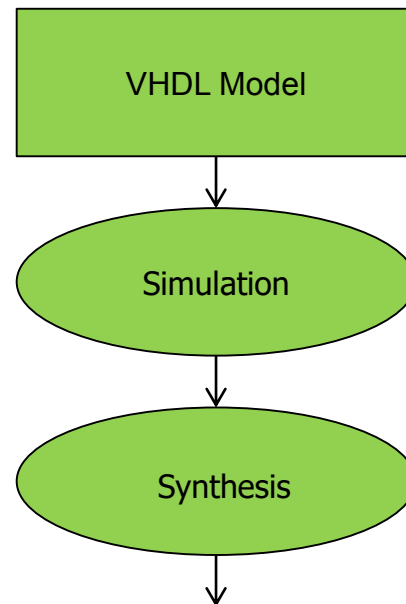


# VHDL Modeling

Stefano Spellini

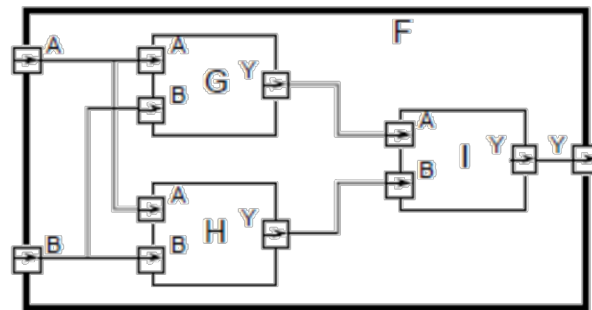
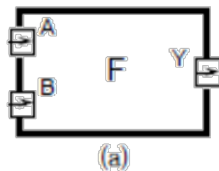
# VHDL

- Standard for all stages of hardware design *You can synth and get an implementation, starting from VHDL Models*
  - United States Government's Very High Speed Integrated Circuits (VHSIC) program, initiated in 1980
- Different levels of abstraction
- Mixture of description at different levels
- Can be used to specify both data and control path
- Describe the specification of:
  - inputs
  - synthesis results
- For logic and high-level synthesis tools.



# VHDL Description

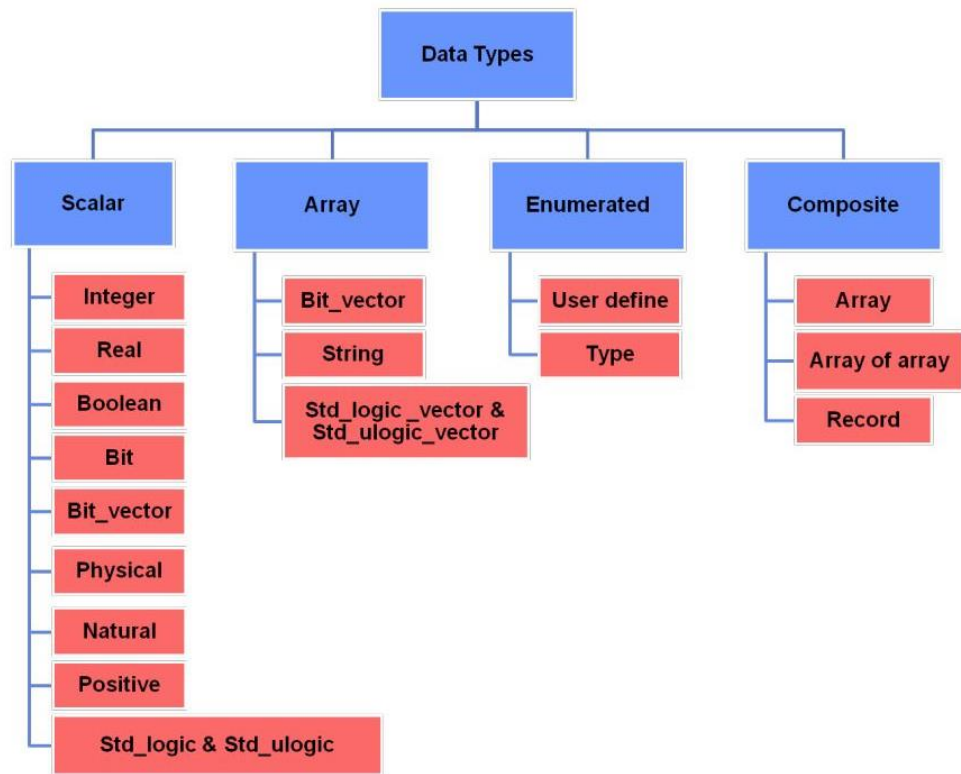
- Allows to describe:
  - Structural description
    - Structure of a design
    - How it is decomposed into sub-designs
  - Behavioral description
    - Specification of the function of designs
- Allows a design to be simulated before being manufactured
  - Compare alternatives and test for correctness without hardware prototyping



