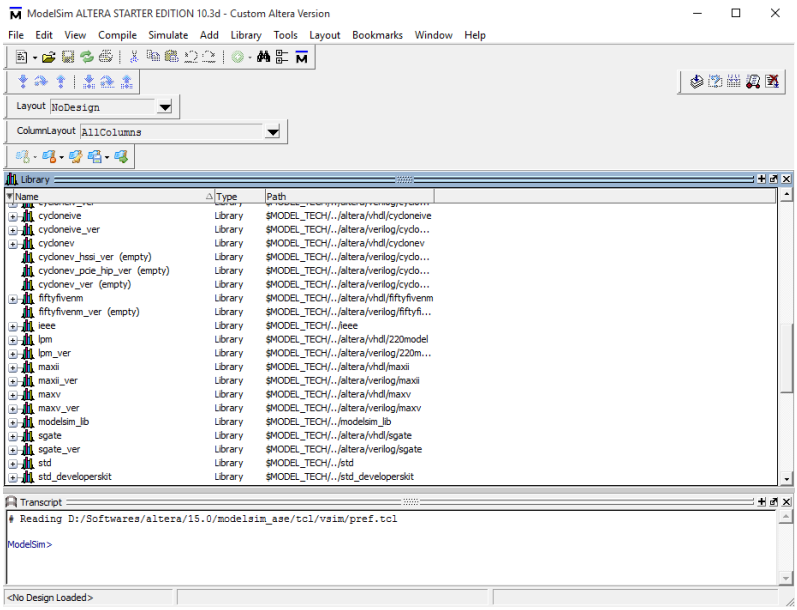
Introduction to Digital Electronics Design Part B

Ready? Okay! Let's get a bit practical now:

1. We have provided the following files for you to play with: `full_adder_1_bit.sv`, `full_adder_4_bit.sv`, `test_fulladder_4bit.sv`. Copy these into a fresh new directory. In general for the Altera tools, **always avoid spaces and special characters in the path names**.

2. Start ModelSim Altera Starter Edition. If you work on the lab computers, you can find it in the start menu under "Altera -> ModelSim Intel FPGA Starter Edition". Skip the possible "first open" dialog box. Here is what it then should look like[1] :

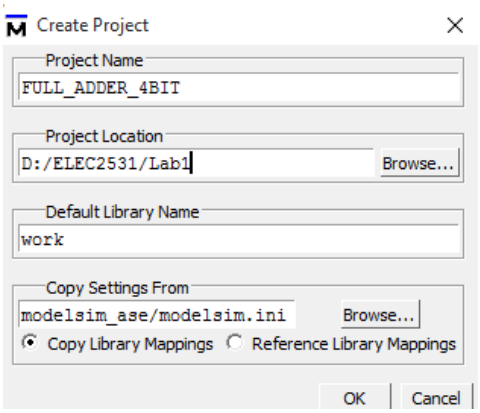


3. Create a new project

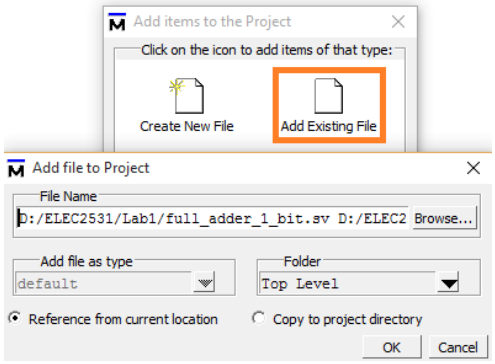
    a. File→New→Project

    b. Enter in the Create Project dialog box, FULL_ADDER_4_BIT as project name and select the directory where the provided files are stored (which, as a recall, must have a path free of spaces and special characters). Do not modify the remaining fields and click OK.

