Custom Computing & Embedded Systems

FPGA simulation: ModelSim /Quest

1. **Program configuration**

   Any changes to the simulation system parameters are made in **the Tools -> Edit Preferences menu ...** The setting results should be saved in the system startup script (*modelsim.tcl*). Saving settings:

   command line : `write preferences <filename>`

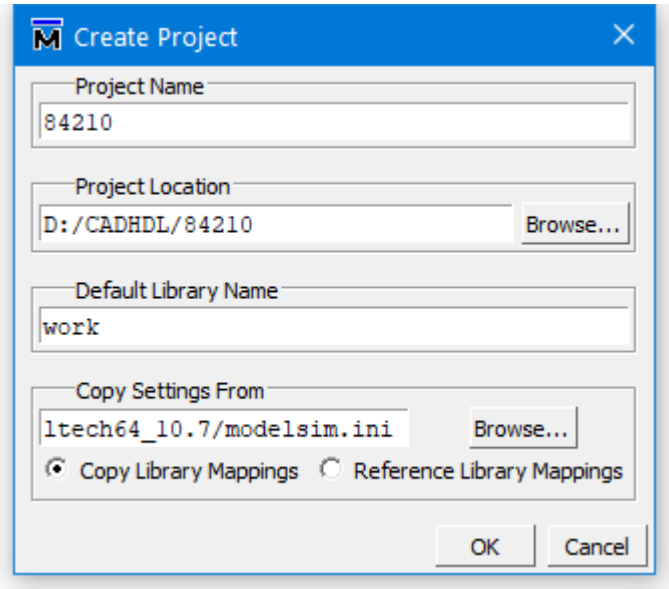
   To make the settings visible every time you start the system, set the system variable **MODELSIM_TCL=< path >\ modelsim.tcl**

   < path > - path to the modelsim.tcl file

   The MODELSIM system variable points to the default environment configuration file, *modelsim.ini*. The value of this variable is particularly important when installing different versions of the simulator in one system.

2. **Project management**

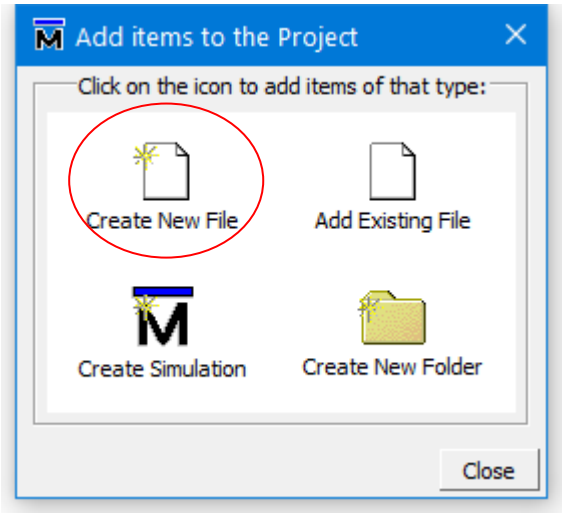
   Creating a new project: **File -> New -> Project...**

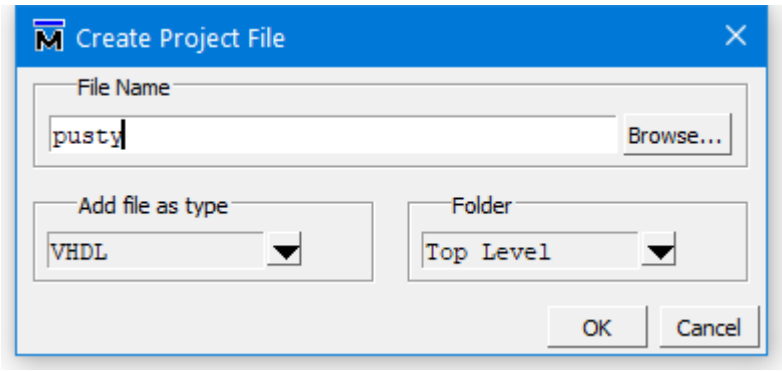


When a new project is created, a working library is created with the name given in the window above (in this case **work**). To create a new library or change the mapping (alias) to an existing library, use the commands described in point 4 of this manual or in [1].