# VHDL Types

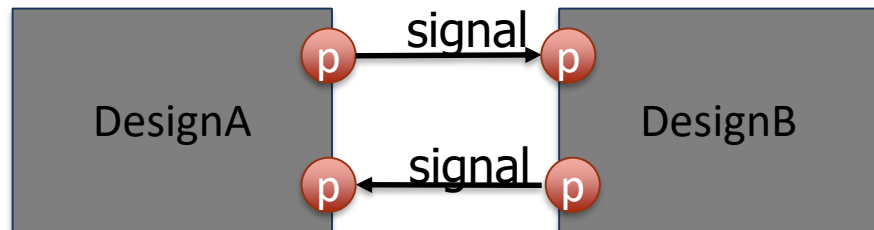
- VHDL types
  - Types for systems modelling
  - 2 values ('0','1')
  - 7 values ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-')
  - Arbitrary size integer (Signed/Unsigned)
  - Fixed point types

Type	Description
<b>bit</b>	Simple bit with 2-values
<b>bit_vector</b>	Arbitrary size 2-values vector
<b>signed</b>	Arbitrary size signed integer
<b>unsigned</b>	Arbitrary size unsigned integer
<b>std_logic</b>	Simple bit with 7-values
<b>std_logic_vector</b>	Arbitrary size 4-values vector
<b>sfixed</b>	Arbitrary sized signed fixed point
<b>Ufixed</b>	Arbitrary sized unsigned fixed point

