



Electronics Systems (938II)

Lecture 1.2

Modern Electronic Systems (intro) – SystemVerilog (intro) and simulation

Intro to HDLs

- HDLs

- Concurrent language (not sequential !!!)
 - Description of HW circuits

- VHDL → .vhd
- Verilog → .v
- SystemVerilog (extension of Verilog) → .sv

Intro to HDLs

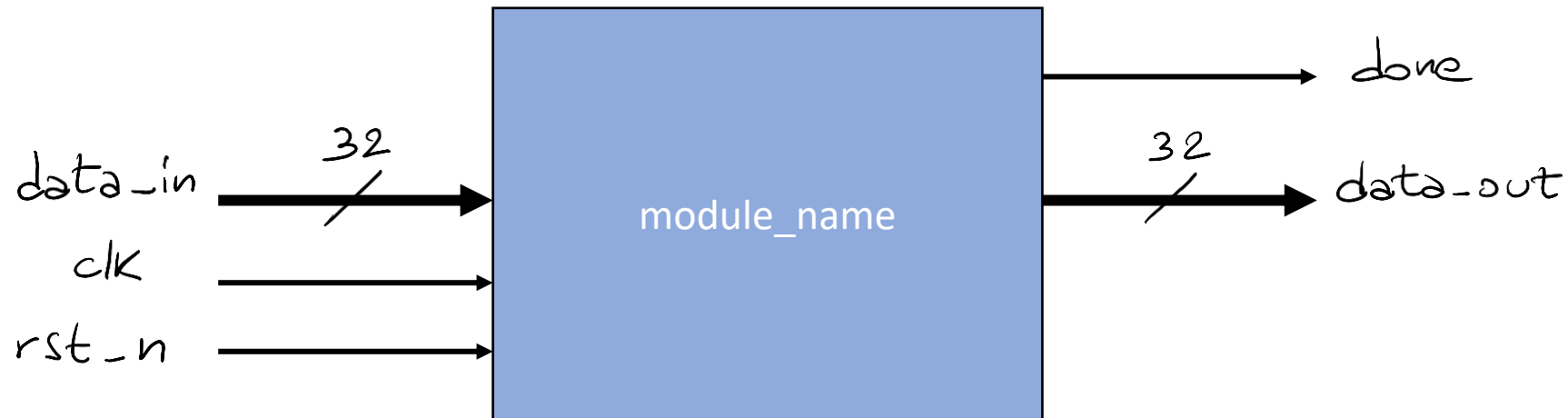
- HDLs
 - Concurrent language (not sequential !!!)
 - Description of HW circuits
 - VHDL → .vhd
 - Verilog → .v
 - **SystemVerilog** (extension of Verilog) → .sv

Intro to circuit design in SystemVerilog

- The main building block in SystemVerilog (SV) is the **module**
 - Like Verilog
 - Represents an electronic circuit or a subpart thereof
 - Defined by **ports** (input and output)

Intro to circuit design in SystemVerilog

- Example of SV module



Intro to circuit design in SystemVerilog

- Example of SV module

```
module module_name (  
    input clk  
    ,input rst_n  
    ,input [31:0] data_in  
    ,output done  
    ,output reg [31:0] data_out  
);  
  
    // Logic description  
  
endmodule
```

Intro to circuit design in SystemVerilog

- Example of SV module

Keyword module

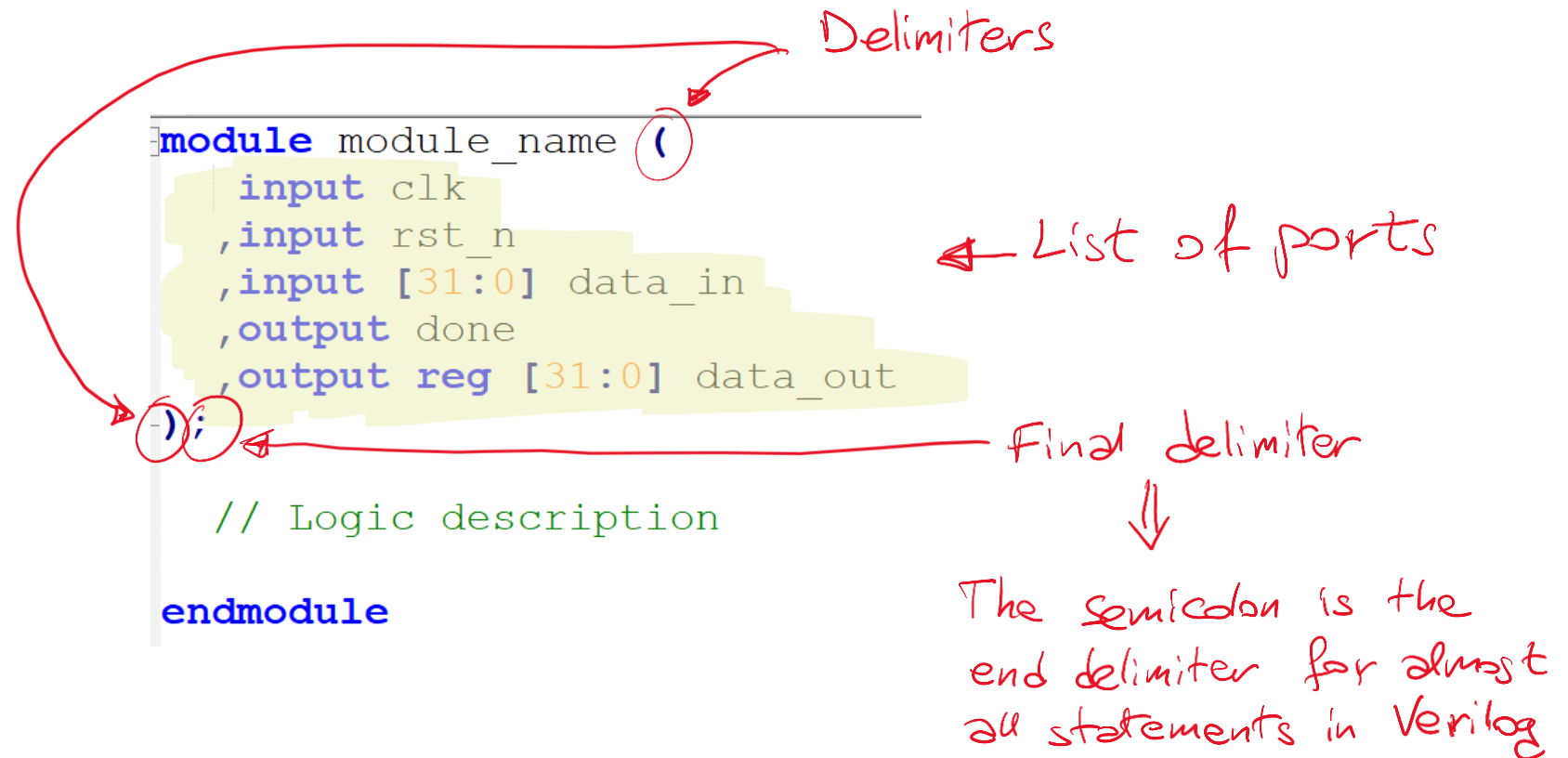
```
module module_name (  
    input clk  
    ,input rst_n  
    ,input [31:0] data_in  
    ,output done  
    ,output reg [31:0] data_out  
);  
  
    // Logic description  
  
endmodule
```

name of the module

one word
no spaces
underscore admitted

Intro to circuit design in SystemVerilog

- Example of SV module



```
module module_name (  
    input clk  
    ,input rst_n  
    ,input [31:0] data_in  
    ,output done  
    ,output reg [31:0] data_out  
);  
    // Logic description  
endmodule
```

Delimiters

List of ports

Final delimiter

The semicolon is the end delimiter for almost all statements in Verilog

Intro to circuit design in SystemVerilog

- Example of SV module
 - List of ports
 - Syntax: <polarity> [<type>] [<bit width and range>] <name>
 - Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module
 - List of ports

- Syntax: `<polarity> [<type>] [<bit width and range>] <name>`

↓
input ⇒ Also inout is admitted (for tri-state ports), but unless
output highly specialized module, you will mostly use only
input and output

- Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module

- List of ports

This is the default type, i.e., if not specified, the type of the corresponding port is wire. For this reason, the declaration of port type is optional

wire
reg

- Syntax: <polarity> [**<type>**] [<bit width and range>] <name>

input
output

- Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module

- List of ports

- Syntax: <polarity> [<type>] [<bit width and range>] <name>

↓
} input
{ output

{ wire
 reg



We are going to see in detail the meaning of wire and reg later, but I can anticipate that reg can be used for sequential logic, hence, input ports are always wire, whereas output ports can be either wire or reg

- Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module
 - List of ports

- Syntax: <polarity> [¹<type>] [<bit width and range>] <name>

↓
input
output

\Downarrow

$$\underbrace{[\text{initial index} : \text{final index}]}_{\text{bit range}} \Rightarrow \text{bit width} = |\text{initial index} - \text{final index}| + 1$$

- Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module
 - List of ports

- Syntax: <polarity> [^{wire}<type>] [^{reg}<bit width and range>] <name>

↓
 { input
 output }

↓
 [initial index: final index] ⇒ bit width = |initial index - final index| + 1
 bit range

- Ports must be separated by a **comma** (,)

For indexes you can use any natural number, you are not forced to include 0, and for the range you can use both ascending and descending order

Intro to circuit design in SystemVerilog

- Example of SV module
 - List of ports

- Syntax: <polarity> [\uparrow <type>] [\downarrow <bit width and range>] <name>

\uparrow { wire
reg

\downarrow
 { input
output

\downarrow
 [initial index : final index] \Rightarrow bit width = $| \text{initial index} - \text{final index} | + 1$
 bit range

\downarrow
 E.g.: [31:0] \rightarrow width = 32 bits
 [0:31] \rightarrow width = 32 bits
 [2:40] \rightarrow width = 39 bits
 [128:1] \rightarrow width = 128 bits

- Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module
 - List of ports

- Syntax: <polarity> [^{wire}<type>] [^{reg}<bit width and range>] <name>

\downarrow
 { input
 output

\downarrow
 $[initial\ index : final\ index] \Rightarrow bit\ width = |initial\ index - final\ index| + 1$
 bit range

- Ports must be separated by a **comma** (,)

\downarrow
 This field is optional: if not specified, the bit width is 1

Intro to circuit design in SystemVerilog

- Example of SV module

- List of ports

- Syntax: <polarity> [<type>] [<bit width and range>] <name>

↓
 { input
 output }

{ wire
 reg }

↓
 [initial index: final index] ⇒ bit width = |initial index - final index| + 1
 bit range

↑
 Arbitrary word (following the same rules as the module name)

- Ports must be separated by a **comma** (,)

Intro to circuit design in SystemVerilog

- Example of SV module

```
module module_name (  
    input clk  
    ,input rst_n  
    ,input [31:0] data_in  
    ,output done  
    ,output reg [31:0] data_out  
);
```

This is a single-line
comment (delimiter = //);
for multi-line comments
use delimiters /* and */

```
// Logic description  
endmodule
```

} This space is the
body of the module
and will contain the
description of logic
circuit

Intro to circuit design in SystemVerilog

- Example of SV module

```
module module_name (  
    input clk  
    ,input rst_n  
    ,input [31:0] data_in  
    ,output done  
    ,output reg [31:0] data_out  
);  
  
    // Logic description  
  
endmodule
```

Keyword endmodule
to declare the end of
the module

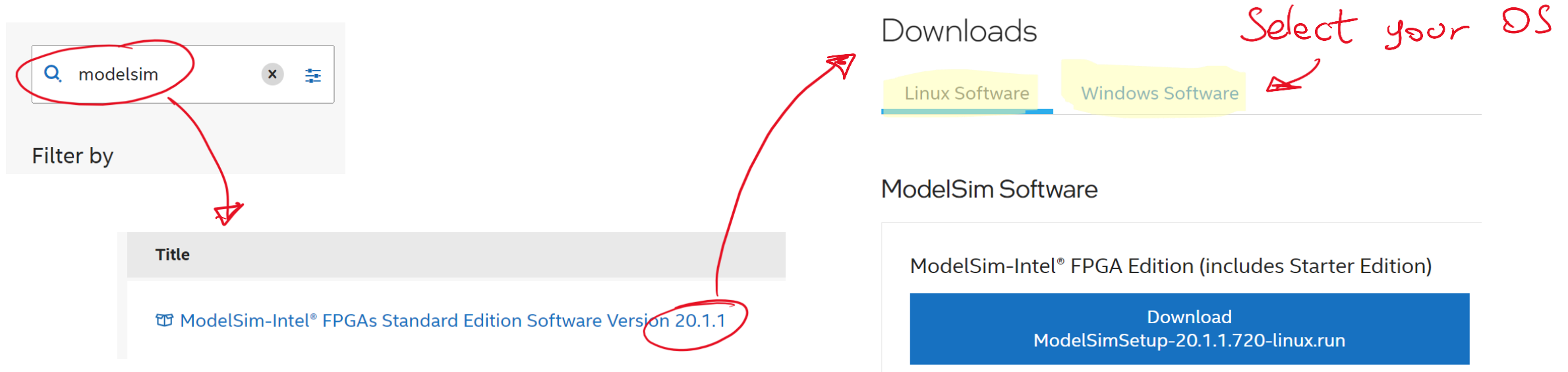
Intro to circuit design in SystemVerilog

- Modules (or sub-modules) in SV can be simulated
 - Debug and (functional) verification
 - RTL simulation
 - Zero-delay: no information about timing of logic gates (propagation delays, ...)
 - Dedicated tools
 - Example: Modelsim = used in the industry
- We are going to use **Modelsim**, ...

Intro to circuit design in SystemVerilog

..., so you can download it, for free, from the Intel FPGA Download Center

<https://www.intel.com/content/www/us/en/collections/products/fpga/software/downloads.html?q=modelsim&s=Relevancy>



The screenshot shows the Intel FPGA Download Center search results for "modelsim". The search bar contains "modelsim" and is circled in red. Below the search bar, the "Filter by" section is visible. The search results list "ModelSim-Intel® FPGAs Standard Edition Software Version 20.1.1" with a red arrow pointing to the version number. To the right, the "Downloads" section shows "Linux Software" and "Windows Software" tabs, with a red arrow pointing to the "Linux Software" tab. Below the tabs, the "ModelSim Software" section displays "ModelSim-Intel® FPGA Edition (includes Starter Edition)" and a blue "Download" button with the filename "ModelSimSetup-20.1.1.720-linux.run". A handwritten red note "Select your OS" with an arrow points to the OS selection tabs.

Search results for "modelsim":

- ModelSim-Intel® FPGAs Standard Edition Software Version 20.1.1

Downloads section:

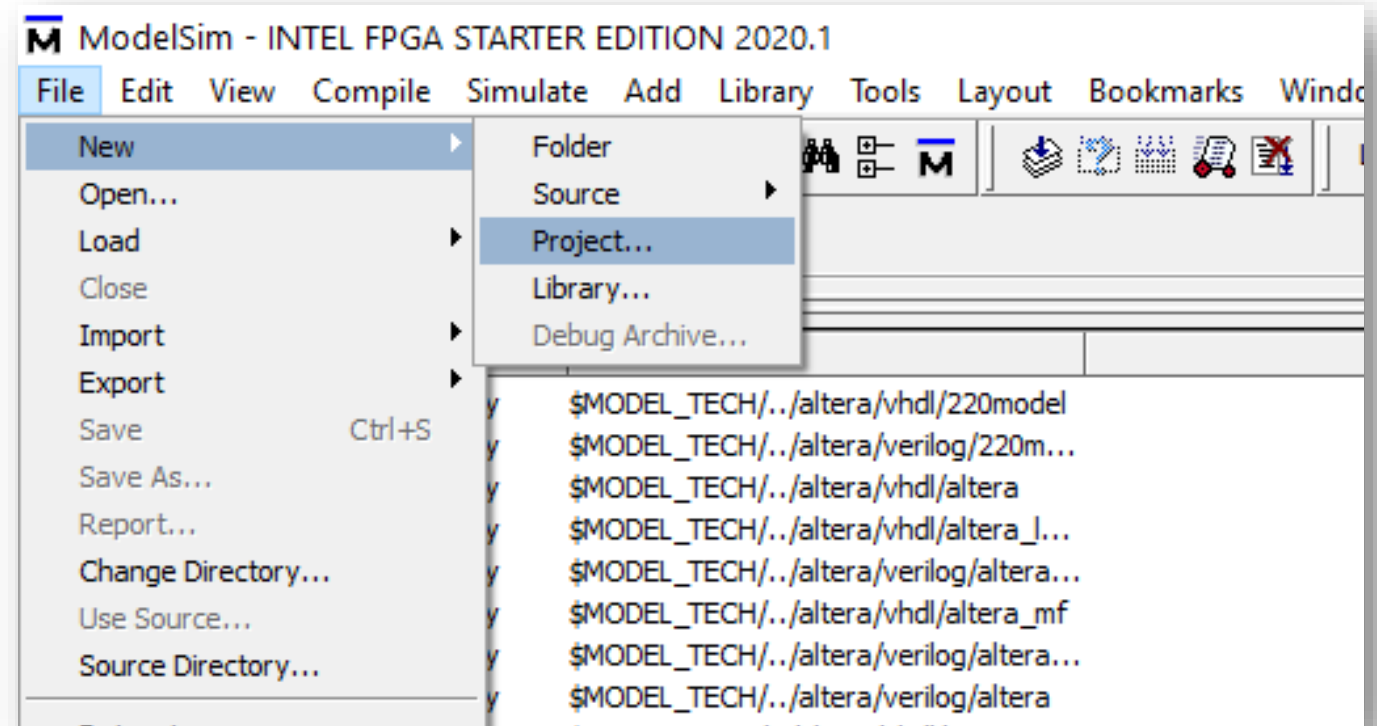
- Linux Software
- Windows Software

ModelSim Software section:

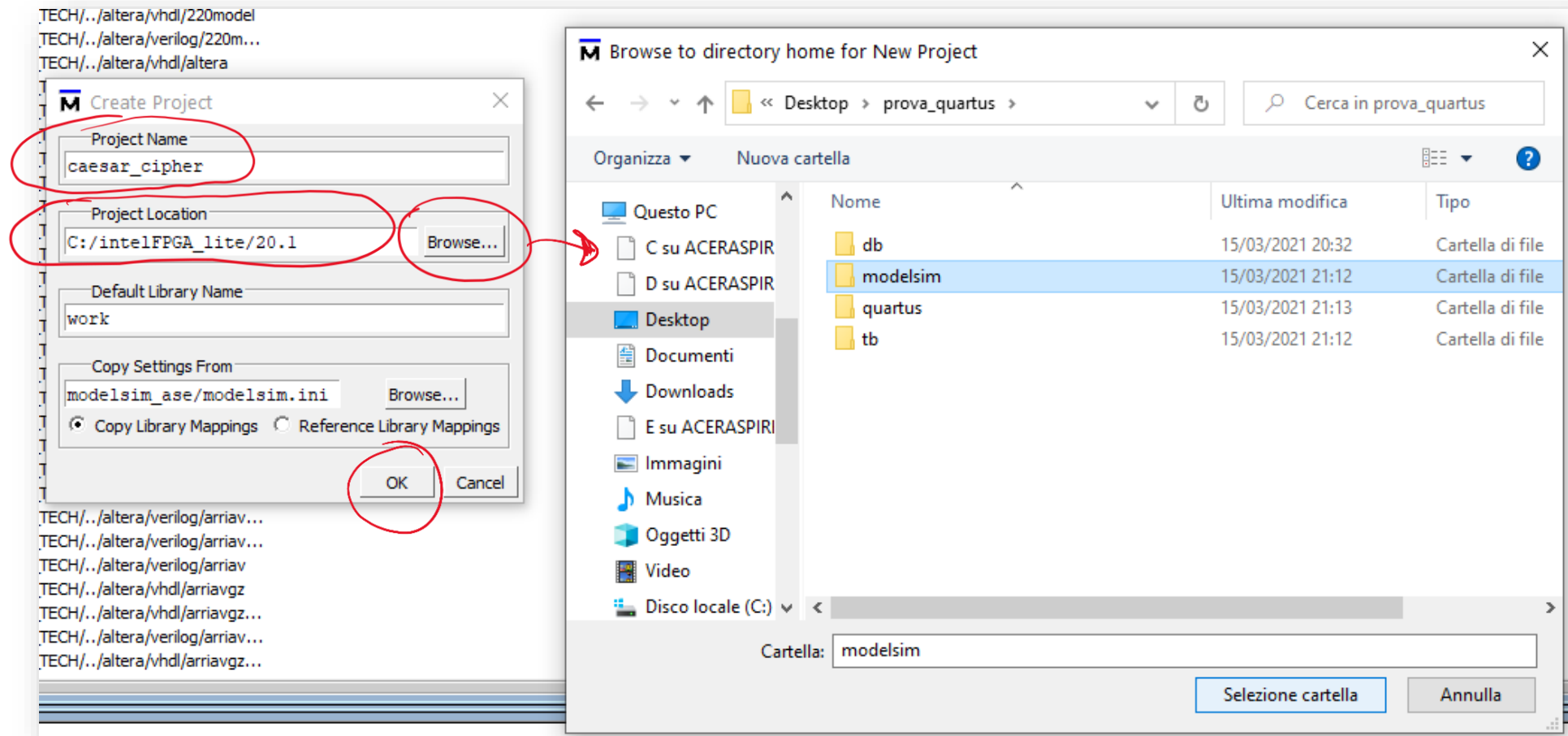
- ModelSim-Intel® FPGA Edition (includes Starter Edition)
- Download ModelSimSetup-20.1.1.720-linux.run

Quick tutorial on Modelsim

- Run Modelsim and create a new project
 - File > New > Project...

