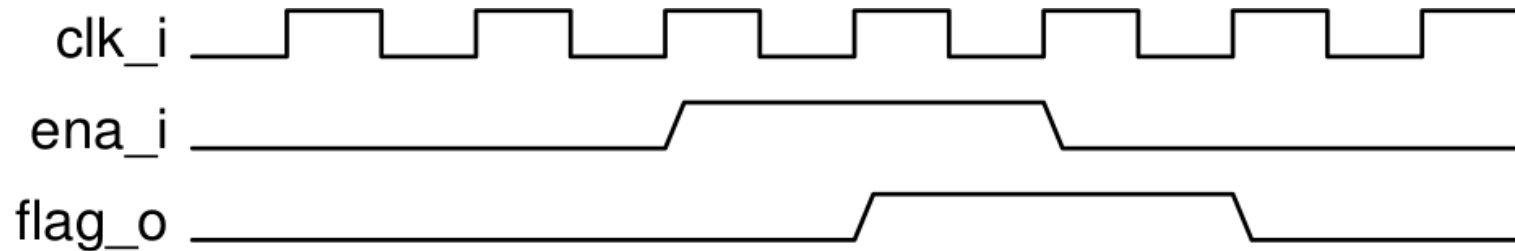
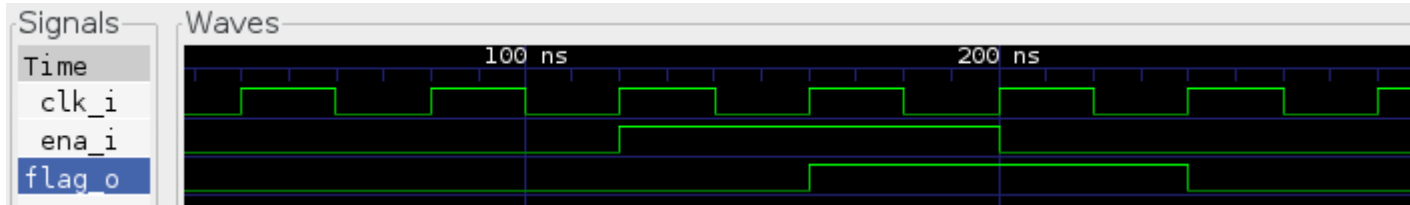
```
architecture rtl of example is
    signal int1, int2 : std_logic;
begin
    int1 <= not A;
    int2 <= int1 and B;
    D    <= int2 or C;
end architecture rtl;
```



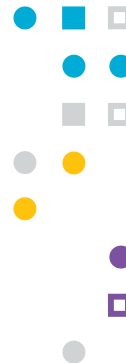
Waveforms interpretation

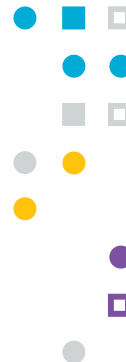


```
if rising_edge(clk_i) then
  flag_o <= '0';
  if ena_i = '1' then
    flag_o <= '1';
  end if;
end if;
```



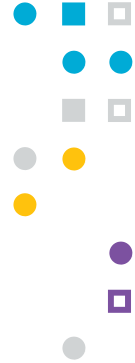
- `counter.vhdl`: entity/component
- `counter_pkg.vhdl`: package which contains the component
 - The package name is `COUNTER_PKG` (name defined into the VHDL file)
- `counter_tb.vhdl`: testbench of the component
 - The library name is `COUNTER_LIB` (name defined by the tool)