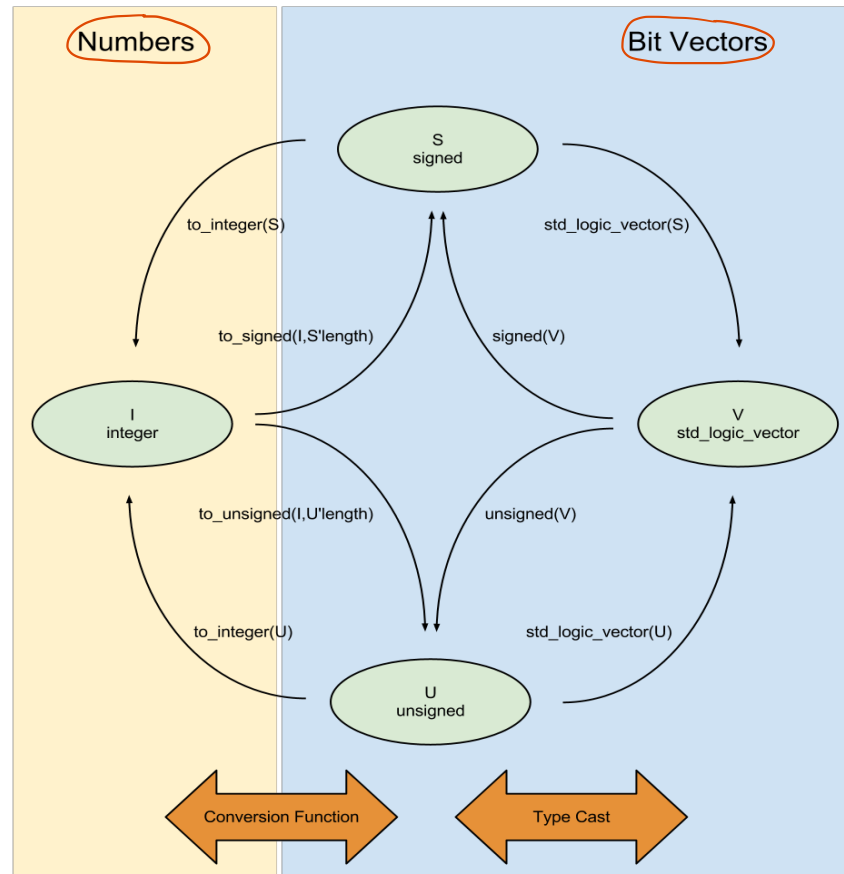


# Ports and Signals

- Ports of a module are the external interfaces that pass information to and from a module
  - In VHDL one port can be IN, OUT or INOUT
- Signals are used to connect module ports allowing modules to communicate
- Types of ports and signals:
  - All VHDL Data types
  - User defined types
- How to declare
  - Input: `<port_name> : IN <port_type>`
  - Output: `<port_name>: OUT <port_type>`
  - Bi-Directional : `<port_name>: INOUT<port_type>`



# VHDL Type Conversion



# VHDL device description

- Package
  - Collection of declarations and objects
- Entity
  - Interface of the module
- Architecture
  - Functionality implementation

# Package

- Package

- Facility for collecting declarations and objects into modular units
- Data abstraction and information hiding
  - E.g., define some constants (describing, for example, the FSM states)

```
PACKAGE root_pack IS
    CONSTANT SIZE : INTEGER := 32;
    CONSTANT Reset_ST : INTEGER := 0;
    CONSTANT ST_0 : INTEGER := 1;
    CONSTANT ST_1 : INTEGER := 2;
    CONSTANT ST_2 : INTEGER := 3;
    CONSTANT ST_3 : INTEGER := 4;
    CONSTANT ST_4 : INTEGER := 5;
END root_pack;
```



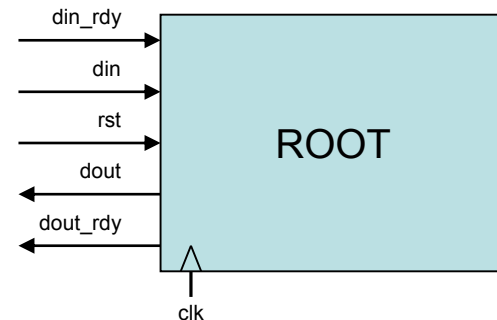
# Entity

- Entity

- Interface

- Each module has a set of ports which constitute its interface to the outside world

```
entity root is      port (  
    clk, rst        : in  bit;  
    din             : in  UNSIGNED (SIZE-1 DOWNT0 0);  
    din_rdy         : in  bit;  
    dout            : out UNSIGNED (SIZE-1 DOWNT0 0);  
    dout_rdy        : out  bit  ) ;  
end root;
```



# Architecture

- Architecture
  - Implementations of the entity
    - Behavioral Style
    - Structural Style
    - Dataflow Style
    - Mixed Style
  - Signals and components may be declared here and used to construct a structural description

*This is behavioral because we have processes*

```


architecture behav of root is
  subtype status_t is integer range 0 to 5;
  subtype internal_t is UNSIGNED (SIZE-1 DOWNT0 0);
  signal STATUS:      status_t;
  signal NEXT_STATUS: status_t;
  signal Root:        internal_t;
  signal Number:       internal_t;
  signal Counter:      internal_t;
  signal Rem_s:        internal_t;
  CONSTANT ZERO :    internal_t :=
    "00000000000000000000000000000000";
  CONSTANT SEDICI : internal_t :=
    "000000000000000000000000000010000";

  begin
    process (STATUS, din_rdy, din, Counter)
      ...
      process (clk, rst)
        ...
      end process;
    end behav;
  
```

# VHDL Behavioral: Processes

- Process
  - Used to model sequential circuits
  - Primary unit of behavioral description in VHDL
  - Sequential body of code which can be activated in response to changes in state
    - When more than one process is activated at the same time, they execute concurrently
  - Concurrent statement which can be used in an architecture body or block

## Sensitivity List



```
process (clk, rst)
begin
  ...
end process;


...
end root;
```

# VHDL Behavioral: Processes

- Execution

- Activated initially during the initialization phase of simulation
- Executes all the sequential statements, and then repeats, starting again with the first statement
  - A process may be suspended by executing a wait statement
  - The sensitivity list of the wait statement specifies a set of signals to which the process is sensitive while it is suspended