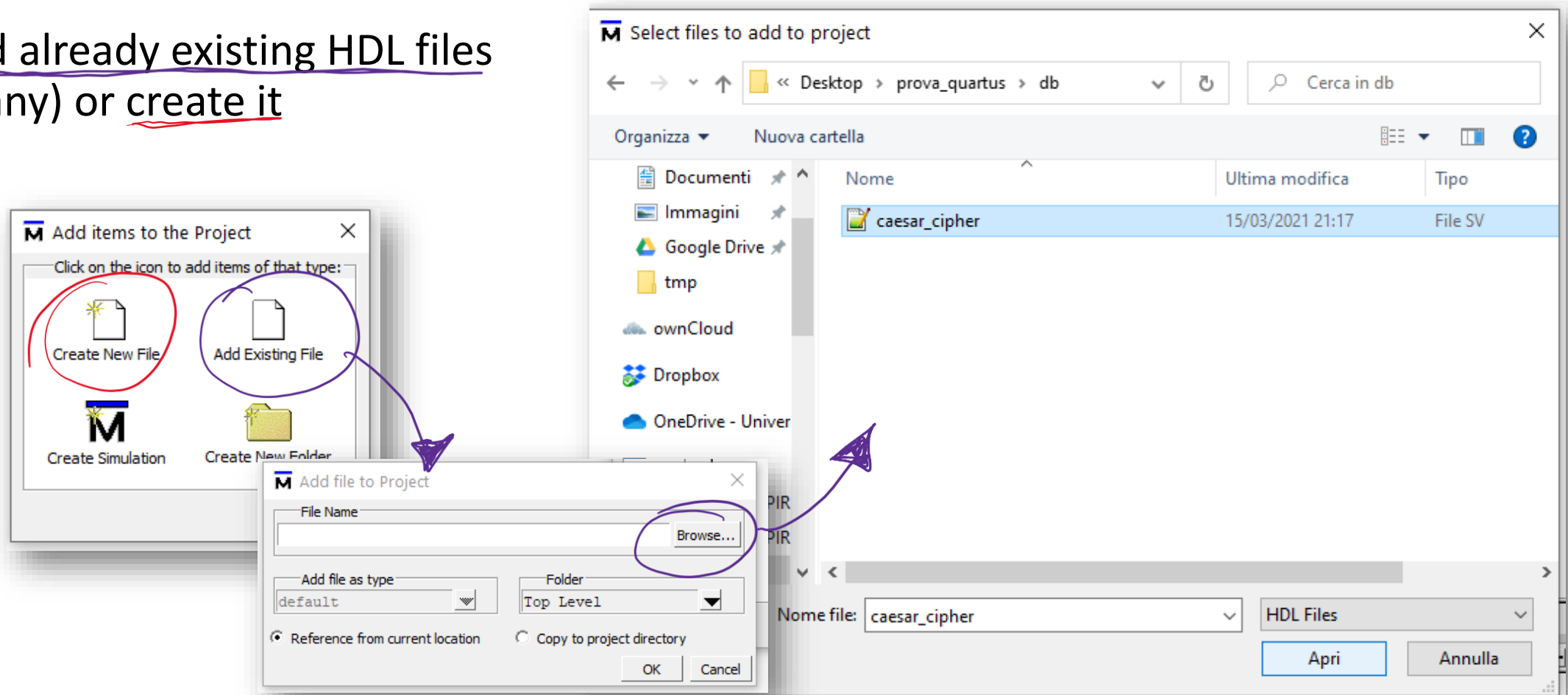


Quick tutorial on Modelsim



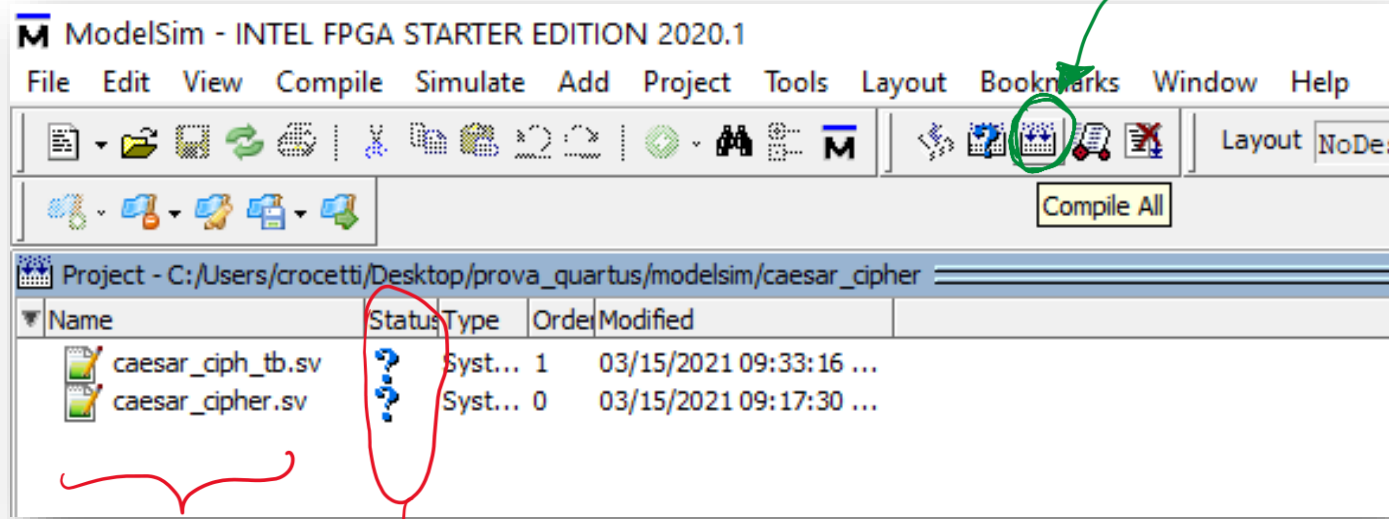
Quick tutorial on Modelsim

- Add already existing HDL files (if any) or create it



Quick tutorial on Modelsim

Compile files



Project - C:/Users/crocetti/Desktop/prova_quartus/modelsim/caesar_cipher

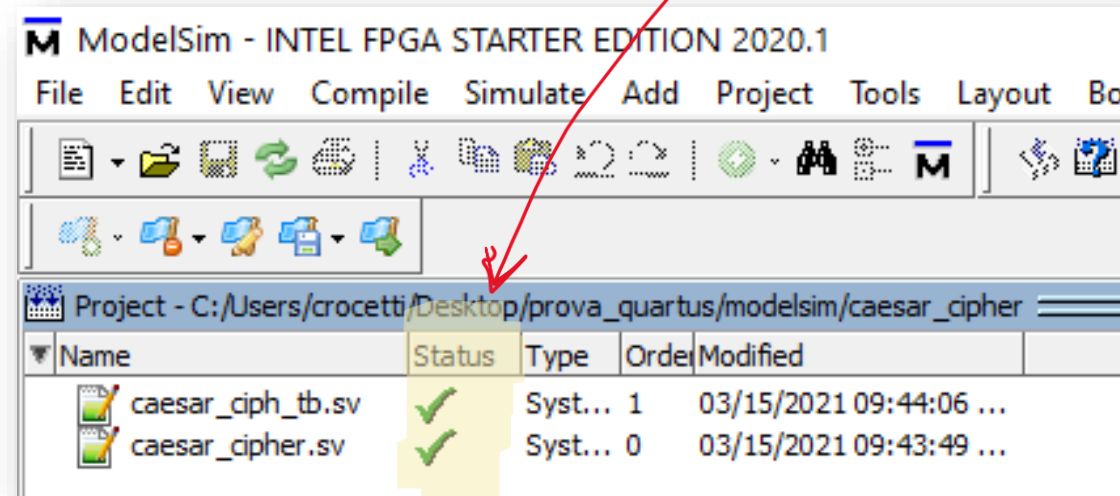
Name	Status	Type	Order	Modified
caesar_ciph_tb.sv	?	Syst...	1	03/15/2021 09:33:16 ...
caesar_cipher.sv	?	Syst...	0	03/15/2021 09:17:30 ...

List of files

Status of files: ? = not compiled

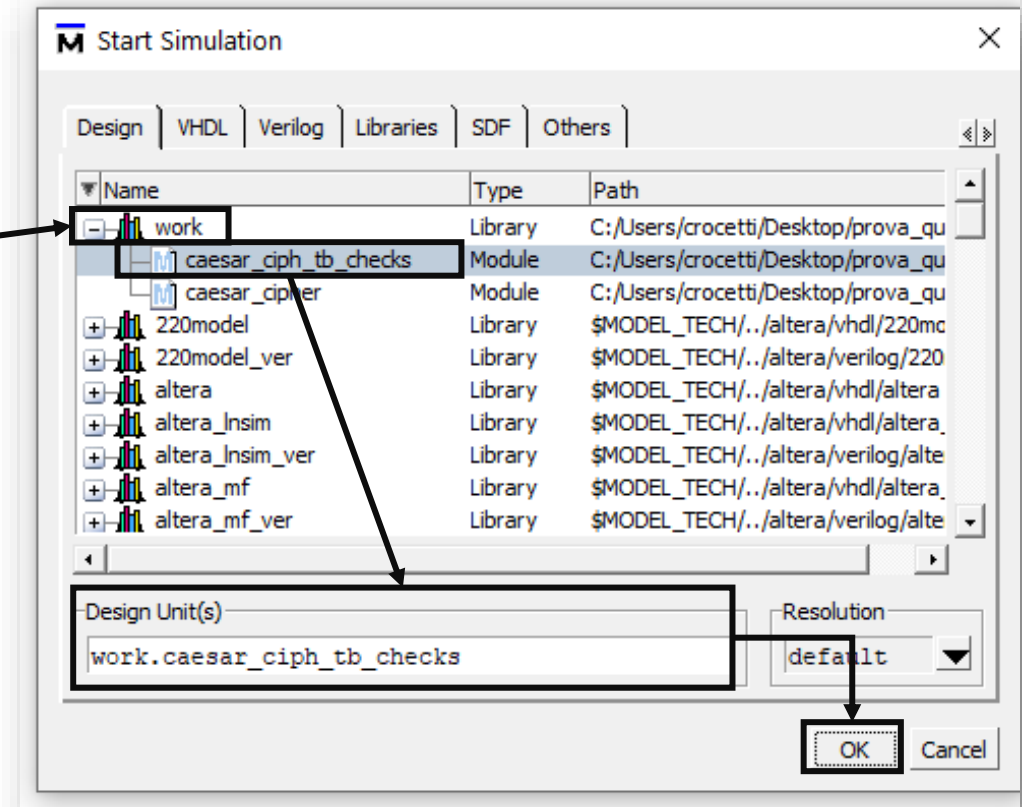
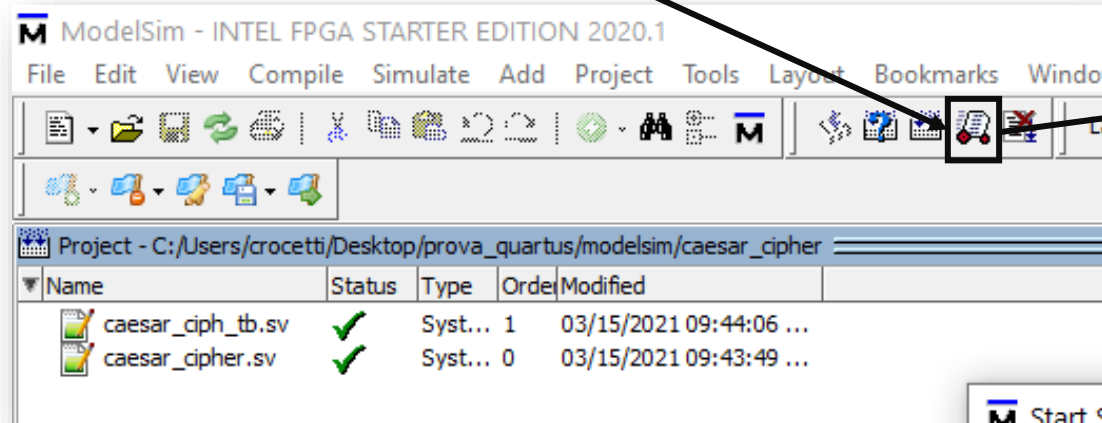
Quick tutorial on Modelsim