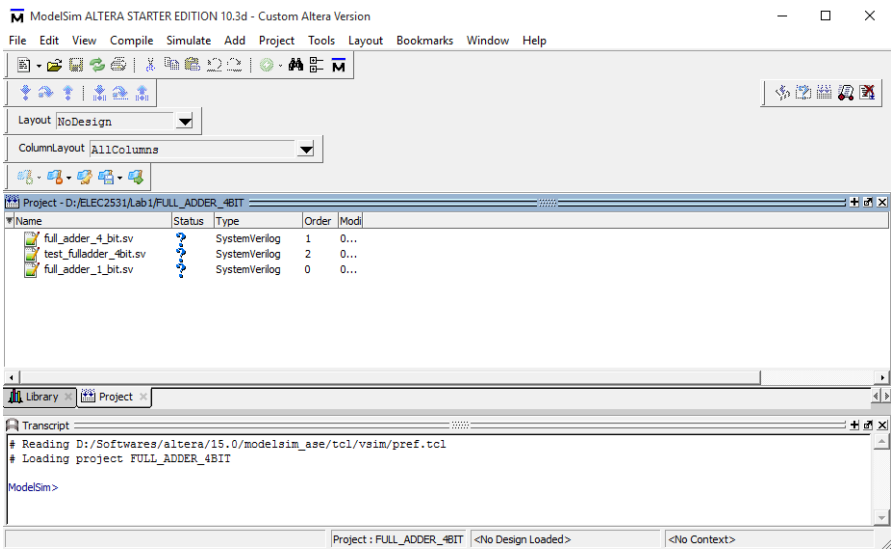
[1] Please note that the snapshots of ModelSim in this document are for the 10.3d version, not the current 10.5b, but there are minor differences between the versions.



4. Now, add the files to the project:



5. Click OK, you should now see the following window:



6. Double click on each file to open them and try to understand the design based on the block schemes provided above. If you have difficulties, feel free to call us.

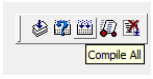
   Let's already emphasize something: **the 'initial' construct**. This describes behavior occurring only once in simulation. All statements inside an `initial` statement constitute an `initial` block. An `initial` block starts at time 0, executes exactly once, and then does not execute again. If there are multiple `initial` blocks, each block starts to execute concurrently at time 0. Each block finishes execution independently of other blocks. Be careful when using `initial` outside of a testbench, since such constructs are generally not synthesizable; therefore, it is often better to use a 'reset' signal instead.

   A practical use of the initial block is the generation of stimuli during simulation. Here is an example:

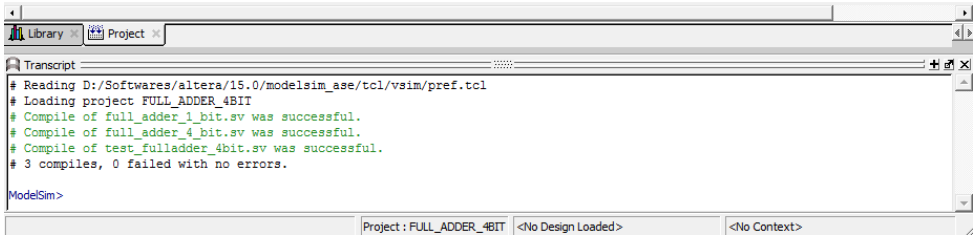
```
initial begin
        a = 0; b = 0; c = 0; #10;
        c = 1; #20;
        b = 1; c = 0; #10;
        end
```

   In this example, at the start of simulation the `initial` statement assigns value 0, 0, and 0 to a, b and c respectively, then waits 10 time units. It then assigns the value 1 to c, waits 20 time units, and so forth. The time unit used is defined via the `` `timescale 1ns/1ps `` directive (in this case setting a base time unit of 1ns, with a simulation internal resolution of 1ps).

   You can now fully understand how the testbench provided ("test_fulladder_4bit.sv") works. Check your book[2] to learn more about this; it is primary for you to get how delays work as well as how to set a timescale and also how to quickly write a small testbench to test a design.

7. Compile all files with the button "Compile all": 

   There should be no errors:

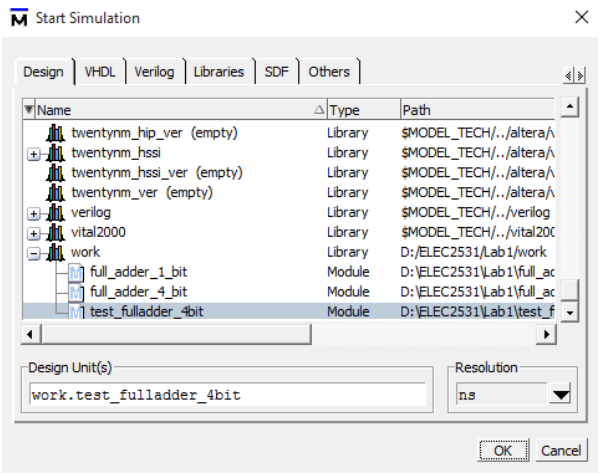


8. Start the simulation:

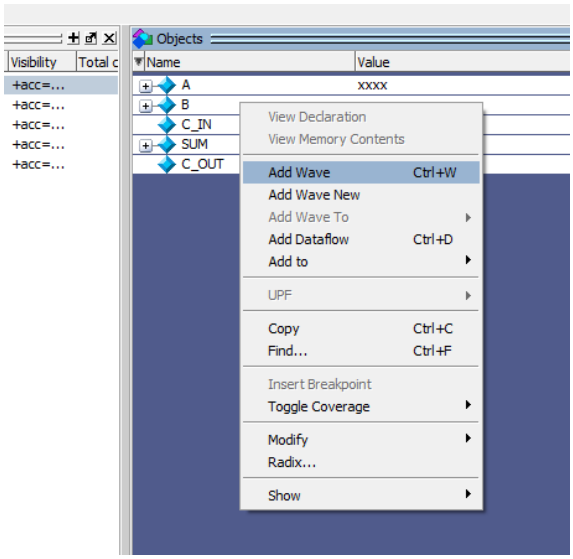
   a. Simulate→Start Simulation

[2] Harris & Harris (ARM ed.), HDL example 4.13, page 189.

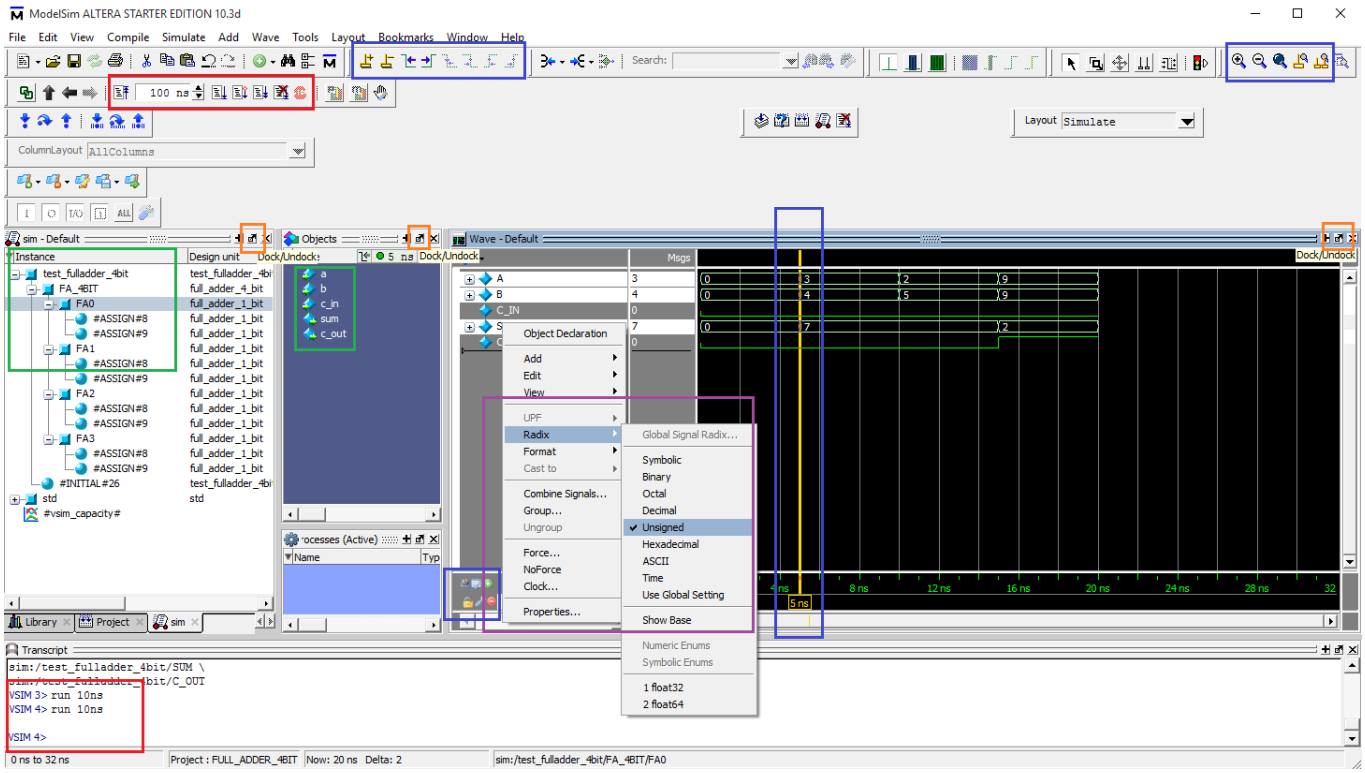
b. Expand “work”, select the “test” module which simulates external stimuli, set the “Resolution” to “ns” and click OK:



c. Once the tool is ready, select the available signals as shown below and click on “Add Wave”:



d. If not already present, the “Wave” sub-window should open, normally docked to the main ModelSim GUI. Now, in the command bar (“Transcript” sub-window) at the bottom, type in “run 10ns” and then once again. You should get something resembling the figure at the next page.



