Quick-Guide to first Verilog simulation
(for Windows users)

© Jakob Ratschenberger
Institute for Integrated Circuits, Johannes Kepler University

v0.1 WS2022/23

1 Installations

The following programs have to be downloaded and installed:

- Docker (https://docs.docker.com/desktop/install/windows-install)
- Xming X-server (https://sourceforge.net/projects/xming)

2 Download IIC-OSIC-TOOLS

Download the `IIC-OSIC-TOOLS` from GitHub (https://github.com/iic-jku/iic-osic-tools) as an archive and extract the files.

3 Setting up the environment

3.1 Start the X-server

Start the X-server by launching the XLaunch application. A new window opens, and the configurations should be set as in figure 1.

3.2 Start Docker

Start Docker and leave it as is (it takes a while to start up).

3.3 Start a terminal

- Start a terminal and navigate to the folder of the extracted `IIC-OSIC-TOOLS` archive.
- Type in

  ```
  > .\ start_x.bat
  ```

  and press enter.

If everything works out correctly, a new terminal window should pop up as in figure 3.

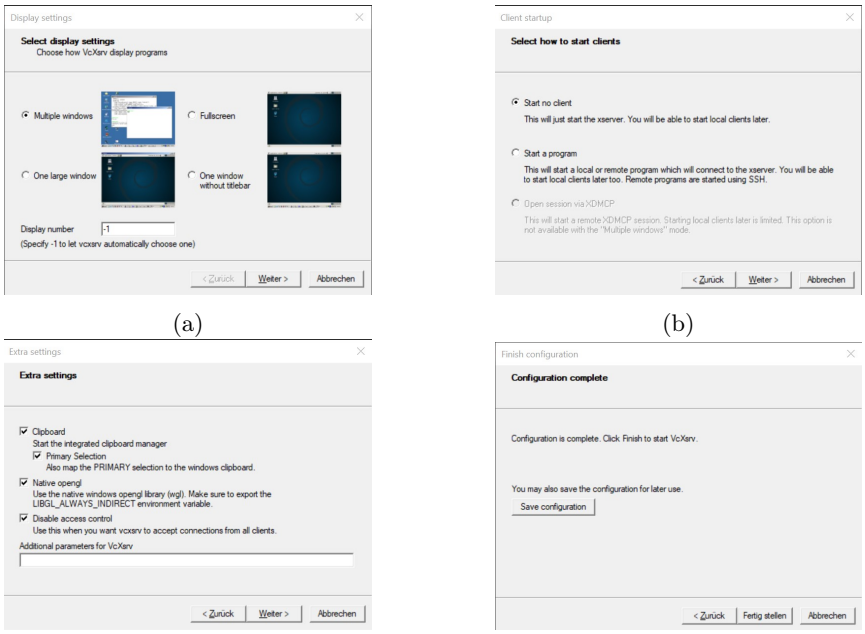

Figure 1: XLaunch configurations.

```
C:\Users\jakob\Documents\JKU\IIC\iic-osic-tools-main>dir
 Volume in Laufwerk C: hat keine Bezeichnung.
 Volumeseriennummer: 1C34-C0ED

 Verzeichnis von C:\Users\jakob\Documents\JKU\IIC\iic-osic-tools-main

06.10.2022  08:38    <DIR>          .
06.10.2022  08:38    <DIR>          ..
06.10.2022  08:37                22 .gitignore
06.10.2022  08:37             2 560 build-all.sh
06.10.2022  08:37               922 builder-clear.sh
06.10.2022  08:37               593 buildkitd.toml
06.10.2022  08:37            17 703 Dockerfile
06.10.2022  08:38    <DIR>          images
06.10.2022  08:37            11 415 LICENSE
06.10.2022  08:38    <DIR>          old_build
06.10.2022  08:37            18 673 README.md
06.10.2022  08:37             2 510 start_shell.sh
06.10.2022  08:37             2 226 start_vnc.bat
06.10.2022  08:37             2 871 start_vnc.sh
06.10.2022  08:37             1 840 start_x.bat
06.10.2022  08:37             4 844 start_x.sh
06.10.2022  08:38    <DIR>          tools
06.10.2022  08:37             2 078 tool_metadata.yml
06.10.2022  08:37             1 718 tool_metadata_add.yml
              14 Datei(en),         69 975 Bytes
               5 Verzeichnis(se), 554 738 192 384 Bytes frei

C:\Users\jakob\Documents\JKU\IIC\iic-osic-tools-main>.\start_x.bat
```

Figure 2: Setting up the `IIC-OSIC-TOOLS` in the terminal.

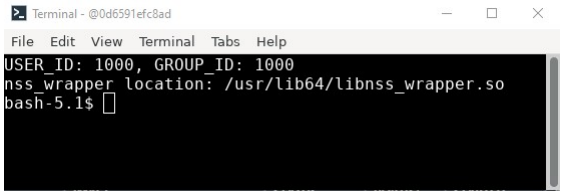

Figure 3: Final result.