Compiled without errors

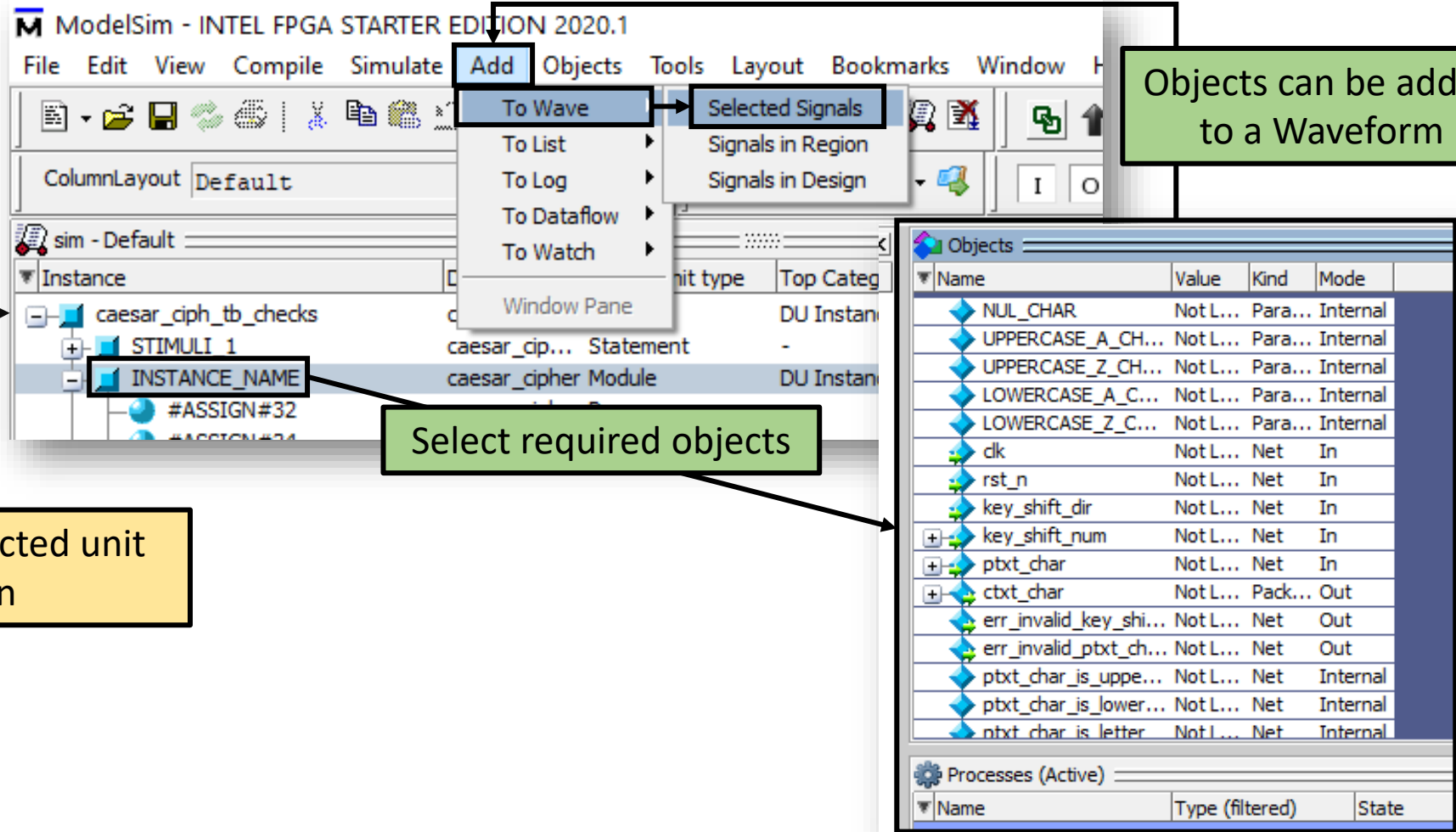


Quick tutorial on Modelsim

Launch the simulation



Quick tutorial on Modelsim



The screenshot shows the ModelSim interface. The 'Add' menu is open, showing options like 'To Wave', 'To List', 'To Log', 'To Dataflow', 'To Watch', and 'Window Pane'. The 'To Wave' option is selected, and a sub-menu is open showing 'Selected Signals', 'Signals in Region', and 'Signals in Design'. The 'Objects' window is also open, displaying a list of objects with columns for Name, Value, Kind, and Mode. The 'Hierarchy of the selected unit for simulation' is shown in the left pane, with 'INSTANCE_NAME' selected. A green box highlights the 'Objects' window, and a yellow box highlights the hierarchy pane.

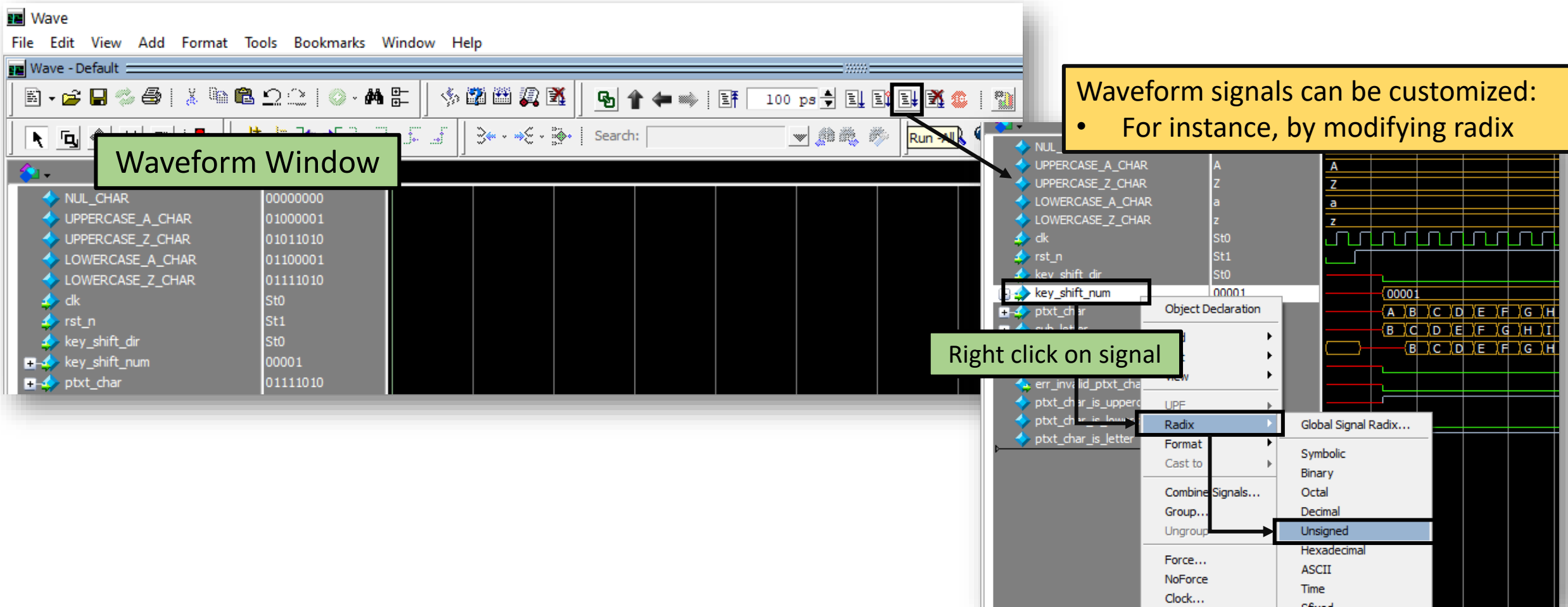
Objects can be added to a Waveform

Select required objects

Hierarchy of the selected unit for simulation

Name	Value	Kind	Mode
NUL_CHAR	Not L...	Para...	Internal
UPPERCASE_A_CH...	Not L...	Para...	Internal
UPPERCASE_Z_CH...	Not L...	Para...	Internal
LOWERCASE_A_C...	Not L...	Para...	Internal
LOWERCASE_Z_C...	Not L...	Para...	Internal
clk	Not L...	Net	In
rst_n	Not L...	Net	In
key_shift_dir	Not L...	Net	In
key_shift_num	Not L...	Net	In
ptxt_char	Not L...	Net	In
ctxt_char	Not L...	Pack...	Out
err_invalid_key_shi...	Not L...	Net	Out
err_invalid_ptxt_ch...	Not L...	Net	Out
ptxt_char_is_uppe...	Not L...	Net	Internal
ptxt_char_is_lower...	Not L...	Net	Internal
ntxt_char is letter	Not L...	Net	Internal

Quick tutorial on Modelsim



The screenshot shows the Modelsim Waveform Window. The left pane lists signals: NUL_CHAR, UPPERCASE_A_CHAR, UPPERCASE_Z_CHAR, LOWERCASE_A_CHAR, LOWERCASE_Z_CHAR, clk, rst_n, key_shift_dir, key_shift_num, and ptxt_char. The right pane shows a waveform for these signals. A right-click context menu is open over the 'key_shift_num' signal, with the 'Radix' option selected, leading to a 'Global Signal Radix...' dialog box. The 'Unsigned' option is highlighted in the dialog.

Waveform Window

Waveform signals can be customized:

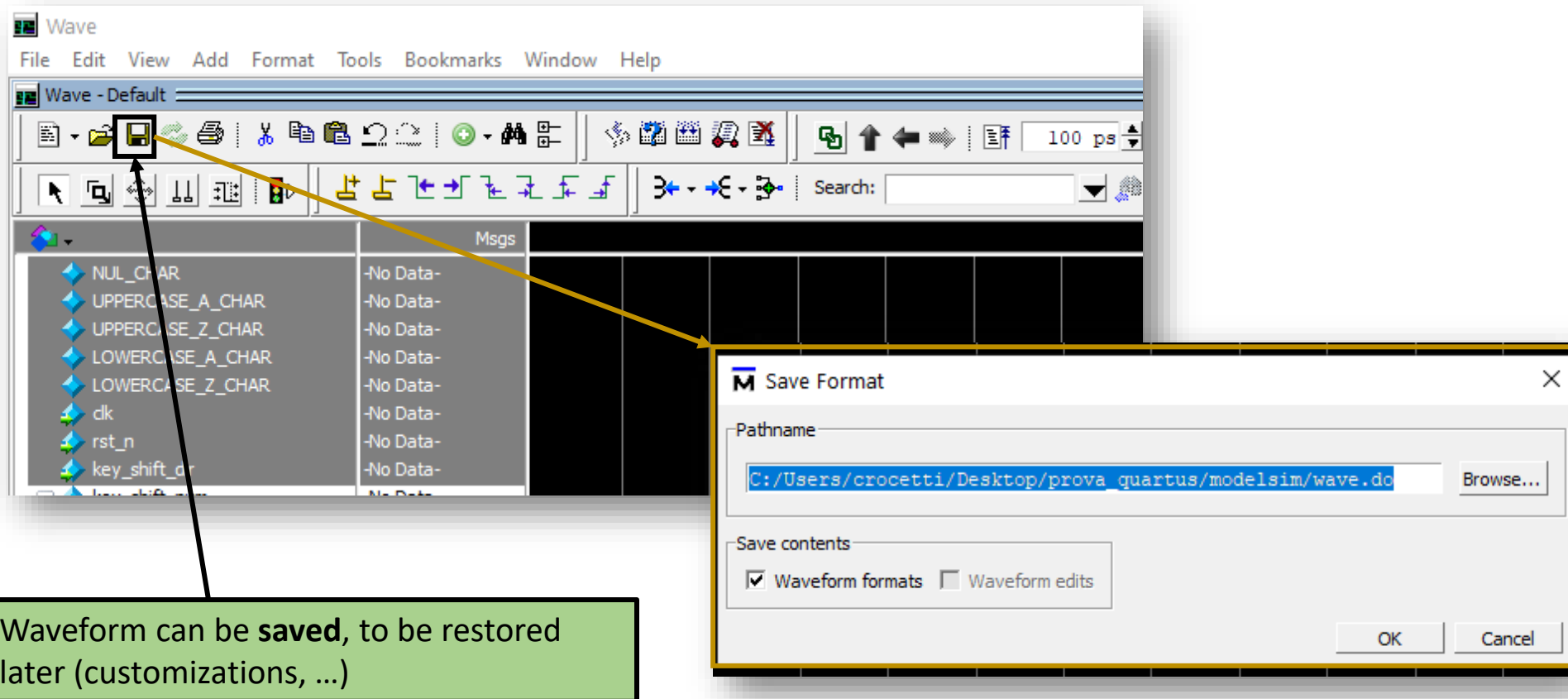
- For instance, by modifying radix

Right click on signal

Global Signal Radix...

Radix	Symbolic	Binary	Octal	Decimal	Unsigned	Hexadecimal	ASCII	Time	Custom
Symbolic									
Binary									
Octal									
Decimal									
Unsigned									
Hexadecimal									
ASCII									
Time									
Custom									

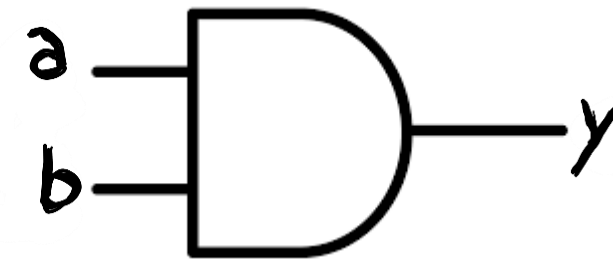
Quick tutorial on Modelsim



Quick tutorial on Modelsim

- Exercise

```
module and_gate (  
    input  a  
    ,input  b  
    ,output y  
);  
  
    assign y = a & b;  
  
endmodule
```



Quick tutorial on Modelsim

- Exercise

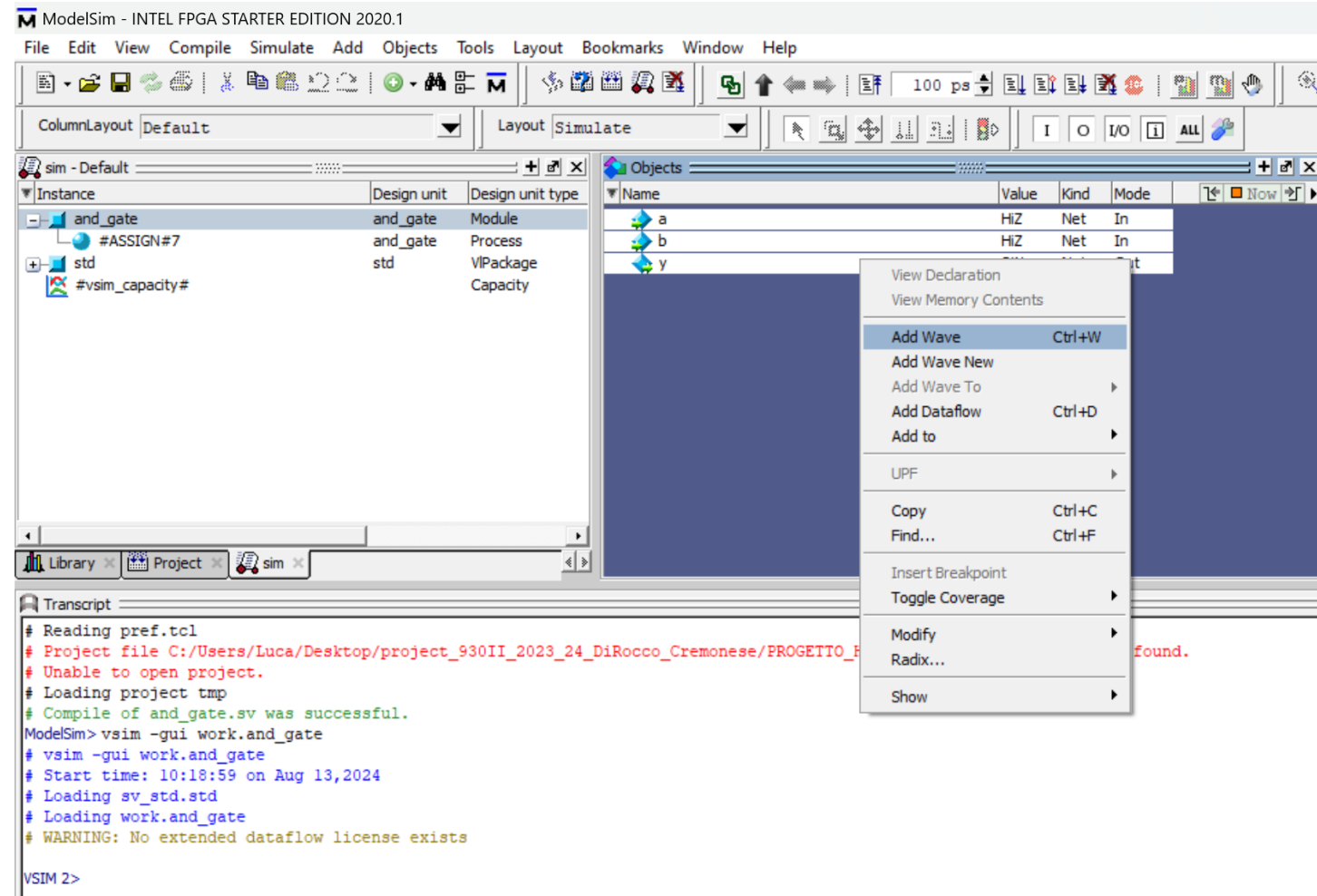
1. Create the SV file (and_gate.sv) and fill with the previous example
2. Create a Modelsim project adding the SV file at the previous step
3. Compile and launch the simulation selecting the **and_gate** unit/module
4. Now run manually the simulation using the **force** and **run** commands in the **Transcript** Tab, as indicated in the next slide

Quick tutorial on Modelsim

- Simulation (manual)
 - Using the Transcript Tab (\approx terminal/command line interface)
 - The **force** command apply logic values to ports (and internal signals) of module(s)
 - We are going to use it to apply stimuli on the input ports of the simulate unit/module
 - Syntax: **force** <input port name> <logic value>
 - The **run** command advances the simulation by the specified number of timesteps
 - When launching the simulation, the initial simulation time is 0 (seconds)
 - Syntax: **run** <timestep>
 - We are going to see the outputs using the **waveform** (at the same time)