## Sensitivity List



```
process (clk, rst)
begin
...
end process;
```

## Empty Sensitivity List

```
process
begin
...
wait (on signal(s), until
condition(s), for time);
end process;
```

# VHDL process styles

1. Processes with a *sensitivity list including all read signals* and assigning all signals and variables in all conditional branches
  - The sensitivity list will contain also all the intermediate signals
  - **The process must be reevaluated every time one of the inputs to the circuit it models changes**
    - **Combinational logics**

# VHDL process styles

2. Processes with a sensitivity list including all read signals and *assigning all variables in all conditional branches*
  - Models a mixture of pure combinational logic and asynchronous registers
  - Registers are inferred when signals are not assigned in a conditional branch

# VHDL process styles

## 3. Processes with a *wait statement* for detecting clock edges

- Clocked circuit
  - Synchronous sequential machine
- Used to represent Finite State Machines (FSM)

# VHDL process styles

4. Processes with a *sensitivity list including a clock signal* and optionally an asynchronous reset signal
  - An if statement constitutes the process, sensitive to the clock (and reset) events



# VHDL Dataflow: Concurrent Assignments

- Used to describe combinational circuits
- Basic mechanism: **concurrent assignment**
  - Assignment of a value to a signal
- Assignment operations involve the basic Boolean functions (operators):

- and

- or

- xor

- nand

- nor

- xnor

```
entity AND_gate is
port (a: in std_logic;
      b: in std_logic;
      c: out std_logic
);
```

```
architecture AND_gate_dataflow_arch of AND_gate is
begin
    c <= a and b;
end AND_gate_dataflow_arch;
```

# VHDL Structural: Components Hierarchy

```

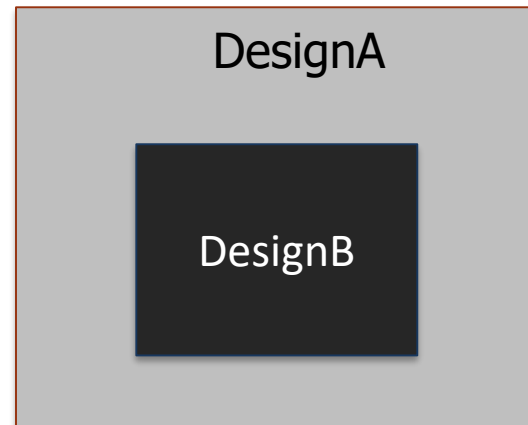
...
architecture bh of DesignA is
  signal clk_sig: bit;
  signal din_sig,dout_sig: unsigned(31 downto 0);
...
component DesignB is
  port (
    clk: bit;
    din: unsigned(31 downto 0);
    dout: unsigned(31 downto 0);
  );
end component DesignB;

begin

  dut : DesignB port map (
    clk => clk_sig,
    din => din_sig,
    dout => dout_sig);
...
end architecture;