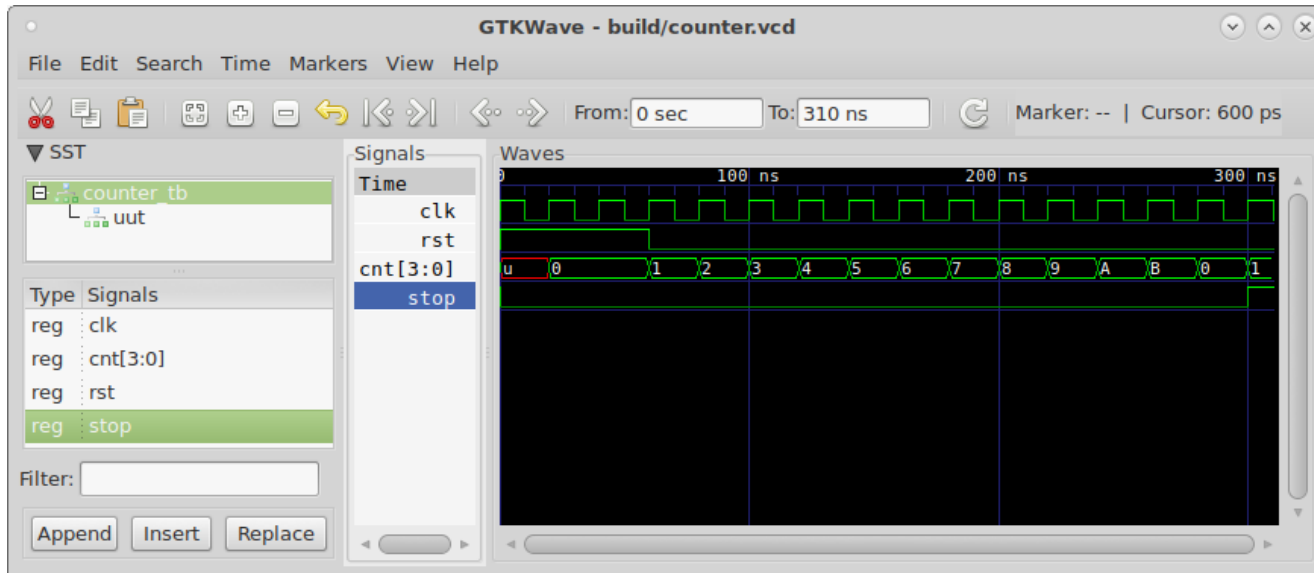


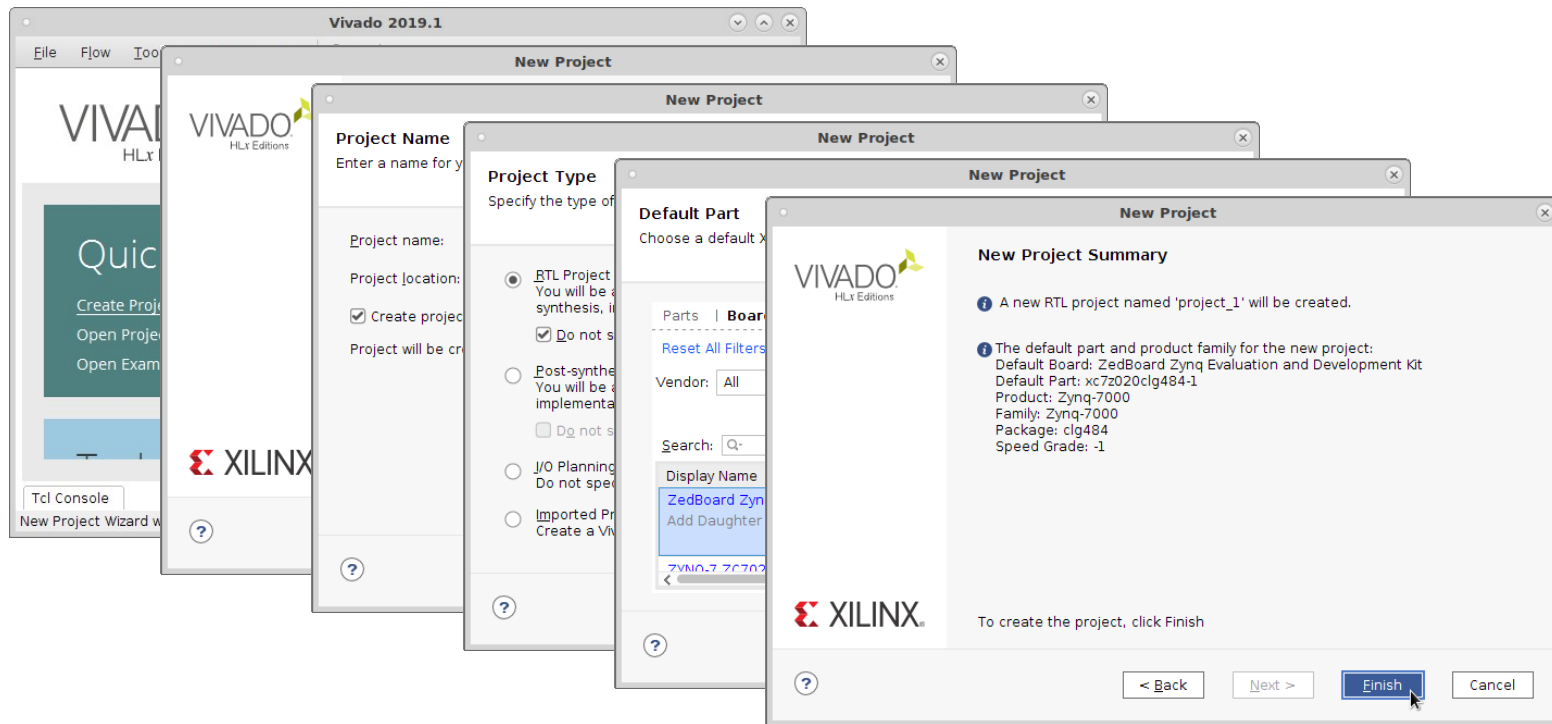
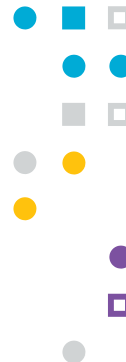
```
$ mkdir -p build
$ ghdl -a --workdir=build --work=COUNTER_LIB counter.vhdl counter_pkg.vhdl
$ ghdl -a --workdir=build -Pbuild counter_tb.vhdl
$ ghdl -e --workdir=build -Pbuild counter_tb
$ ghdl -r --workdir=build -Pbuild counter_tb --vcd=build/counter.vcd
```

- GHDL analyze (-a), elaborate (-e) and run (-r) our simulation.
- Use --workdir to specify where to put generated files (build directory).
- Use --work to specify the LIBRARY NAME (COUNTER_LIB).
- Use -P to specify where to find libraries (no space between P and the directory).
- Use --vcd or --wave (.ghw), which are runtime options, to specify where to dump waveforms.



```
$ gtkwave build/counter.vcd
```