4 First simulation

4.1 Create a simple binary counter

and name the file `counter.v`:

```
/*
    Simple counter with generic bitwidth.
*/

`default_nettype none
`ifndef __COUNTER__
`define __COUNTER__

module counter
#(
    parameter BW = 8 // optional parameter
) (
    // define I/O's of the module
    input                   clk_i,          // clock
    input                   rst_i,          // reset
    output wire [BW-1:0]    counter_val_o   // counter value
);

    // start the module implementation

    // register to store the counter value
    reg [BW-1:0] counter_val;

    // assign the counter value to the output
    assign counter_val_o = counter_val;

    always @(posedge clk_i) begin
    // gets active always when a positive edge of the clock signal occours

        if (rst_i == 1'b1) begin
            // if reset is enabled
            counter_val <= {BW{1'b0}}; //reset the counter value
        end else begin
            //increment the counter value by 1
            counter_val <= counter_val + {{(BW-1){1'b0}}, 1'b1};
        end
    end

endmodule // counter

`endif
`default_nettype wire
```

4.2 Create a test bench for the counter

and name the file `counter_tb.v`:

```
/*
    Simple testbench for the counter.
*/

`timescale 1ns / 1ns

`include "counter.v"

module counter_tb;

    parameter BW = 3;

    // inputs
    reg         rst_i = 1'b1;
    reg         clk_i = 1'b0;
    wire [BW-1:0]   cnt_val;

    //DUT
    counter
        #(BW)
        counter_dut (
            .clk_i(clk_i),
            .rst_i(rst_i),
            .counter_val_o(cnt_val)
        );

    //Generate clock
    /* verilator lint_off STMTDLY */
    always #5 clk_i = ~clk_i;
    /* verilator lint_on STMTDLY */

    initial begin
        $dumpfile("counter_tb.vcd");
        $dumpvars;

        /* verilator lint_off STMTDLY */
        #50 rst_i = 1'b0; // deassert reset
        #200 $finish; // finish
```

```
39          /* verilator lint_on STMTDLY */
40      end
41
42  endmodule // counter_tb
```

4.3 Save the files in the working directory

Go to the directory `C:\Users\USERNAME\eda\designs\` and make a new subdirectory named `\FirstSimulation`. Now save `counter.v` and `counter_tb.v` in this directory.

If your terminal window isn't open, you could easily start a new one, by restarting the container in the Docker application.

Now check if the files are in the `\FirstSimulation` directory. If all worked out well, the result should look similar as in figure 4.

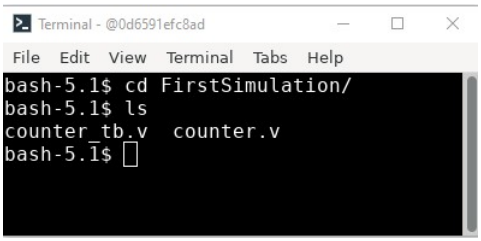

Figure 4: File navigation.

4.4 Check the code

To check the code for possible syntax errors (or typical coding mistakes), the tool `iic-vlint.sh` can be used. The syntax is as follows:

```
> iic-vlint.sh <file.v>
```

4.5 Compile the program

To compile the program, `Icarus Verilog` will be used. For compiling the `Verilog` code, the syntax is as follows:

```
> iverilog -g2005 -o <OUTPUTFILE_NAME> <file.v>
```

Now compile the program by

```
> iverilog -g2005 -o COUNTER counter_tb.v
```

4.6 Execute the compiled program

The compiled program can be executed by the following command:

```
> vvp <OUTPUTFILE_NAME>
```

Therefore, to execute our counter, type in

```
> vvp COUNTER
```

If everything worked out correctly, the results should look like in figure 5.

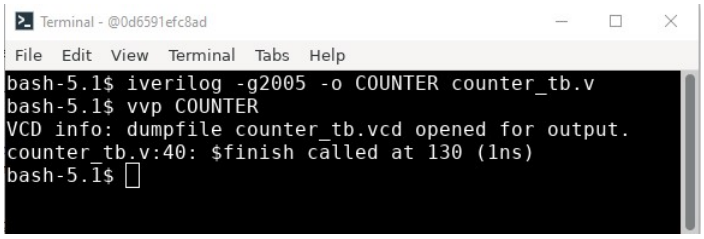

Figure 5: Compiled and executed counter.

4.7 Look at the waveform

Since we dumped all variables in the result file `counter_tb.vcd` we can investigate our counter design. For this, we use `gtkwave`, which is a waveform plotting tool for digital signals.

To view the waveforms of our counter, type in and execute:

```
> gtkwave counter_tb.vcd
```

Now a new window pops up and the waveforms can be investigated. The results should look like in figure 6.

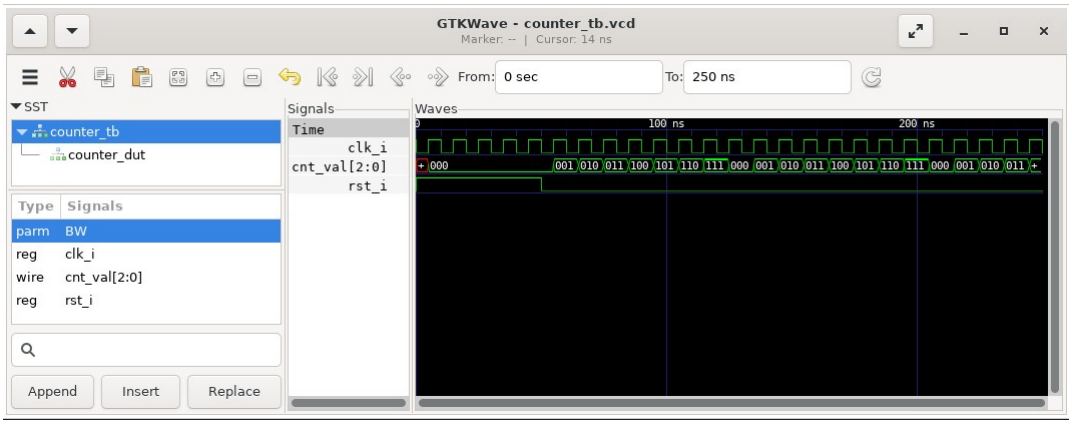

Figure 6: Waveforms of the counter.