```

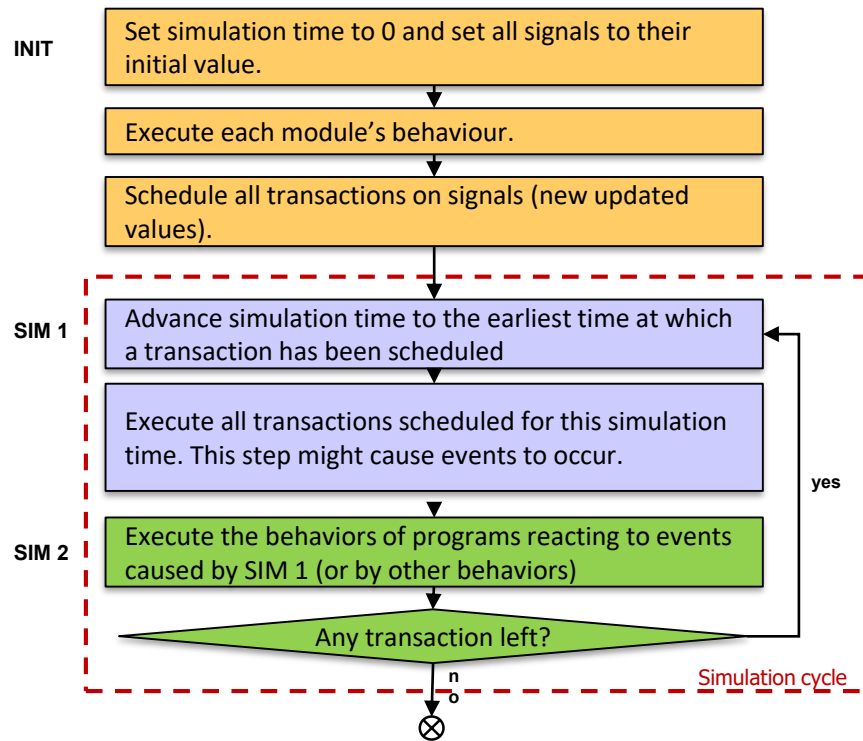
Interface  
definition  
of  
DesignB



Instantiation of  
DesignB as DUT  
(Design Under Test)

# VHDL Simulation: “scheduler”

- Same background as for SystemC
- Not a simulation kernel, but simulation tools



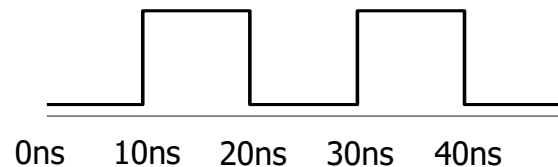
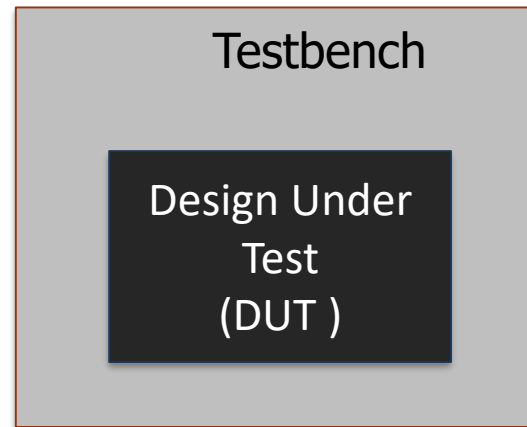
# VHDL Simulation

- Simulate the module
  - Simulating the passage of time in discrete steps
    - At some simulation time, a module input may be stimulated by changing the value on an input port
    - The module reacts by running the code of its behavioral description
    - New values to be placed on the signals connected to its output ports are scheduled
      - Scheduling a transaction on that signal
    - If the new value is different from the previous value on the signal, an event occurs
      - Other modules with input ports connected to the signal may be activated

# VHDL into Action

- How to stimulate and simulate our design?
  - Define a Testbench as Top Level
  - Testbench generates Clock and other stimuli

```
...  
architecture bh of tb is  
  
signal clk: bit;  
constant ClockPeriod : time := 20 ns;  
...  
begin  
  
clk <= not clk after ClockPeriod/2;  
...  
end architecture;
```



# Lecture Assignment

- Write your Fixed-point Multiplier by using the VHDL language
  - To stimulate the design, write a stimuli.tcl script
- Constraints:
  - Every process has to be implemented using a valid style for the synthesis
    - Combinational
    - Mixed Combinational with asynchronous latches     *Multiplier = behavioural style*
    - Sequential asynchronous
    - Sequential synchronous
  - The used data-types have to be HW datatypes for synthesis
    - Logic, bit, unsigned
- For the report
  - Justify the choices related the process style and datatypes
  - Comment the produced simulation waveforms

# Xilinx Vivado



- Xilinx inc.
  - Leading company in the programmer logic market
  - Inventor of the FPGA (1985)
  - First semiconductor company with a fabless manufacturing model

- Xilinx Inc. donated to the University of Verona:

- 5 PYNQ-Z1 FPGAs,
- 25 SDSoC licenses,

- And gave full access to their training material

- More information on the **Xilinx University Program**:

- <https://www.xilinx.com/support/university.html>

- Vivado WebPACK
  - Full HDL and System design framework
    - Supporting VHDL, Verilog, SystemVerilog, SystemC
    - Integrated editor
    - Integrated simulator
      - TCL-based scripts
    - Integrated Logic and High-level synthesis tools

# Vivado into Action

- Download the tar.gz from the elearning

- \$> tar xzf 01\_vhdl\_modeling.tar.gz

- \$> ls

- root/   dist/   div/

The directories contains three VHDL designs and stimuli.tcl

- Open Vivado

- Click on **Create New Project**

- A dialog window will appear, click on Next

- Choose the **directory** where you want to save the project

- E.g., /home/user/PSE/vivado\_projects

- Give a **name** to the project

- E.g., root

- **Tick** the «create project subdirectory» option

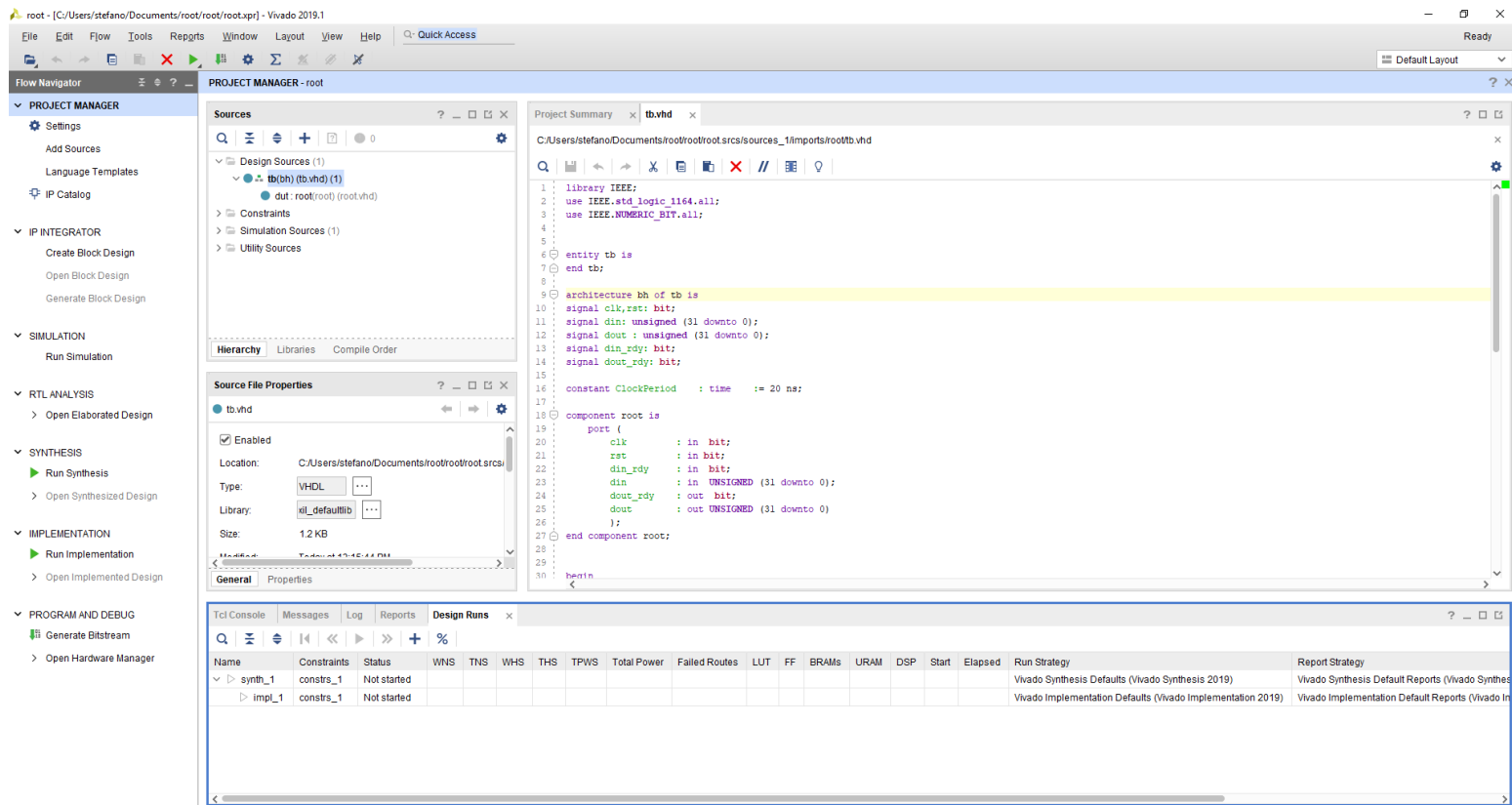
- **Next**



# Vivado into Action

- Project type: choose the **RTL Project** option
- **Next**
- **Click on the +**
  - **Add the root.vhd and tb.vhd files** in the vhdl directory you downloaded
  - **Tick all the options**
    - Scan and add RTL include files into project
    - **Copy sources into project**
  - **Be sure that the Target the Simulator Languages are set on VHDL**
  - **Next**
  - **Skip the next page (Next again)**
  - **Choose the PYNQ**
    - Family: Zynq-7000
    - Package: clg400
    - Speed grade: -1
    - **Code: xc7z020clg400-1**
  - **Finish**

