# CS 207 Digital Logic - Spring 2020
# Lab 1

James Yu

Feb. 19, 2020

## Objective

1. Install Verilog simulator.
2. Learn compilation and execution of your first Verilog program in command line.

## Course Logistics

1. **Lab exercises**: The content of lab exercise is prepared according to the knowledge introduced in lectures. From next week on, exercises of each lab session count towards a maximum 2% of the final grade.
2. **Lab attendance**: From next week on, we will record the attendance of lab sessions. If you attend the lab session, you will get at least 1% of the above 2% per se.
3. **Assignment**: The assignment is independent from the lab exercise, and each assignment counts towards 3% of the final grade.

## Lab Exercise Submission

1. You should name all source code as instructed. Mis-named files will not be recognized.
2. You should submit all source code files with an extension "`.v`".
3. You should submit all source code directly into the sakai system below. **Do not compress them into one folder.**

   `https://sakai.sustech.edu.cn/portal/site/ebf68254-68c5-4cfe-9d42-b26758f854ee`
4. Lab exercises should be submitted before the deadline, typically one week after the lab session. No late submission policy applies to lab exercises.

# 1 Software Installation

To design the circuits it is essential to have a Verilog simulator. The free tools we will use are:

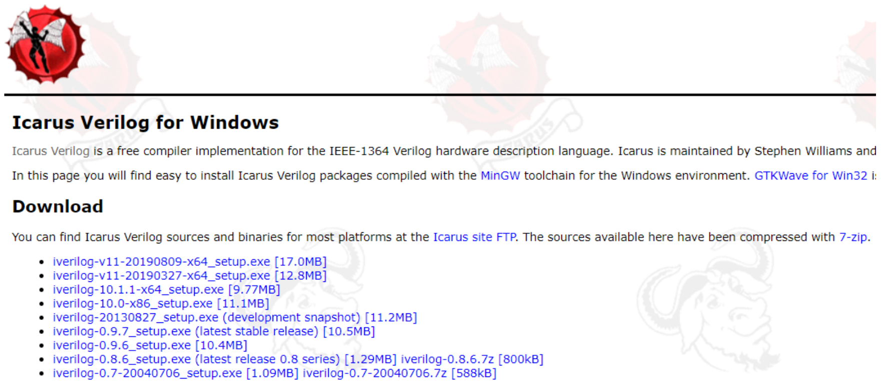
- **Verilog Simulator**: `Icarus Verilog`
- **Waveform Viewer**: `GTKwave`

`Icarus Verilog` creates an executable file from the Verilog code. When executed, the simulation is performed. The results are dumped into a `.vcd` file which is displayed with the `GTKwave` tool. This allows us to inspect the signals to verify their correct operation.

## 1.1 Install `Icarus Verilog`

### 1.1.1 Windows

Download `Icarus Verilog` from `https://bleyer.org/icarus/` under section “Download”. Select the first setup file (“`iverilog-v11-20190809-x64_setup.exe`” as of Jan. 2020).



Icarus Verilog for Windows

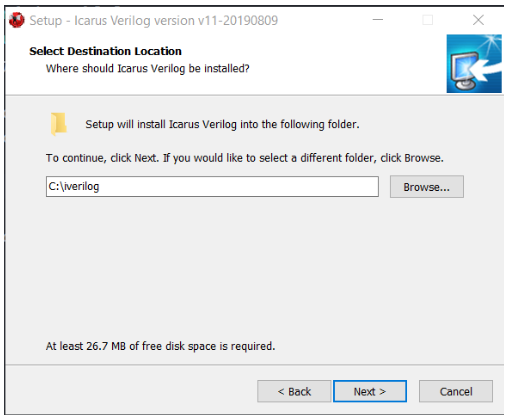
Icarus Verilog is a free compiler implementation for the IEEE-1364 Verilog hardware description language. Icarus is maintained by Stephen Williams and

In this page you will find easy to install Icarus Verilog packages compiled with the MinGW toolchain for the Windows environment. GTKWave for Win32 i

Download

You can find Icarus Verilog sources and binaries for most platforms at the Icarus site FTP. The sources available here have been compressed with 7-zip.

- iverilog-v11-20190809-x64_setup.exe [17.0MB]
- iverilog-v11-20190327-x64_setup.exe [12.8MB]
- iverilog-10.1.1-x64_setup.exe [9.77MB]
- iverilog-10.0-x86_setup.exe [11.1MB]
- iverilog-20130827_setup.exe (development snapshot) [11.2MB]
- iverilog-0.9.7_setup.exe (latest stable release) [10.5MB]
- iverilog-0.9.6_setup.exe [10.4MB]
- iverilog-0.8.6_setup.exe (latest release 0.8 series) [1.29MB] iverilog-0.8.6.7z [800kB]
- iverilog-0.7-20040706_setup.exe [1.09MB] iverilog-0.7-20040706.7z [588kB]



Setup - Icarus Verilog version v11-20190809

Select Destination Location

Where should Icarus Verilog be installed?