[!] Exercise: create a new project called <index_no> in the working directory on the disk indicated by the teacher (if the directory does not exist, create it); leave the default name for the working library;

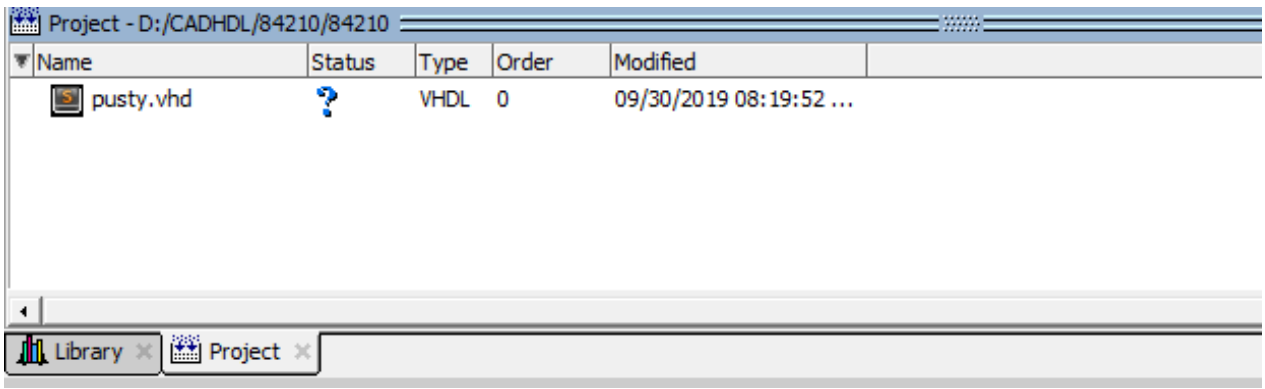
Once you have created a design space, you can add existing HDL models to it or create new design elements. For this purpose, you can use the wizard that starts after creating the project (window below) or create a new source using the command **Project -> Add to Project -> New File...**



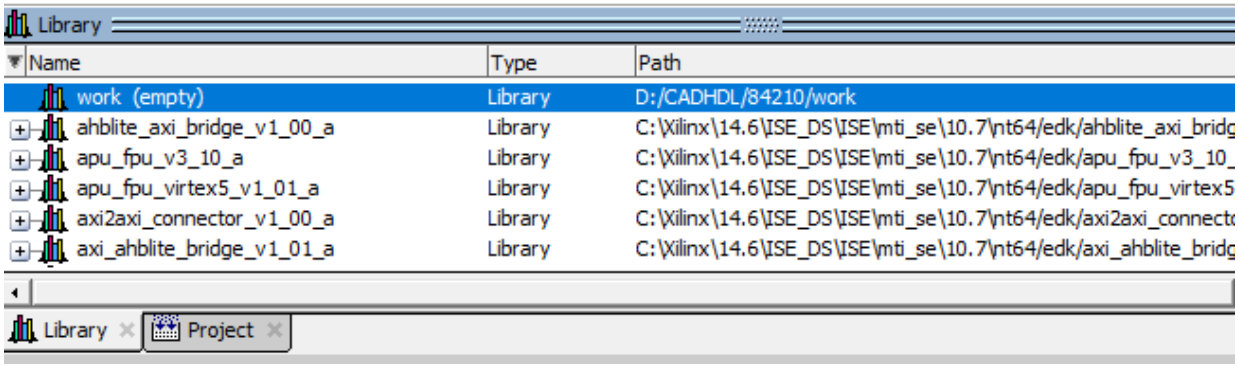


[!] Exercise: create a new design source called 'empty'; source type 'VHDL'; leave the default file location;