# Vivado into Action



The screenshot displays the Vivado 2019.1 IDE interface. The top menu bar includes File, Edit, Flow, Tools, Reports, Window, Layout, View, and Help. The left sidebar contains the Project Manager with sections for Settings, Add Sources, Language Templates, IP Catalog, IP Integrator, Simulation, RTL Analysis, Synthesis, Implementation, and Program and Debug. The main workspace is divided into three panes: Sources, Project Summary, and a code editor.

**Sources Pane:** Shows the project structure with Design Sources (1) containing a file named tb.vhd. The Source File Properties pane for tb.vhd is open, showing it is a VHDL file of type 'VHDL' and size '1.2 KB'.

**Project Summary Pane:** Displays the file path C:/Users/stefano/Documents/root/root/srcs/sources\_1/imports/root/tb.vhd.

**Code Editor:** Contains the following VHDL code:

```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3 use IEEE.NUMERIC_BIT.all;
4
5 entity tb is
6 end tb;
7
8
9 architecture bh of tb is
10 signal clk_rst: bit;
11 signal din: unsigned (31 downto 0);
12 signal dout: unsigned (31 downto 0);
13 signal din_rdy: bit;
14 signal dout_rdy: bit;
15
16 constant ClockPeriod : time := 20 ns;
17
18 component root is
19     port (
20         clk      : in bit;
21         rst      : in bit;
22         din_rdy  : in bit;
23         din      : in UNSIGNED (31 downto 0);
24         dout_rdy : out bit;
25         dout     : out UNSIGNED (31 downto 0)
26     );
27 end component root;
28
29
30 begin