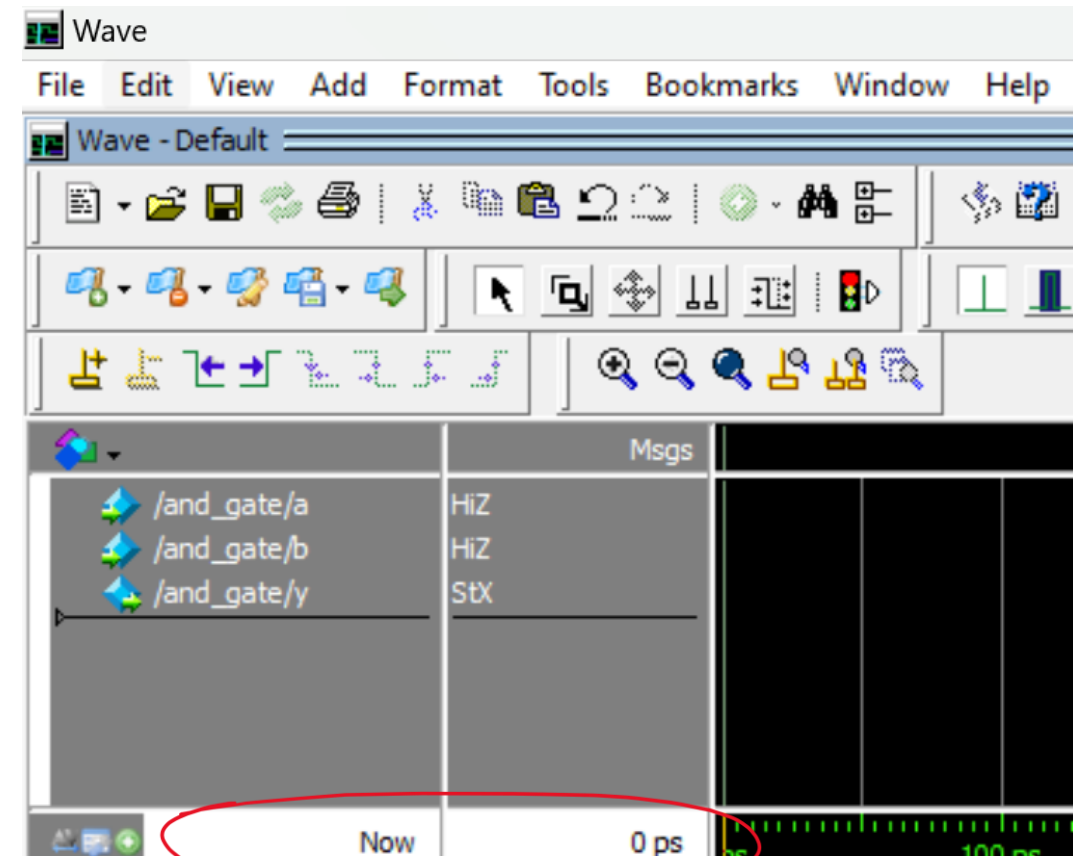
Quick tutorial on Modelsim

- Simulation (manual)
 1. Add the and_gate unit signals to the waveform



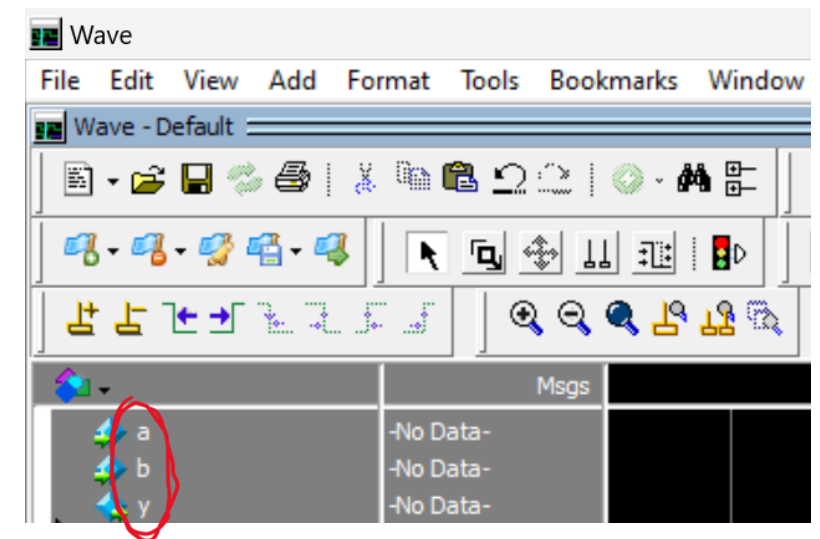
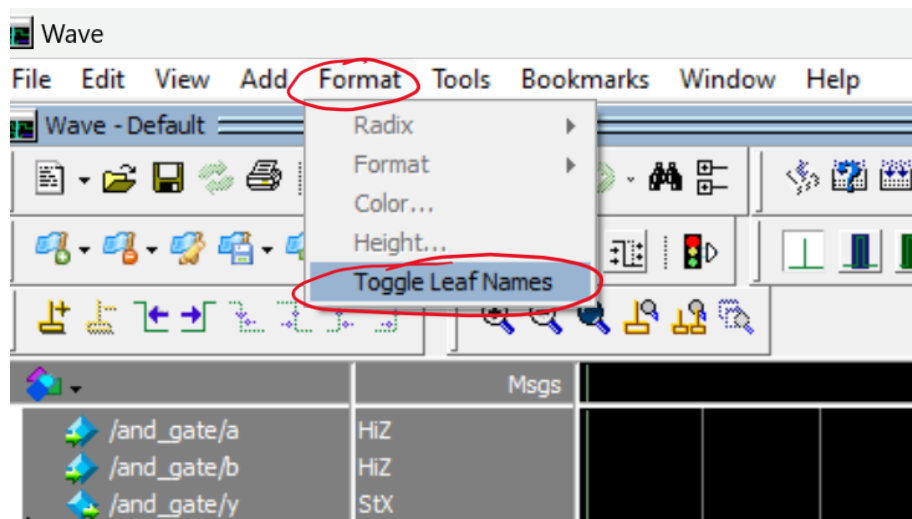
Quick tutorial on Modelsim

- Simulation (manual)
 1. Add the and_gate unit signals to the waveform
 - You will get something like this
 - Note the simulation time
 - 0 ps = zero picoseconds



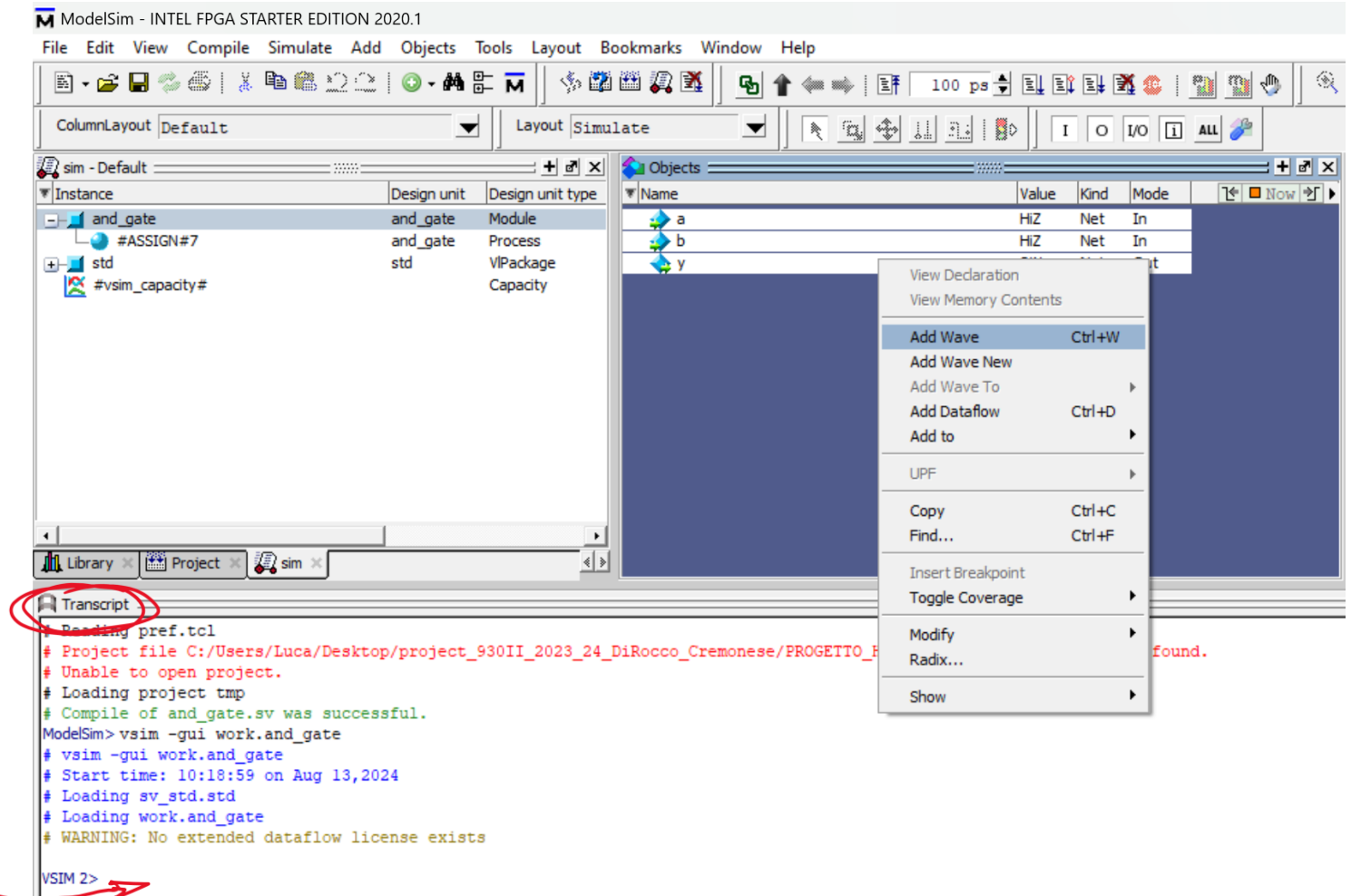
Quick tutorial on Modelsim

- Simulation (manual)
 1. Add the and_gate unit signals to the waveform
 - You can remove the hierarchical path from the displayed signals
 - More readable
 - Format > Toggle Leaf Names