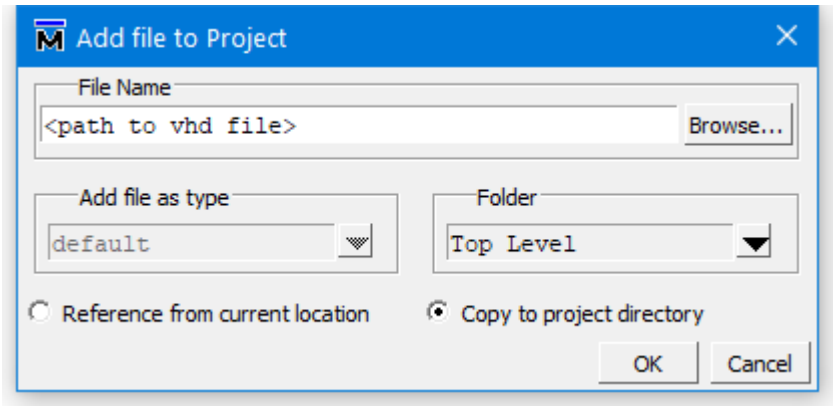
The effect of each command that modifies the structure and state of the project is visible in the workspace:



At the project creation stage, there are two tabs in the workspace: 'Project' for source files and their status, and 'Library' containing data on libraries seen from the project level. At the current stage of the project, an empty ' work ' library should be visible in the 'Library' tab .

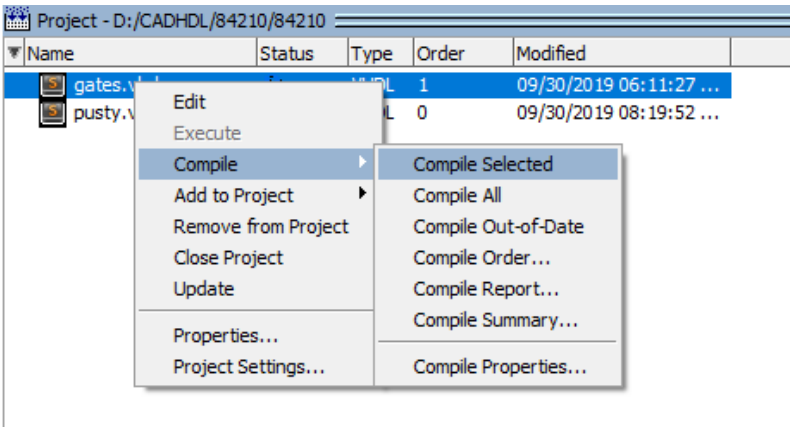


In addition to creating new models, you can add existing HDL files to your project to reuse design units once defined. Existing files can be added using **the Project -> Add to Project -> Existing File... command** or from the context menu of the 'Project' window.



[!] Exercise: download the ' gates.vhd ' file from the course server and save it in any temporary directory; add this file to the project with the ' Copy to project directory '; source type 'VHDL'; (If the file was saved in the project directory, it should be added to the project with the 'Reference from current location '.)

