VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh
Department of Electronics and Communication Engineering
IIIT-Delhi

Tutorial 4
Lecture 18
Simulation-based Verification using Icarus

Hello everyone, my name is Amina Haroon and I am a PhD student at IIIT Delhi and I will be your TA for the course VLSI Design Flow. In the previous tutorial, you learned about the high level synthesis. In this tutorial, we will see how to perform a simulation based verification of a digital circuit modelled in Verilog. To do that, I will be using an open source tool called Icarus Verilog, but you can use any other tool as well. So let us begin. Also in this video, I will be referencing a tutorial sheet and that sheet will be shared with you.

So we will see how to install the Icarus Verilog. So I expect that you already have Linux distribution on your system. I am using Ubuntu. Now you need to launch the terminal of Ubuntu and type the following command git clone and go to the website of iverilog and copy the link from there.

```
$ git clone https://github.com/steveicarus/iverilog.git
```

So this will start cloning the GitHub repository. So it is done now. Let us check the contents of the directory by typing in the following command that is

```
$ ls
```

So a folder is formed that is iverilog. You need to change into the iverilog directory.

```
$ cd iverilog
```

Once we are in that directory, we will start building the configuration files. The command for that is

```
$ sh autoconf.sh
```

But it says that there are a few dependencies missing. So that is gperf and autoconf.

First you need to do the

```
$ sudo apt-get update
```

to update the existing packages. So it is done now. Then you need to start installing the dependencies one by one. So these commands are already written in your tutorial sheet. So the command for installing a dependencies is

```
$ sudo apt-get install gperf
```

```
$ sudo apt-get install autoconf
```

So it is getting installed. Let us wait for a moment. So it is done now. So the next step is to build the configuration files again because it was not done previously because of the missing dependencies. So it is completed.

```
$ sh autoconf.sh
```

The next step is to configure and the command for that is

```
$ ./configure
```

So again it requires few more dependencies, that is, C compiler. So you will see the description of the error in the tutorial sheet. So it says that there is no C compiler and few more packages that I have already listed in the sheet.

So you need to install them one by one. So the next dependency that I am installing is flex. Another one is bison. So the dependencies are installed. Let us go back to the configure because it was interrupted.

```
$ sudo apt -get install gcc g++
```

```
$ sudo apt -get install flex
```

```
$ sudo apt -get install bison
```

```
$./configure
```

So the configure part is done and you can see that configure.h file is showing in your terminal. Now you need to compile the source code.

```
$ make
```

So again it says that the make is not found.

So you need to install the make package.

```
$ sudo apt-get install make
```

Make is installed. You need to run the compile source code command again which is

```
$ make
```

This will require a few minutes. So the make is done.

Now the next step is to do the

```
$ sudo make install.
```

So type in the command. So the installation is done. So you need to first leave the current directory by typing cd and then launch the tool that is iverilog. So since there is no source file, it is not doing anything. We will see how to use this tool with an example later in the demo.

```
$ cd

$ iverilog
```

Another tool that we require is a waveform analysis tool. So I am using GTKwave tool which is an open source tool and you can use any other tool as well. So let us begin. So to install the GTKwave tool, you need to first go to your Linux distribution and type in the following command that is

```
$ sudo apt install gtkwave
```

So this will take few minutes to run.

So it's done now. You can launch the tool by typing

```
$ gtkwave
```

and it looks like this. Another tool that I will be using is code coverage tool called Covered which is an open source tool and it is used to check whether the test bench has the diversified set of a stimuli. But you can use any other tool as well. So let's start. So the steps to install the Covered are shown here.

So first you need to clone the GitHub repository. In the terminal type in the following command git clone and the path to the GitHub repository.

```
$ git clone https :// github .com/ chiphackers / covered
```

So it's done. So now I have the covered directory in my working path. You need to change to the covered directory.

And let's start the installation process. So let's type in the command

```
$ ./configure
```

It's done now. The next step is to compile the source code by using the command

```
$ make
```

However, it comes back with error.

