

Exercise 2: SystemC and Virtual Prototyping

SystemC Modules

Lukas Steiner

WS 2023/2024

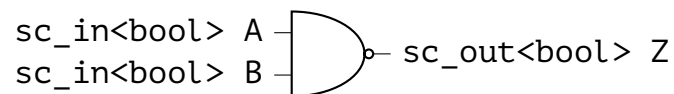
The source code to start this exercise is available here:

<https://github.com/TUK-SCVP/SCVP.Exercise2>

Task 1

NAND Gate

In this task you will write your first SystemC module. The module should have the name `nand` and should implement the functionality of a NAND gate, shown below.



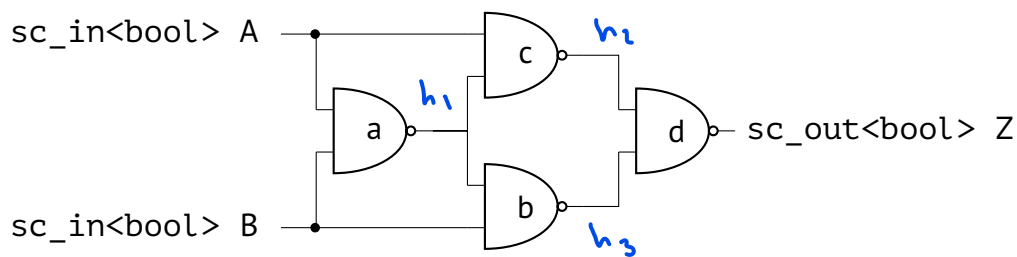
As input and output signals `sc_in` and `sc_out` should be used with the template type `bool`. The input and output signals should be initialized with a proper name in the `SC_CTOR`. The module should have one `SC_METHOD` called `do_nand()`, which is sensitive to the input signals `A` and `B`. The module should be implemented in the file `nand.h`.

In order to test your module make sure `nand_main.cpp` is included in the CMake file `CMakeLists.txt`. After successfully testing your NAND gate change the project file to include `exor_main.cpp`. This is necessary to test your next SystemC module in which you will implement an XOR gate using four instances of your NAND.

Task 2

SystemC Module Hierarchy – XOR

In this task you will write a SystemC module that is composed of other SystemC modules. The module should have the name `exor` and should implement the functionality of an XOR using only NAND gates, as shown below.



In order to connect the nand modules, you need additional helping signals which you will implement by using the `sc_signal<bool>` datatype. The signals should have the names `h1`, `h2` and `h3`. All input, output and helping signals as well as the nand modules should be initialized properly with a name from the `SC_CTOR(exor)`.

If you are done with the implementation, have a look at the `SC_MODULES` `stim` and `mon` and the `sc_main()` function and try to understand what these components are doing.

Why is the `stim` class using an `SC_THREAD` for its process and not an `SC_METHOD`?

Now lets compile and run your program. If you did everything correctly, you should see the following output:

| time | A | B | F |
|-------|---|---|---|
| 0 s | 0 | 0 | 1 |
| 0 s | 0 | 0 | 0 |
| 10 ns | 0 | 1 | 0 |
| 10 ns | 0 | 1 | 1 |
| 25 ns | 1 | 0 | 1 |
| 35 ns | 1 | 1 | 1 |
| 35 ns | 1 | 1 | 0 |
| 45 ns | 0 | 0 | 0 |

Why are you seeing several outputs for each time, sometimes even with wrong results?

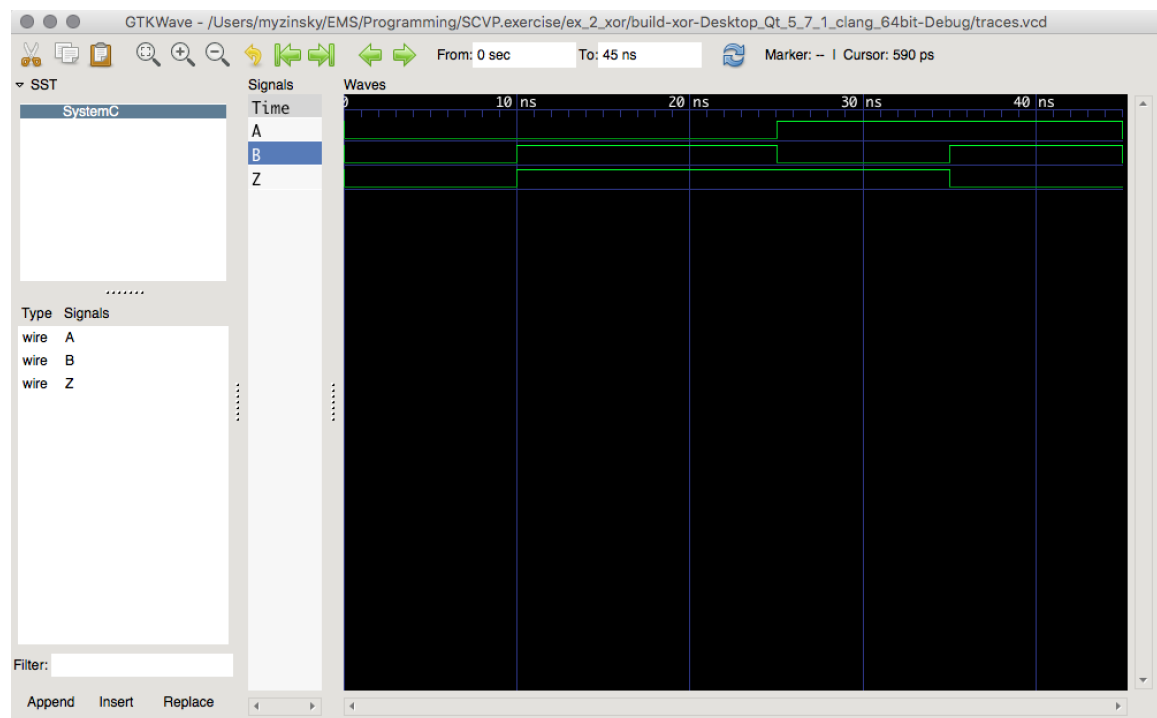
Task 3

Debugging Tracing

Additionally to the mon component, use the waveform feature of SystemC in the `sc_main` method before `sc_start`. Then use the tool GTKWave in order to have a look on the waveform. The file is located inside the build folder.

Information on this feature can be found here: <https://www.doulos.com/knowhow/systemc/tutorial/debugging/>

In GTKWave you have to drop the signals to the waveform and you have to zoom out a little in order to see the final result:



Task 4

Clocked Processes

Add an `sc_clock` to the `sc_main` function. Remove the `wait(XX, SC_NS)` statements in the `stim` module and replace them with empty `wait()` statements. Add an `sc_in<bool> Clk` to the `stim` and the `mon` components and make the processes of both modules only sensitive to the positive edge of the clock. Then connect the `sc_clock` in the `sc_main` to the modules. What you will observe at the terminal output? Now add the clock signal to the waveform and analyze it with GTKWave.