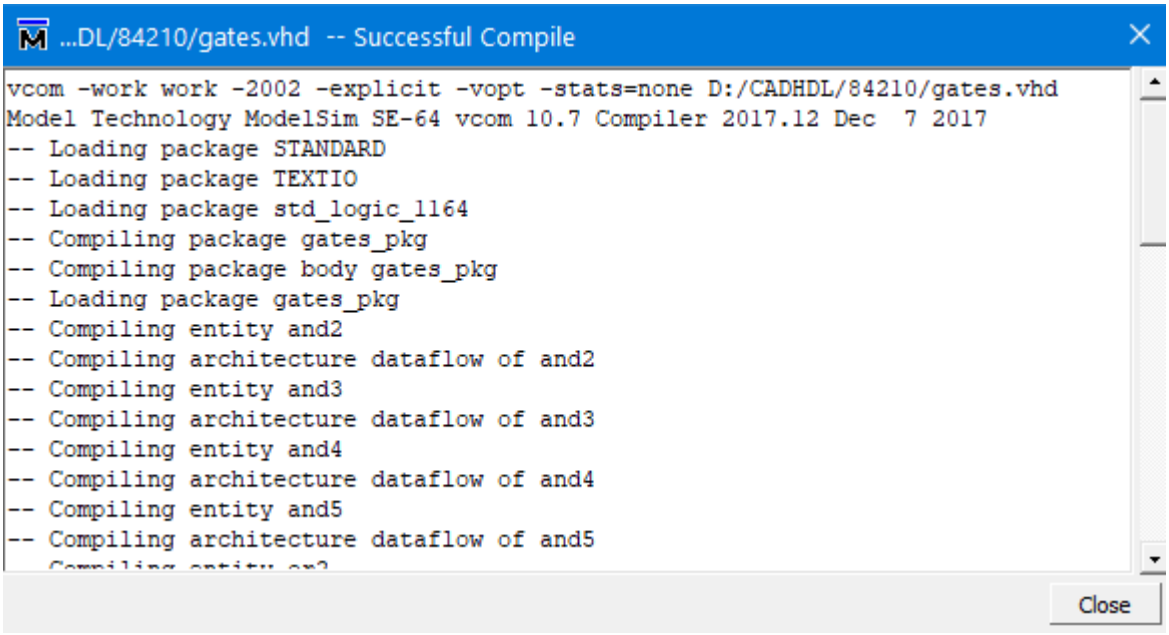
The added file may contain one or more design unit models. To make them available in the working library, the file must be compiled using the **Compile -> Compile Selected** (menu or right mouse button).



The effect of the compilation command is visible in the transcription window:

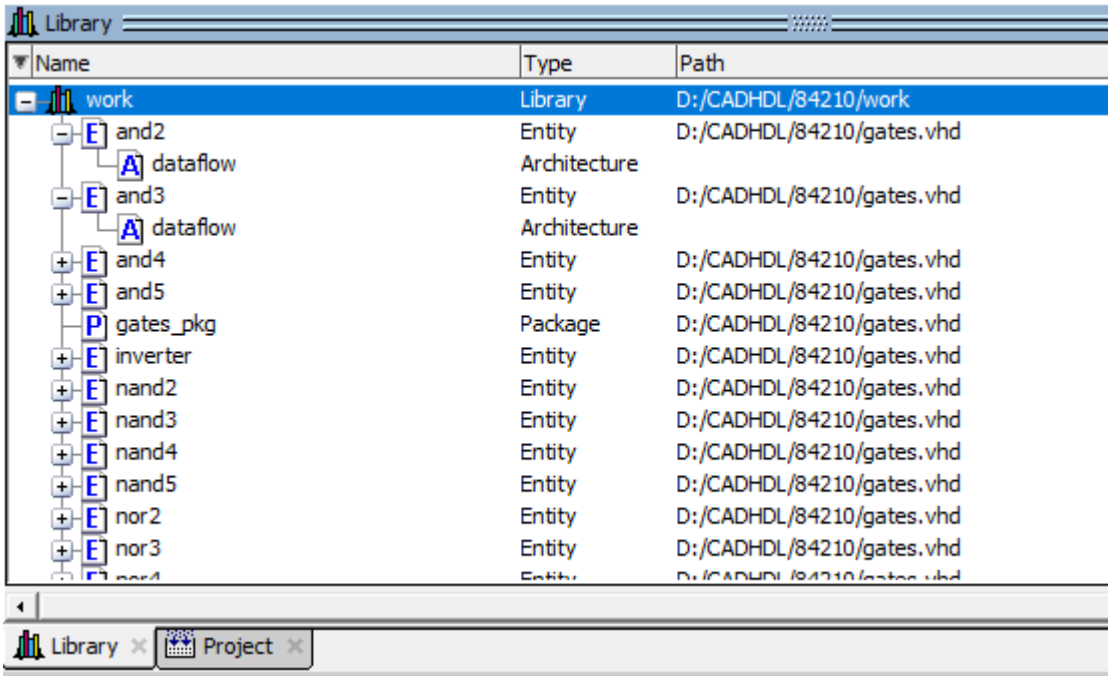
```
# Compile of gates.vhd was successful.
```

Detailed information about the operation of the command (syntax, error numbers, etc.) can be obtained by double-clicking on the transcription window message. The circled line in the window below shows the syntax of the command invoked by the **Compile -> Compile Selected** .