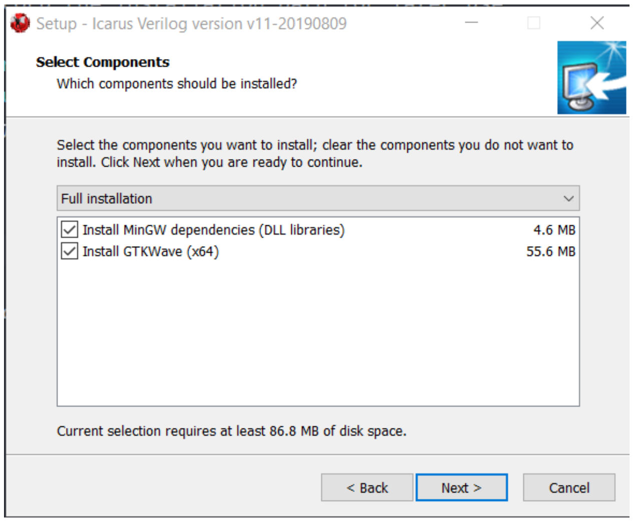
Setup will install Icarus Verilog into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\iverilog

At least 26.7 MB of free disk space is required.

Once downloaded, run the executable. Follow the prompt to install `Icarus Verilog`. Do record the installation path for later use. Install both components as shown below, and tick “Add executable folder(s) to the user PATH”.


Setup - Icarus Verilog version v11-20190809

**Select Components**

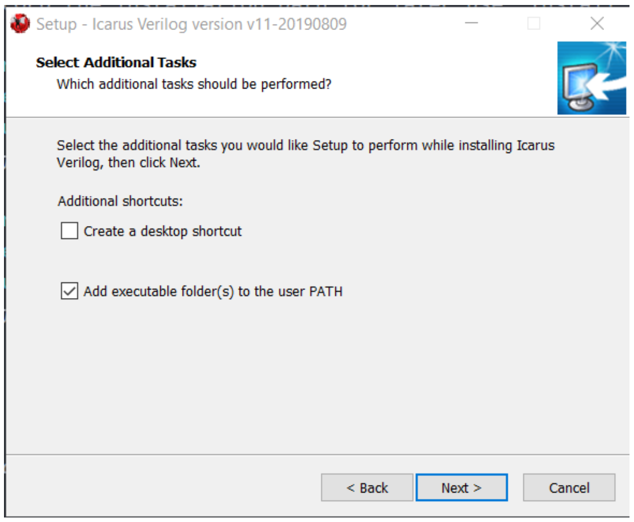
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

| | |
|---|---|
| ✓ Install MinGW dependencies (DLL libraries) | 4.6 MB |
| ✓ Install GTKWave (x64) | 55.6 MB |

Current selection requires at least 86.8 MB of disk space.


Setup - Icarus Verilog version v11-20190809

**Select Additional Tasks**

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Icarus Verilog, then click Next.

Additional shortcuts:

☐ Create a desktop shortcut

✓ Add executable folder(s) to the user PATH

Open a command prompt (on Windows 10, Start - Windows Systems - Command Prompt), type the following command

```
> iverilog
```

If it returns as follows, you have installed it successfully.

```
> iverilog
iverilog: no source files.

Usage: iverilog [-EiSuvV] [-B base] [-c cmdfile|-f cmdfile]
```

```
                [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] [-l file] source_file(s)

See the man page for details.
```

### 1.1.2 Debian-based Linux distributions

Open a terminal, type the following command

```
> sudo apt install iverilog
```

After installation, try

```
> iverilog
```

If it returns as follows, you have installed it successfully.

```
> iverilog
iverilog: no source files.

Usage: iverilog [-EiSuvV] [-B base] [-c cmdfile|-f cmdfile]
                [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] [-l file] source_file(s)

See the man page for details.
```

### 1.1.3 Mac OS

Press `Command+Space` and type `Terminal` and press `enter` key. Then run the following in Terminal app:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/
install)" < /dev/null 2> /dev/null
```

If the screen prompts you to enter a password, please enter your Mac's user password to continue. Then wait for the command to finish.

Run:

```
brew install icarus-verilog
```

then

```
> iverilog
```

If it returns as follows, you have installed it successfully.

```
> iverilog
iverilog: no source files.