9. Now, take as long as it takes here to fully discover everything we highlighted in this environment. **It is absolutely primary that you quickly get familiar with all these features** in order to be able to set an efficient debug environment :

    a. **Sub-windows organization**: the ModelSim interface is composed of several sub-windows that you can individually open (using the "View" menu) or close ("X" button), dock/undock (button within the **orange** squares) to/from the main GUI, maximize ("+") or minimize ("-"), and move within the interface (drag & drop using the dotted space in the middle of the top bar).

    b. **Add internal signals/nodes to the wave**: what we did here was to display the signals of the top-level module (which is here our testbench "test_fulladder_4bit"). This is always what ModelSim propose you by default when launching a simulation. However, it is very often interesting to have a glance at what is going on deeper in the design. Using the "sim" sub-window, you can navigate on the modules hierarchy, seeking for the signals you want to display. For example, within the left **green** square, we selected the first 1-bit full adder ("FA0") that composes our 4-bit full adder; in the "Objects" sub-window, the signals are now the 1-bit ones of FA0 and we can add them to the wave (drag & drop or right-click + add wave). Trust us, you will need this a lot this year.

c. **Get rid of the wave name paths**: you maybe noticed that, unlike you, we only have the signals name on the Wave and not their complete path. This can be achieved by clicking on the "Wave" sub-window and then using menu: "Wave > Format > Toggle Leaf Names". This gets really useful when having dealing with several levels of hierarchy.

d. **Wave display options**: in the middle of the wave, we made appear the wave display options menu (**purple** square, right click on a wave) which affects all the wave selected at once (here "A", "B" and "SUM"). Two options you should directly know about are the signals radix (which should be "unsigned" for these signals, and binary for the two others) and the possibility to group signals (using "Group"). Setting the former is inevitable in order to read what you display while the latter becomes very useful when you display the signals from several modules at once. Besides, you have the possibility to modify the wave color in its "Properties"; tremendously useful as well.

e. **Simulation options**: to run simulations, you can either use the "Transcript" sub-window (**red** bottom left square) with Tcl commands[3] - you will find documentation under the "Help" menu > PDF documentation > Reference manual - or you can use the dedicated buttons (**red** top left square, just slide your mouse on them to discover their usage). For example, if you add a new signal to the Wave after having already launched simulation for some amount of time, data won't be available for this new wave; if you want to display its behavior for the time already "passed", you will need to restart the simulation (by using the "restart" Tcl command or the dedicated button (left one)).

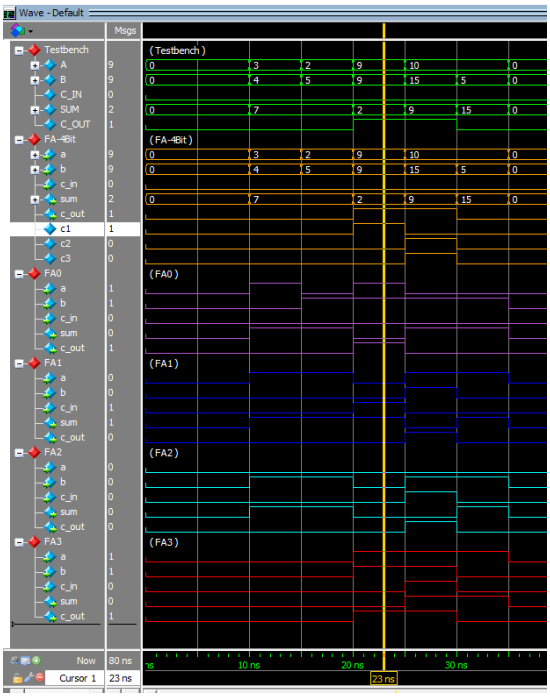
f. **Wave time scale and cursors**: you might need at some point to modify the time scale of the Wave grid; to do so, you have to open the "Grid, Timeline and Cursor controls" by clicking on the top center button on the bottom left **blue** square. On the same box, the top left button can be used to "Toggle Leaf Names" as well while the right button can be used to add cursors. The bottom buttons are the controls for the cursor that is displayed at "5 ns" (center **blue** square). One usage of cursors is to display the value of all the signals at a given time (in the "Msgs" central box). The top left **blue** square contains buttons related to cursors. And finally, the top right **blue** square allows you to zoom/unzoom on the Wave.