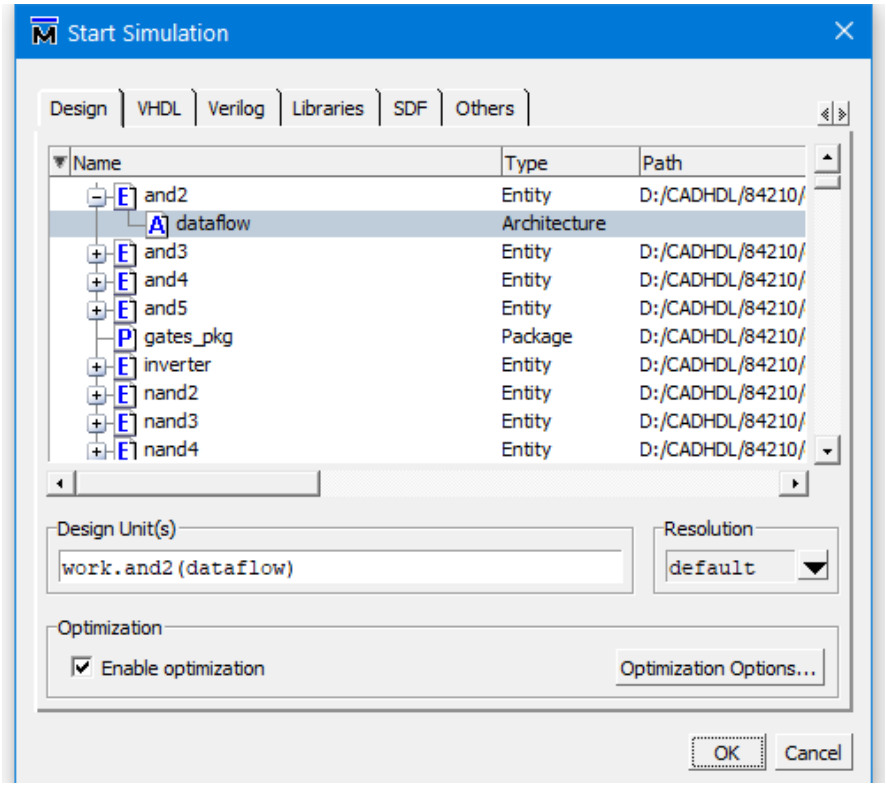
[!] Exercise: run compilation of ' gates.vhd ' file; observe the build results in the transcription window. Note the compilation command (vcom) syntax shown in the first line. Check the meaning of the **-work** parameter in the manual

After successful compilation, the working library shows the design units defined in the VHDL file being compiled.