So it says that it requires few more dependencies and I have provided the list of them in the tutorial sheet. So for that first you need to update the existing packages using

```
$ sudo apt update
```

So I'm showing how the first one is installed. You can install them one by one using the same process. So the required dependencies are installed.

```
$ sudo apt-get install zlib1g -dev
```

```
$ git clone https://git.savannah.gnu.org/git/libiconv.git
```

```
$ sudo apt-get install tcl8.6
```

```
$ sudo apt-get install tcl8.6-dev
```

```
$ sudo apt-get install tk8.6
```

```
$ sudo apt-get install tk8.6-dev
```

```
$ sudo apt-get install doxygen
```

Let's do the configure again. Configure is done now. Let's do the make again. This will take few minutes.

```
$ ./configure
```

```
$ make
```

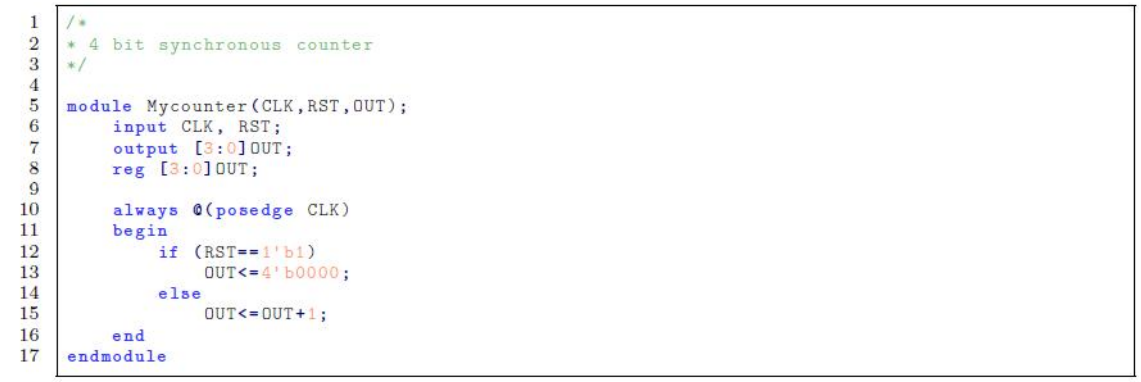
So the make is done. The next step is to do

```
$ sudo make install
```

So the covered is installed. Let's move to the demonstration of Icarus and covered using an example. So let's make a directory where you will be keeping your iverilog files. Since I already have a directory, so I will just change into the icarus_codes.

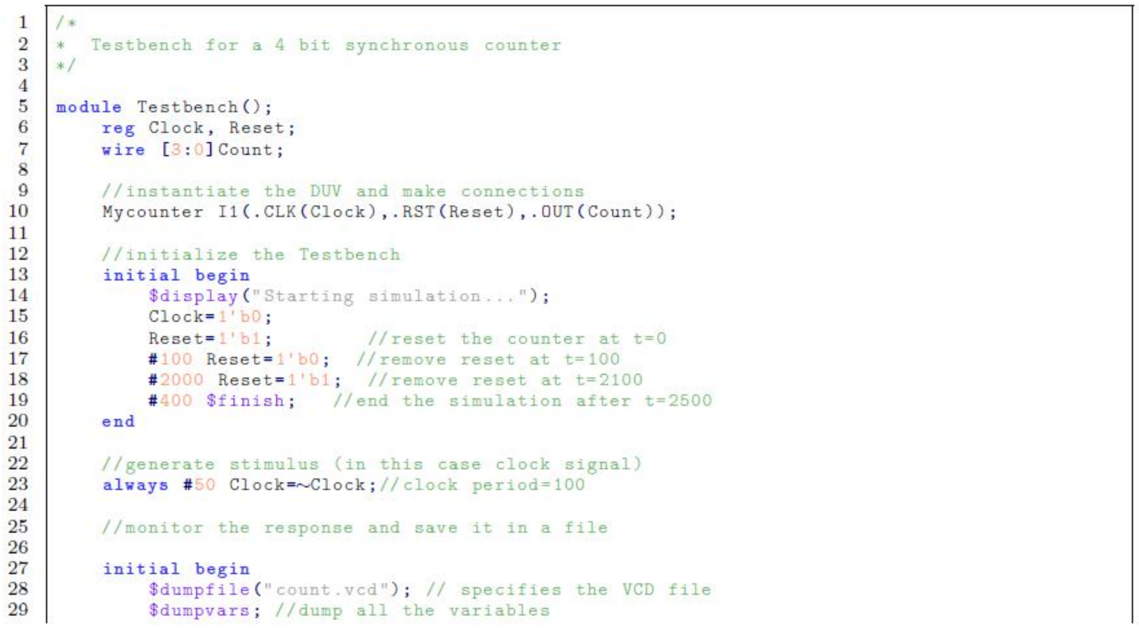
In this directory, you need to have a verilog code and the test bench for your design under verification. So let's check the content of my directory and here I have my counter which is a 4-bit synchronous counter and a test bench. So let's see how the code looks like. I'm using gedit but you can use a nano or vim etc. So this is how the code looks like and this is the test bench.