g. **Save a "Wave" configuration**: set an efficient debug environment using the "Wave" is particularly important but may seem time-consuming, especially if you have to rebuild it at each simulation. Indeed, debug implies generally code modifications, which in turns calls for recompiling and then starting a new simulation that will close your current Wave. However, you can save your settings in a Macro (".do" file) by first clicking on the "Wave" sub-window and then using the menu: "File > Save Format". Later, after having

[3] Obviously you can achieve everything using commands, but running & restarting simulations or adding wave are the most common usages.

start a simulation, you can bring back your environment using “File > Load > Macro File”, or simply using the command “do file_name.do” in the transcript. One more thing to note with this: be sure to have the right zoom in your “Wave” before saving it into the macro, because it is saved as well and is way more convenient when loading it back.

10. Here are two examples of (exaggeratedly) exhaustive debug environment for the 4-bit full adder design. They both contain all available signal but differ in the color usage: the one on the left has one color per module, while the one on the right use the same color to track the same wire across the different modules.



a. Get used to the ModelSim features by building the environment from the left and save it to a Macro (“my_wave.do”). Try to figure the time it takes you to do so.

b. Let’s now illustrate how scripting the steps of your simulation can make you save a great amount of time and smooth your ModelSim usage all along this year:

   i. Open your saved Macro (the .do file) in a text editor. You can use the native one from ModelSim using the command “notepad my_wave.do” in the transcript. Then, copy paste the following over-commented Tcl lines before and after the already present commands :

```
# The $ sign can be used to pass argument through the "do" command ($1 for the first, $2 for the second, etc.)
if {$1 == 0} {

        # restart the simulation (can be useful if you add a new signal to the Wave)
        restart –f

} else {
        # delete all signals in the Wave (actually not necessary if you start a new simulation (with vsim))
        delete wave *

        # recompile all ".sv" files (warning: unlike using the "Compile ..." buttons of the ModelSim GUI,
        # this will not update the file "Status"). Obviously, this becomes not really efficient with large designs
        vlog *.sv

        # restart a simulation (necessary if you recompile one of your design files)
        vsim -gui -t ns work.test_fulladder_4bit

        #
        # replace this comment by the commands that were already present in the file
        #
}

run 50 ns
```

ii. You have now an operational Tcl script for your simulation. First, read the comments and try to understand what the purpose of each line is. Its first possible use is by typing "do my_wave.do 0" in the transcript: that will simply restart and rerun the simulation

iii. On the "test_fulladder_4bit.sv" file, modify for example the first delay instruction in the initial block from "#5" to "#10". As explained above, the effect will be that the first input values will be set after 10ns of simulation instead of 5ns.

iv. Having modified a module of your design, you now have to recompile the file that contains it before being able to simulate it. To do so, you can use your Tcl script the second possible way: type "do my_wave.do 1" and let the magic operate. In one command, you are now able to achieve what you did manually in point "10.a". Pretty neat huh ?

c. Another noticeable feature is the following: to be able to observe the behavior of a signal under simulation, it has to be displayed on the wave before running. However,

once it is simulated you can remove it or put it back on the Wave as you want, without losing the data. Therefore, you have the possibility to switch between several Wave configurations after running and still observe the signal behavior.

Here is an illustrating exercise for you:

- Modify the environment you built to match the one on the right proposed at page 7 and save it into a new Macro file.

- Now, modify your Tcl script (e.g. using new arguments ($2, $3, …)) to parametrize the following things:

    a. Which environment you want to display.
    b. Do a recompilation + start a new simulation, or not.
    c. Restart and run X ns, or run an extra X ns, or do nothing (this last is useful if you want to simply switch the environments)

Conclusion

Okay, in this tutorial the following items were briefly covered:

- What Verilog and SystemVerilog are and how to use them.
- The guidelines ("Three-Y's rule") to design digital electronics circuits efficiently using an HDL.
- How to simulate Verilog in ModelSim and set an efficient debug environment, either with the GUI or using a dedicated Tcl script.
- The difference between "synthesizable" Verilog (the actual circuit) and "non-synthesizable" Verilog (the testbench).

We would like to stress how important Verilog is for the rest of this course. Handling Verilog "natively" is essential, and an integral part of the rest of the course. To do so, please read the relevant chapters in the course reference textbook. Although later lessons in this course will focus on SystemVerilog, the Verilog subset is an ideal learning tool.

To help you revise, there is a small homework due for next week, please see the Moodle website.

That's all folks!

Enjoy!