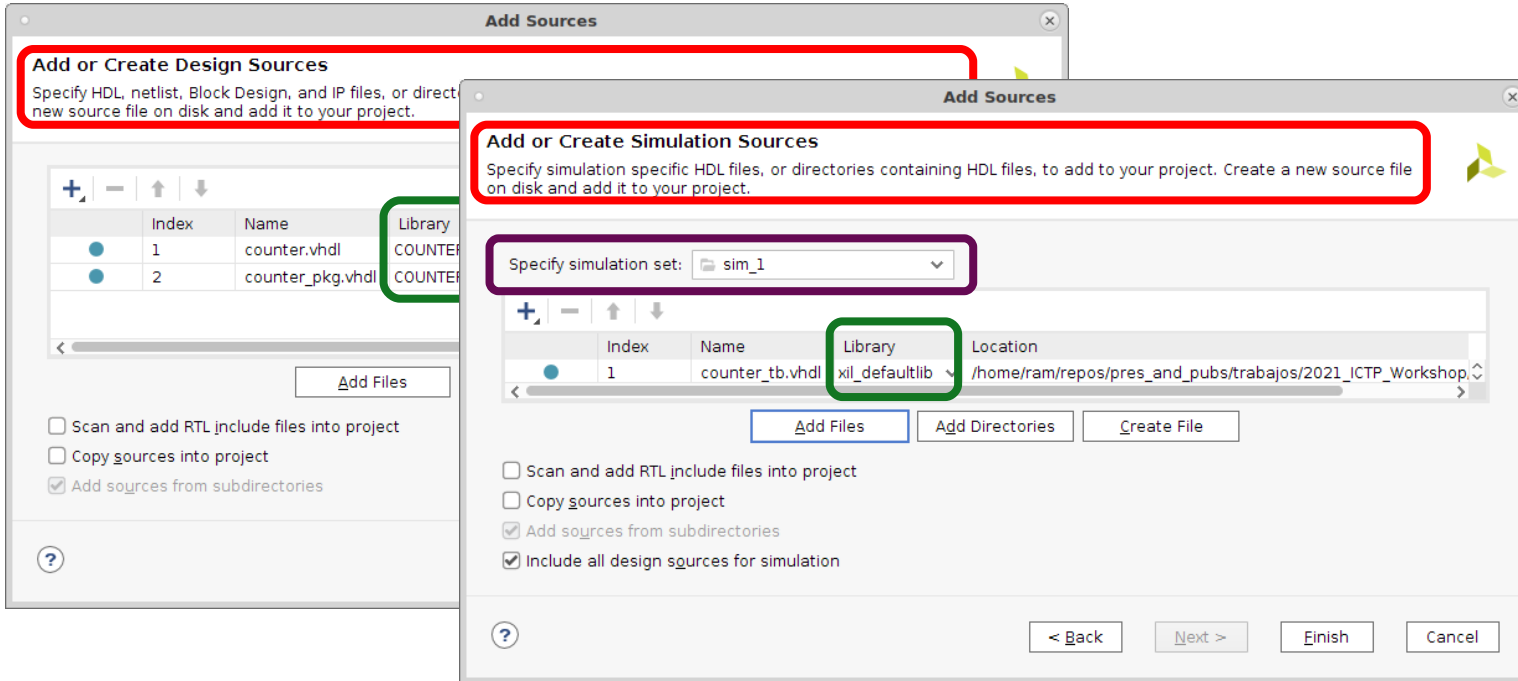
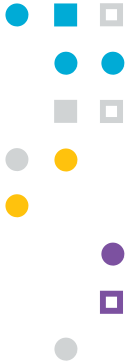


The image displays a series of overlapping 'New Project' dialog boxes in Vivado 2019.1, illustrating the steps to create a project:

- Project Name:** Enter a name for your project.
- Project Type:** Specify the type of project:
 - RTL Project: You will be performing synthesis, implementation, and simulation.
 - Post-synthesis: You will be performing simulation on a design that has already been synthesized and implemented.
 - I/O Planning: Do not specify a default part.
 - Imported Project: Create a Vivado project from an existing design.
- Default Part:** Choose a default part for the project.
 - Parts: Board
 - Reset All Filters
 - Vendor: All
 - Search: [Q]
 - Display Name: ZedBoard Zynq
 - Add Daughter
- New Project Summary:** A new RTL project named 'project_1' will be created. The default part and product family for the new project are:
 - Default Board: ZedBoard Zynq Evaluation and Development Kit
 - Default Part: xc7z020clg484-1
 - Product: Zynq-7000
 - Family: Zynq-7000
 - Package: clg484
 - Speed Grade: -1

To create the project, click Finish

Buttons: < Back, Next >, Finish, Cancel



Add or Create Design Sources
Specify HDL, netlist, Block Design, and IP files, or direct new source file on disk and add it to your project.