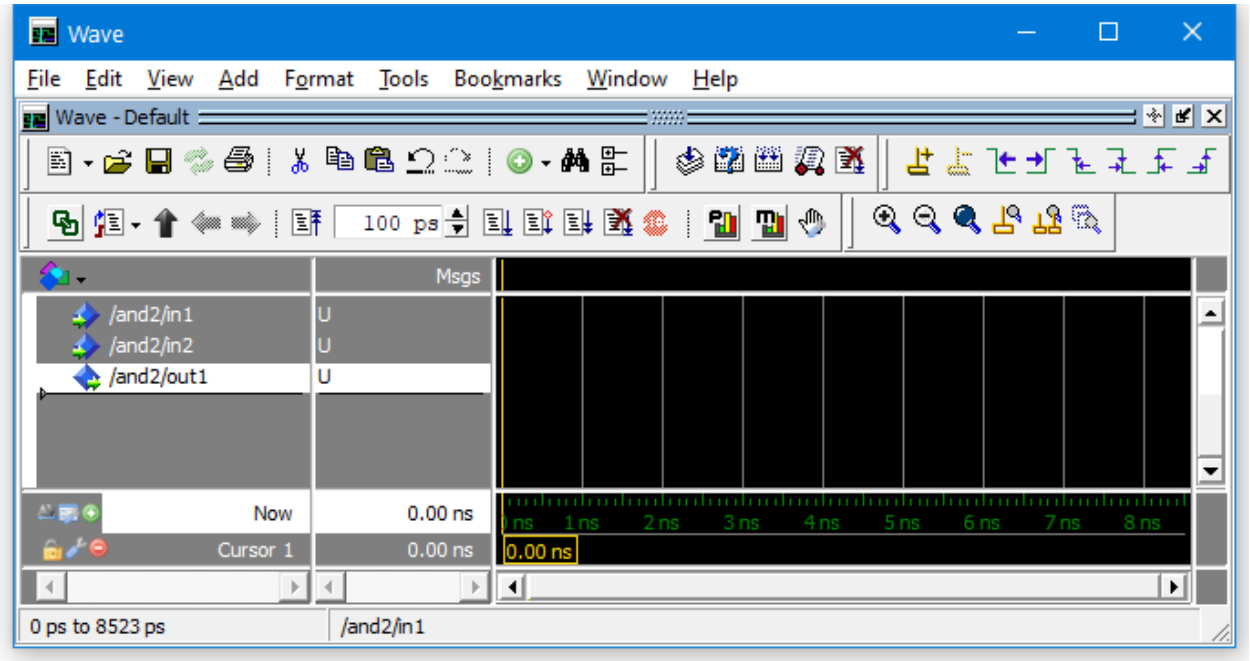


3. Simulation

The way to verify the operation of models described in hardware description languages is their simulation. The VHDL model can be simulated using forces defined in the test environment, the so-called testbench or using the ' force ' command of the simulator .
the **Simulate -> Start Simulation ...** menu.



The simulation results can be observed in text form (transcription window) or graphically (time waveform window). To observe time waveforms, create a graphical debugging window using the **View -> Wave** command and add waveforms from the 'Objects' window to it. Select the desired signals in the 'Objects' window and then run the **Add -> To Wave -> Selected Signals** from the context menu.

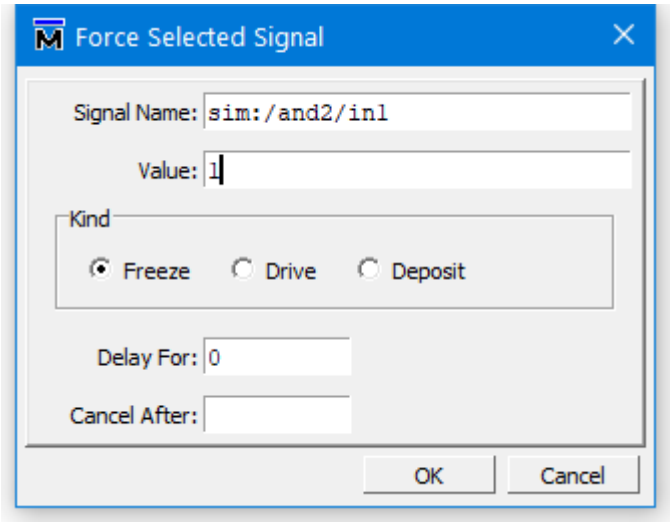


[!] Exercise: run simulation for design unit 'and2' from the work library ; display the ' Wave ' window and add all signals from the 'Objects' window to the window;