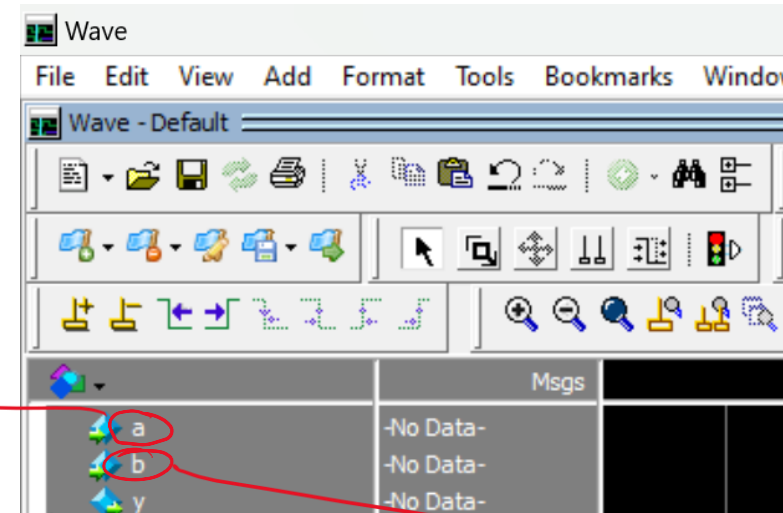
Quick tutorial on Modelsim

- Simulation (manual)
 2. Move to the **Transcript** Tab and execute the **force** and **run** commands to simulate the unit/module



Quick tutorial on Modelsim

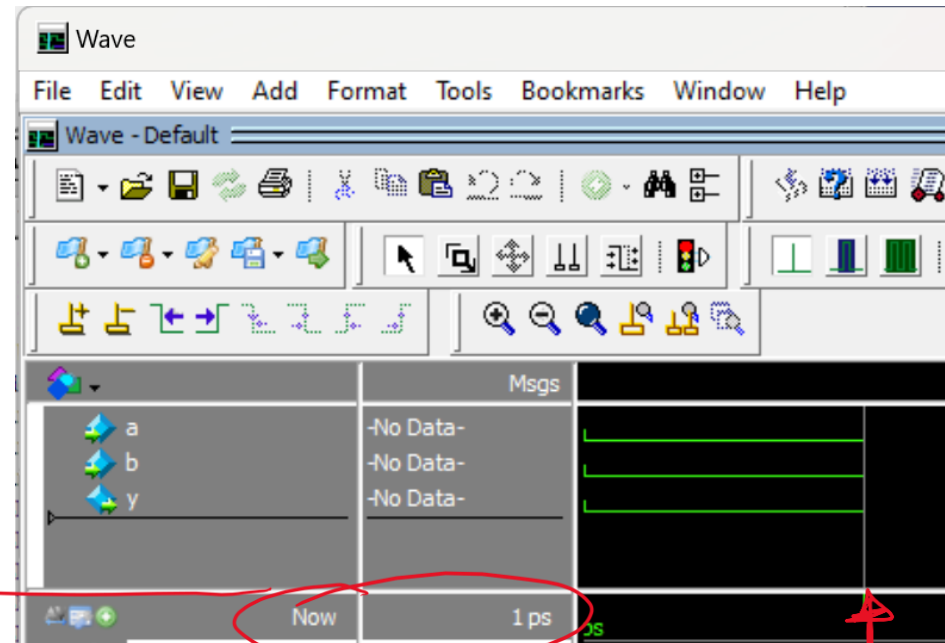
- Simulation (manual)
 2. Move to the **Transcript** Tab and execute the **force** and **run** commands to simulate the unit/module
 - Example:



- force a 0 → set input port 'a' to 0 (otherwise it is unknown, X, by default)
- force b 0 → set input port 'b' to 0 (otherwise it is unknown, X, by default)

Quick tutorial on Modelsim

- Simulation (manual)
 2. Move to the **Transcript** Tab and execute the **force** and **run** commands to simulate the unit/module
 - Example:



— run 1 → advance the simulation by 1 time unit (ps or ns, by default: depends on the tool settings)

- Note: 'y' is determined by the unit/module (thanks to the description specified in the SV code)

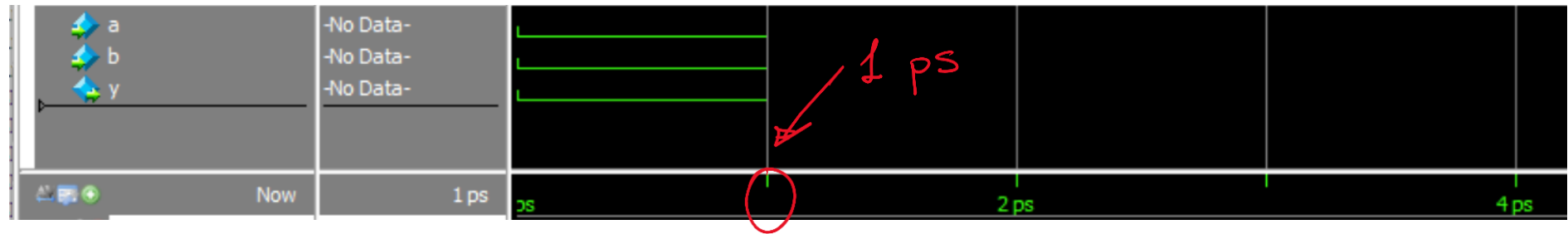
Quick tutorial on Modelsim

- Simulation (manual)
 - Try different combinations of 'a' and 'b' and for each combination run the simulation for some timestep
 - Debug/verification by checking the truth table of the AND gate

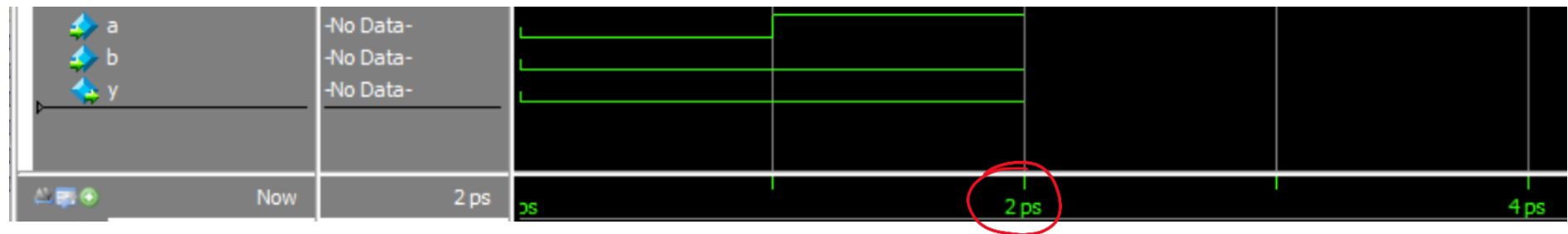


Quick tutorial on Modelsim

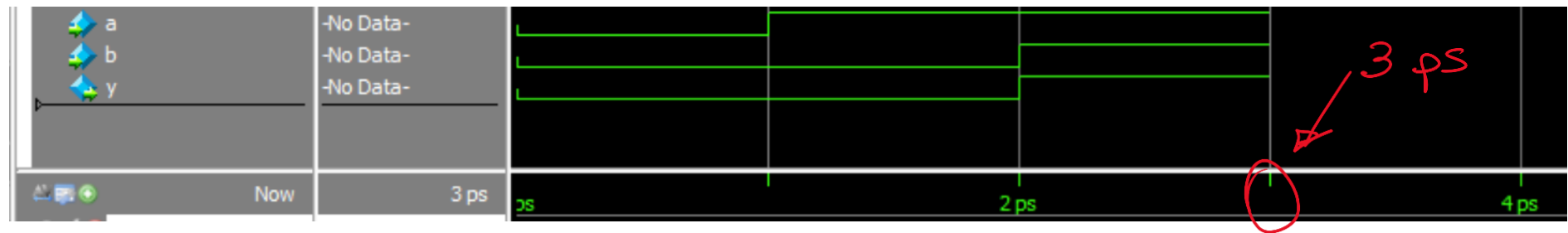
```
VSIM 3> force a 0
VSIM 4> force b 0
VSIM 5> run 1
```



```
VSIM 13> force a 1
VSIM 14> run 1
```



```
VSIM 15> force b 1
VSIM 16> run 1
```



Quick tutorial on Modelsim

- You can find all the files about this exercise in the dedicated folder on the Team of the course
 - File > Electronics Systems module > Crocetti > Exercises > 1.2
 - Please, start reading the README.txt file
- I also included two files (.do files) to automate the waveform and the simulation
 - wave.do, sim.do
 - Refer to README.txt file
 - However, you can run them using again the Transcript Tab and the **do** command
 - After having launched the simulation

```
VSIM 24> do wave.do  
VSIM 25> do sim.do
```



Thank you for your attention

Luca Crocetti
(luca.crocetti@unipi.it)