5.1 Design Under Verification

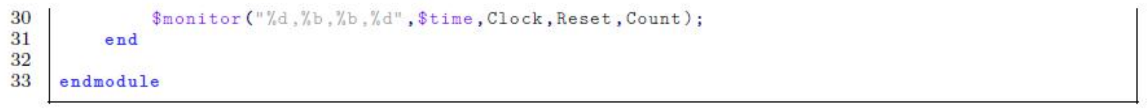


```verilog
/*
* 4 bit synchronous counter
*/

module Mycounter(CLK,RST,OUT);
    input CLK, RST;
    output [3:0]OUT;
    reg [3:0]OUT;

    always @(posedge CLK)
    begin
        if (RST==1'b1)
            OUT<=4'b0000;
        else
            OUT<=OUT+1;
    end
endmodule
```

5.2 Testbench



```verilog
/*
*  Testbench for a 4 bit synchronous counter
*/

module Testbench();
    reg Clock, Reset;
    wire [3:0]Count;

    //instantiate the DUV and make connections
    Mycounter I1(.CLK(Clock),.RST(Reset),.OUT(Count));

    //initialize the Testbench
    initial begin
        $display("Starting simulation...");
        Clock=1'b0;
        Reset=1'b1;        //reset the counter at t=0
        #100 Reset=1'b0;  //remove reset at t=100
        #2000 Reset=1'b1;  //remove reset at t=2100
        #400 $finish;   //end the simulation after t=2500
    end

    //generate stimulus (in this case clock signal)
    always #50 Clock=~Clock;//clock period=100

    //monitor the response and save it in a file

    initial begin
        $dumpfile("count.vcd"); // specifies the VCD file
        $dumpvars; //dump all the variables
```



```
        $monitor("%d,%b,%b,%d",$time,Clock,Reset,Count);
    end

endmodule
```

So this is my counter and it has the input signal RST which is a reset signal, the clock CLK and the output. So at the positive edge of the clock, if reset is 1, the counter sets to 0 and if the reset is 0, then the counter increments at every positive clock edge. Let's see the test bench. So here the input to my design under verification is RST, CLK and OUT. In the test bench, there is a Reset signal, clock signal and Count signal.

So you need to connect them accordingly. So it's instantiated and let's explain the test bench. So at t is equal to 0, the clock is 0 and the reset is 1. At t is equal to 100, the reset is removed and the counter will increment and similarly the other scenarios. Also the clock period is 100 time steps and we will save the output in count.vcd file and it will have clock, reset and count. To simulate, I will be launching the iverilog tool. So the command goes like this.

$ iverilog -o Mycounter Mycounter .v Testbench .v

So -o tells the tool to save the output in my counter file. Next I will need the design under verification that is my counter.v and the test bench that is testbench.v. This command compiles the program. So let's check the contents of the directory. So you can see that there is a my counter file in the present working directory. Now the next command is

$ vvp Mycounter

So this command will execute the compiled program and now my the outputs are stored in the dump file called count.vcd. So it's not very convenient to view them in the terminal. So I will be using the waveform analysis tool the GTKwave. So the command is

$ gtkwave count.vcd

So you can see the CLK, the RST and the OUT.

So let's analyze the result. At the first positive edge, since the reset was 1, the counter is reset to 0. At the next clock edge, it goes to 1 and 2 and 3 and so on. So this is in hexadecimal. So it goes up to f then 0, 1, 2, 3, it counts again. Since the reset was 0, but here the reset goes to 1.

However, this happens at the negative clock edge. So here the reset goes to 1. The effect of that is visible at the positive clock edge and the counter sets to 0. So now we will check the code coverage using the same example of counter. Since I'm already in the present directory, let's check first the contents of the directory.

So it has the vcd file, the design under verification and the testbench. So the command to check the code coverage report is this, and it will save the output in the .cdd file.

```
$ covered score -t Testbench -v Testbench.v -v Mycounter.v -vcd count.vcd -o Mycounter.cdd
```

So it says that this scoring is completed successfully.

Let's check the contents of the directory. So we can see that mycounter.cdd is now in the present working directory. So to view the coverage report, the command is this. So here it shows the code coverage report.

```
$ covered report -d v Mycounter .cdd
```

So you can see the covered tool launched. It shows the line coverage results, the toggle coverage results, the combinational logic coverage results and the FSM coverage results. You can study them and you can look into the details of this result and this completes this tutorial. Thank you very much.