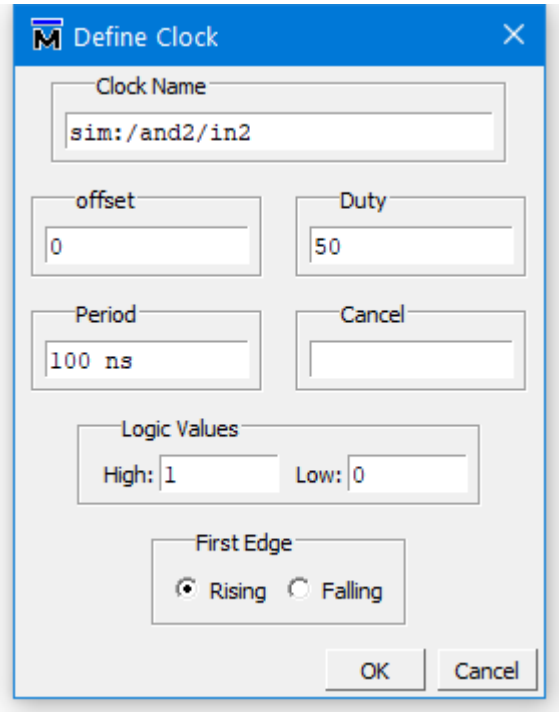
The 'force' command will be used to build forces in ModelSim . It allows you to define simple forces, but when simulating complex systems, issuing commands via a graphical interface is too time-consuming - Tcl scripts or force models defined in HDL are used for this purpose .

In the graphical interface of the simulator (context menu after selecting the appropriate time course), there are two commands for defining forces:

Force (defining single values)



Clock (defining extortion periodic)

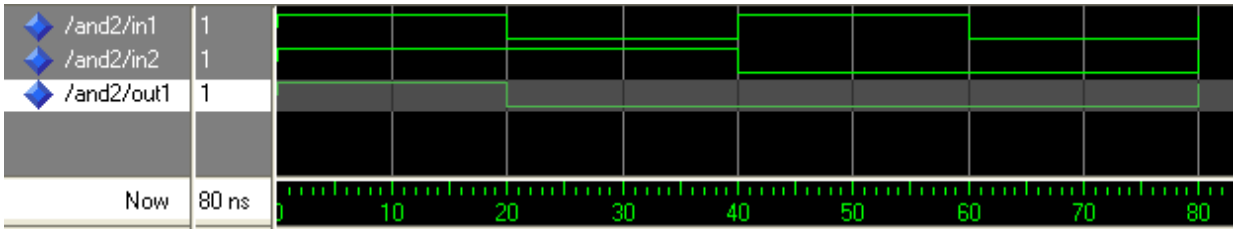


Please note that the above windows are dialog windows - you cannot see previously defined enforcements in them.

[!] Exercise: build forces for both inputs of gate 'and2'; use the 'force' command to build the sequence:

in1 -> '1' lasting 20ns, '0' lasting 20ns, '1' lasting 20ns, '0' lasting 20ns
in2 -> '1' lasting 40ns, '0' lasting 40ns

Run the simulation with the **run 80 ns** command entered in the transcription window; observe the simulation results in the ' Wave ' window;



4. Commands in the transcription window

The Tcl scripting language command interpreter is built into the ModelSim program interface . This makes it easier to automate design through Tcl scripts (macros have a standard extension of .do). Each operation described in the previous point can be performed by issuing a command in the transcription window.

Creating a new library

command line : `vlib <library name>`

by default, a new directory called libraries is created

Mapping a logical name to a library directory

command line : `vmap <logical name> <library name>`

specifying only the logical name displays the current mapping

Compilation of the selected source: **Compile -> Compile Selected**

command line : `vcom - work <library name> -2002 - explicit <file name>`

default work library name = work

Loading the simulation: **Simulate -> Simulate ...**

command line : `vsim <library name>.< entity name >(<architecture>)`

Opening windows signals : **View -> Objects**

line commands : `view objects / view signals`

Opening the Time Diagrams window: **View -> Wave**

line commands : `view wave / view -new wave`

Adding the selected signal to the results window: **Add -> Wave -> Selected Signals**

command line : `add wave - label <label> <signal name>`

Adding all signals to the results window: **Add -> Wave -> Signals in Design**

command line : `add wave -r /*`

Activation simulation : **Simulate -> Run -> Run-All**

line commands : `run -all`

(*all issued commands and messages regarding their results appear in the transcription window; detailed description of commands in item [1]*)