Index	Name	Library
1	counter.vhdl	COUNTER
2	counter_pkg.vhdl	COUNTER

Scan and add RTL include files into project
 Copy sources into project
 Add sources from subdirectories

Add or Create Simulation Sources
Specify simulation specific HDL files, or directories containing HDL files, to add to your project. Create a new source file on disk and add it to your project.

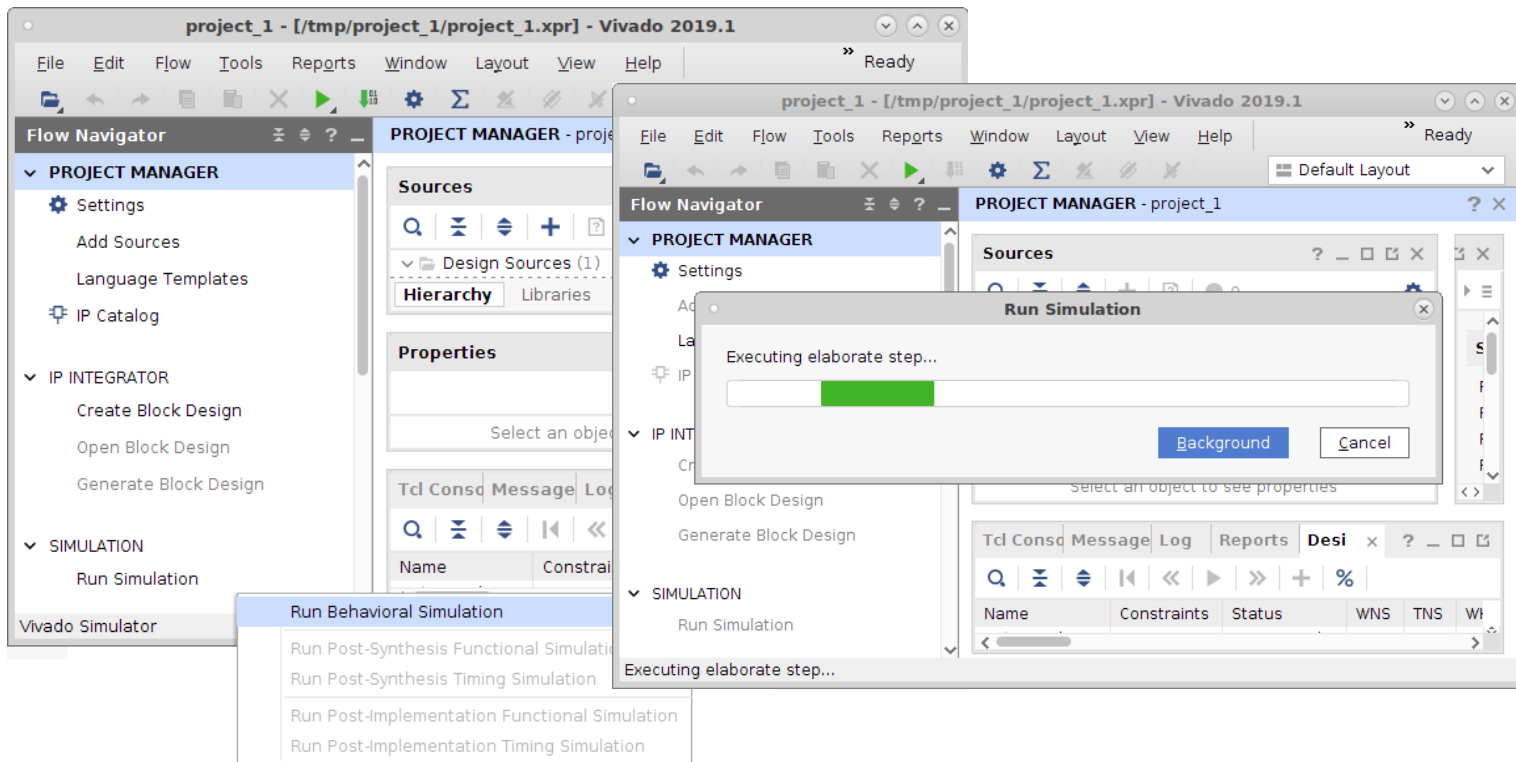
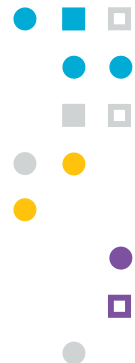
Specify simulation set: **sim_1**