```

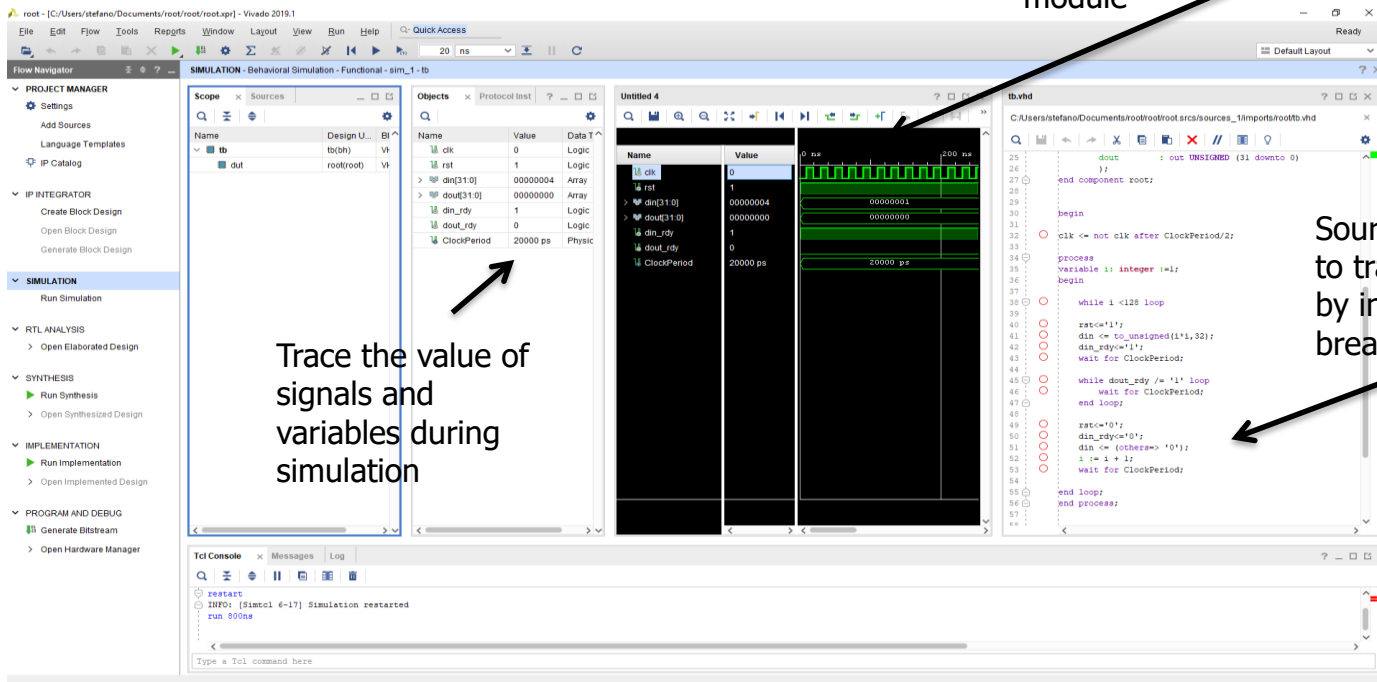
**Design Runs Table:** The bottom pane shows the Design Runs table with columns for Name, Constraints, Status, WNS, TNS, WHS, THS, TPWS, Total Power, Failed Routes, LUT, FF, BRAMs, URAM, DSP, Start, Elapsed, Run Strategy, and Report Strategy.

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed	Run Strategy	Report Strategy
synth_1	constrs_1	Not started															Vivado Synthesis Defaults (Vivado Synthesis 2019)	Vivado Synthesis Default Reports (Vivado Synthesis 2019)
impl_1	constrs_1	Not started															Vivado Implementation Defaults (Vivado Implementation 2019)	Vivado Implementation Default Reports (Vivado Implementation 2019)

# Vivado in Action

1. Click *Run Simulation*
2. Run commands on TCL Console:
  - **run 5000ns** run the simulation for 5000ns

Evolution of the internal signals and ports of the root module



The screenshot displays the Vivado IDE interface during a behavioral simulation. The left sidebar shows the PROJECT MANAGER with the SIMULATION tab selected. The main workspace is divided into several panes:

- Scope:** Lists the design units and their instances. An arrow points to the 'dut' instance.
- Objects:** A table showing the current values of signals and variables. An arrow points to the 'clk' signal.
- Waveform:** A timing diagram showing the evolution of signals over time. An arrow points to the 'clk' signal.
- Source Code:** The VHDL code for the 'root' module. An arrow points to the 'process' block.
- Tcl Console:** Shows the command 'run 5000ns' being executed.

Annotations on the image provide context for the simulation:

- Trace the value of signals and variables during simulation:** Points to the Objects pane.
- Source code allows to trace execution by inserting breakpoints:** Points to the Source Code pane.

# TCL-Script commands

- Simulate the design for <time>
  - `run <time>`
- Reset the simulation and waveform
  - `restart`
- Force a value to the port
  - `Add_force <portname> <value>`
- Get a value from a port
  - `get_value <portname>`
- Examples are available in stimuli.tcl in div and dist design folder
  - To use it remember to set the DUT as TOP level!
    - Simulation Sources -> Right click on “dut” -> set as Top
    - “Source stimuli.tcl” to launch the TCL-script

TCL/TK Tutorial at: <https://www.tutorialspoint.com/tcl-tk/index.htm>

# Vivado in Action

- Play with the simulation and its waveforms
  - Try changing the clock period: how much faster is the root component?
    - Default is 20ns, try with 10ns
    - Hint: add\_force with `-repeat_every`
- Try the other modules and their stimuli files