[!] Exercise: using only the command line, perform the following operations:

- create library ' gates '
- compile the gates.vhd file into the ' gates ' library
- map the ' gates ' library to the logical name 'gates'

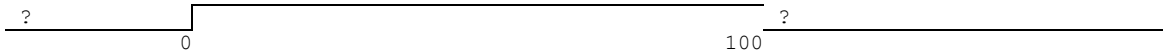
In the Library tab, display the contents of the 'gates' library, is it identical to the contents of the 'gates' library?

Using the **force** command

Simple device models can be tested using **force** commands issued from the transcription window or saved in a macro file (.do). When simulating complex devices, this design approach is ineffective - a more advanced test environment (testbench) should be used.

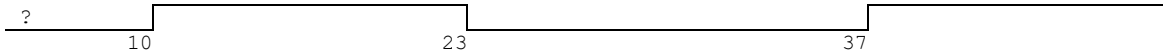
```
force < signal_name > <v1> - cancel < tc >
```
example : `force clk 1 - cancel 100ns`

clk signal at the current hierarchy level takes the value '1' when the command is run; the -cancel switch allows you to cancel the force after a set time



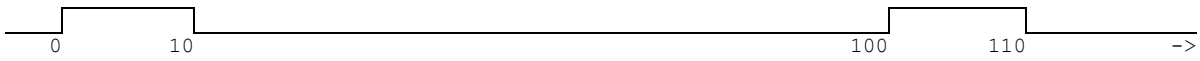

```
force < signal_name > <v1> <t1>, <v2> <t2>
```
example : `force / uut /reset 1 10ns, 0 23ns, 1 37ns`

the reset signal at the / uut level of the hierarchy takes the value '1' at 10ns from the time the command is run, then '0' at 23ns and '1' at 37ns (relative to the time the command is run)




```
force < signal_name > <v1> <t1>, <v2> <t2> - repeat < tp >
```
example : `force clk2 1 0ns, 0 10ns - r 100ns`

-repeat (-r) switch allows you to create periodic forces with infinite duration and repetition period tp ; the example shows the clk2 signal with a frequency of 10MHz
and a duty cycle of 10%




```
force
```
force command without parameters allows you to control currently defined forces, e.g.:

```
force - drive / cntrx / ce 0 {@0 ps } , 1 {@700 ns}
force - drive / cntrx / rst 0 {@1002 us} , 1 {@1002125 ns} , 0 {@1004400 ns}
force - drive / cntrx / clk 0 {@0 ps } , 1 {@50 ns} - repeat {@100 ns}
```

5. Simulation automation

.do script files (macros) are used to automate the operation of the program
They consist of Questa [1] commands and Tcl commands .

macro example for the testowy.vhd file :

```
# compile test model
    vcom - work work -2008 ./testowy.vhd
# load the model into the simulator
    vsim test.work
# display the signals and simulation results window
    view waves –title test_wave
    view signals
# adding all signals to the results window
    add waves *
# extortion construction
    force c 1 0, 0 {20 ns} -r 40ns
    force b 1 0, 0 40ns -r 80ns
    force a 1 0, 0 {80 ns} -r 160ns
# run the simulation
    run 350 ns
```

Download the test.vhd file from the course server and add it to the project, do not compile the file.

[!] Exercise: write a macro file that performs the following operations:

- creating a library with a logical name <student index number>
- compiling the test.vhd source into a library called <student index number>
- starting the simulator for the test model
- display of waveforms
- construction of forces and model simulation
- scaling of the results window for the entire simulation time
- use command line parameters to pass the filename and simulation time as macro command line arguments
- ! upload all sources needed to run the macro (ZIP) as well as a screenshot of the simulation results and macro listing (PDF) to the course server

[1] Siemens EDA, Questa® SIM Command Reference Manual Including Support for Questa Base, Software Version 2023.3, Document Revision 8.3, © 2023 Siemens.