Index	Name	Library	Location
1	counter_tb.vhdl	xil_defaultlib	/home/ram/repos/pres_and_pubs/trabajos/2021 ICTP_Workshop

Scan and add RTL include files into project
 Copy sources into project
 Add sources from subdirectories
 Include all design sources for simulation

< Back Next > Finish Cancel

Vivado simulation – Launch simulation



The image shows two overlapping windows of Vivado 2019.1. The background window displays the PROJECT MANAGER interface with the 'SIMULATION' section expanded. The foreground window shows the 'Run Simulation' dialog box with a progress bar and 'Background' and 'Cancel' buttons. A context menu is open over the 'Run Simulation' button in the background window, listing various simulation options.

Run Simulation

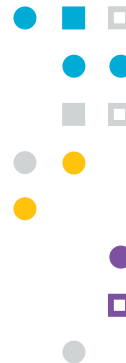
Executing elaborate step...

Background Cancel

Run Behavioral Simulation

- Run Post-Synthesis Functional Simulation
- Run Post-Synthesis Timing Simulation
- Run Post-Implementation Functional Simulation
- Run Post-Implementation Timing Simulation

Vivado simulation – See waveforms



project_1 - [/tmp/project_1/project_1.xpr] - Vivado 2019.1

File Edit Flow Tools Reports Window Layout View Run Help Q: Quick Access Ready

10 us Default Layout

Flow Navigator

- PROJECT MANAGER
 - Settings
 - Add Sources
 - Language Templates
 - IP Catalog
- IP INTEGRATOR
 - Create Block Design
 - Open Block Design
 - Generate Block Design
- SIMULATION**
 - Run Simulation
- RTL ANALYSIS
 - Open Elaborated Design
- SYNTHESIS
 - Run Synthesis
 - Open Synthesized Design
- IMPLEMENTATION
 - Run Implementation

SIMULATION - Behavioral Simulation - Functional - sim_1 - counter_tb

Name	Value
clk	0
rst	0
cnt[3]	1
stop	TRUE
PERIC	2000

Untitled 2*

Tcl Console

```

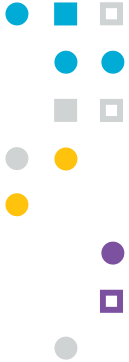
300 ns-> End of test
INFO: [USF-XSim-96] XSim completed. Design snapshot 'counter_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:08 . Memory (MB): peak = 6749.535 ; gain = 0.000 ; free physical = 175 ; free virtu
Type a Tcl command here
    
```

Sim Time: 1 us

Conclusions



- Do not perform a testbench can be only allowed for a very basic (a counter, a ROM) descriptions included in another simulated description. **Professional advice.**
- What we saw today is enough to develop a small testbench with stimulus and assertions.
- Also, you should be capable of read/write files.





rmelo@inti.gov.ar



[rodrigoalejandromelo](https://www.linkedin.com/in/rodrigoalejandromelo)



[@rodrigomelo9ok](https://twitter.com/rodrigomelo9ok)



[rodrigomelo9](https://plus.google.com/rodrigomelo9)



[rodrigomelo9](https://github.com/rodrigomelo9)





Thank you

This work is licensed under CC BY 4.0



If you want to know more about INTI, we wait for you at

 INTIArg

 @INTIargentina

 INTI

 @intiargentina

 canalinti

www.inti.gob.ar

consulta@inti.gob.ar

0800 444 4004