Usage: iverilog [-EiSuvV] [-B base] [-c cmdfile|-f cmdfile]
                [-g1995|-g2001|-g2005|-g2005-sv|-g2009|-g2012] [-g<feature>]
                [-D macro[=defn]] [-I includedir]
                [-M [mode=]depfile] [-m module]
                [-N file] [-o filename] [-p flag=value]
                [-s topmodule] [-t target] [-T min|typ|max]
                [-W class] [-y dir] [-Y suf] [-l file] source_file(s)

See the man page for details.
```

### 1.2 Verilog Editor

For all platforms, we recommend to use Visual Studio Code (`https://code.visualstudio.com/`) with the Verilog HDL/SystemVerilog extension. Or you can simply use any text editor, for example notepad on Windows and TextEdit on Mac.

## 2 Exercise

### 2.1 Simulate a hardware using Verilog

Now we start to create a hardware, or at least simulate one. The simplest digital circuit is just a wire connected to a known logic level, `1`, for example. This way, if you connect a LED to it, it will light up (`1`) or turn off (`0`).

The Verilog code for this “hello world” circuit implementation can be coded in a `setbit.v` file. It looks like this:

`setbit.v`

```
module setbit(output A);
wire A;
assign A = 1;
endmodule
```

When we work with FPGAs we are actually making hardware and we always have to be very careful. We could write a design that contains a short circuit. And it could also happen that the tools don't warn us, (especially in these first versions, still in an alpha stage). If we upload that circuit into the FPGA we could break it.

Because of that, we always simulate the code that we write. Once we are sure enough that it works, (and it doesn't have a critical mistake) we upload it into the FPGA.

However, you can't simulate a verilog component right away, you need to write a testbench that indicates which cables connect to which pins, which input signals to send the circuit, and check that the circuit outputs the right values. This testbench file is also written in Verilog.

How do we test the `setbit` component? It's a chip that has only one output pin, always high. In real life we would plug it onto a breadboard, we power it up, we would connect a cable to the output pin, and we would check it's high in voltage with a multimeter. We will do exactly the same, but with Verilog. We would get something like this:

`setbit.v`

```
module setbit_tb;
//-- Cable for connecting the output pin to setbit
//-- We could give it ANY name, but we call it A (like the setbit pin)
wire A;
//-- Place the component (it's actually called "instance") and
//-- connect the A cable to the A pin.
setbit SetBit (.A (A));

//-- Begin the test (Checking block)
initial begin
//-- After 10 time units, check if the cable is high.
//-- In case of not being high, report the problem but
//-- don't stop the simulation.
  # 10 if (A != 1)
    $display("---->ERROR! Output is not 1");
  else
    $display("Component works!");
//-- End simulation 10 time units after checking
  # 10 $finish;
end
endmodule
```

Finally, we use `Icarus Verilog` for simulation. We execute the following commands one by one:

```
> iverilog -o setbit.o setbit.v # Both the design and testbench are compiled
> vvp setbit.o # The result is a simulation file, which can be executed by vvp
provided by Icarus Verilog
```

See what is output into the command line. Does it meet your expectation?

## 2.2 Edit the hardware a bit

Now, instead of just one bit we will output four of them. It's a fixed value, wired "within the hardware". If we wanted to use the circuit for another value, we will need to synthesize again. We will name this component `fport` (Fixed port). It has a 4-bit output bus, labeled as `data`, wired to the binary value `1010`.

This circuit is similar to the one in the last tutorial, but instead of having just 1 output bits, it has 4 bits. It's described like this:

fport.v

```verilog
module fport(output [3:0] data);
//-- Module output is a 4 wire bus.
wire [3:0] data;
//-- Output the value through that 4-bit bus.
assign data = 4'b1010; //-- 4'hA
endmodule
```

The output is now a 4 wire array. You can express that by writing `[3:0]` before the wire name. In order to assign a value to that wire, we write the value using Verilog's notation: First, the number of bits, then the `'` character, after that, the base of the number (in this case, `b` for "binary") and finally, the 4 binary digits. This number could be expressed like a single hexadecimal digit, by writing `4'hA`. We could also write it in decimal as `4'd10`.

This testbench is similar to the one in the last chapter. But now, instead of checking just one bit, we check the whole 4-bit bus. If it's not exactly as expected, it throws an error. The testbench code is:

fport.v

```verilog
module fport_tb;
//-- 4-wire bus, to connect it to the Fport component output.
wire [3:0] DATA;
//--Instantiating the component. Connect output to DATA.
fport FP1 (.data (DATA));

//-- Begin the test
initial begin
  //-- After 10 time units we check
  //-- whether the cable has the previously given pattern or not.
  # 10 if (DATA != 4'b1010)
      $display("---->ERROR!");
    else
      $display("Component works!");
  //-- Finish the simulation 10 time units after that.
  # 10 $finish;
end

endmodule
```

Finally, we use `Icarus Verilog` for simulation. We execute the following commands one by one:

```
> iverilog -o fport.o fport.v # Both the design and testbench are compiled
> vvp fport.o # The result is a simulation file, which can be executed by vvp provided
 by Icarus Verilog
```

See what is output into the command line. Does it meet your expectation?