

ECE 206/COS 306:

Software Installation Guide

You will be using two required programs for Verilog development in this class. The first, **Icarus Verilog**, is a widely-used Verilog compiler. The scripts that we will provide to compile simulations of your code use Icarus Verilog. The second, **GTKWave**, is a simple and easy-to-use waveform viewer. You'll be using it in the labs to view the results of your simulations and debug your Verilog code. In later labs, you will have the opportunity to use Xilinx's Vivado Design Suite to program FPGA boards using the lab computers in the undergraduate ELE lab and get actual working hardware!

However, in a recent update to the course, we have migrated to using a **Docker** image. In essence, a Docker container is a form of virtualization which is faster and smaller than a full virtual machine. The image you will be using already has iVerilog and GTKWave packaged in a Linux environment. Docker is cross-platform and provides a low bar of entry for the growing industry trend of local development.

In addition, we strongly recommend the use of **Visual Studio Code** for code development. VSCode is a free open-source IDE that can perform syntax highlighting on a variety of programming languages, including Verilog. We recommend it because it is fast, cross-platform, and has a whole host of extensions that can be useful with Verilog, Docker, and beyond. All of the examples/tutorials in this course will use VSCode.

There are general instructions for installation as well as some platform-specific for Windows and Unix-like (MacOS/Linux) systems. If you have difficulty getting these installations to work, contact the lab teaching assistants for help.

Installing: VSCode

The easy-to-install VSCode package is available at <https://code.visualstudio.com/>. Navigate there and download the latest version. Go through the installation and accept the license agreement. Configure the optional settings as desired. One of use may be 'AutoSave' which will automatically save an edited file. To enable click on *File*→*Auto Save*. It should have a check next to it once enabled.

NOTE: if you are on MacOS or Linux, please launch VSCode and type *cmd/ctrl+shift+P* and type *Shell Command: Install 'code' command in PATH*. Then restart any shell sessions. This will allow you to launch VSCode from the command line by simply typing *code <file-or-dir-name>*. For example, *code .* will open the current working directory in VSCode. This should be enabled by default on Windows.

Extension: Verilog HDL

Now open the application (which should be in your Start Menu) and click on the last icon on the left-most bar. This should bring up the extensions toolbar. Search for **Verilog HDL** and install the extension named "Verilog-HDL/SystemVerilog/Bluespec SystemVerilog". This will enable syntax highlighting for Verilog in VSCode. This extension can also be configured to provide syntax checking. This will underline any syntax errors every time a file is saved. To enable syntax checking, click on the gear in the bottom-left of your VSCode window and go to settings. On the left hand side of the settings tab, click on Extensions and navigate to Verilog Configuration. Find the setting named Verilog→Linting: Linter and select iverilog from the dropdown. Lastly, find the setting named Verilog→Linting→Iverilog: Arguments and type the text `-Wall -Wno-timescale`. Syntax checking will work once Icarus Verilog is installed and VSCode is attached to a running container.

Extension: Remote - Containers

The extension **Remote - Containers** allows you to attach a VSCode window to a running Docker container. As iverilog is not installed locally, this will allow you to have the aforementioned real-time linting catch errors early-on! Simply install as was done for the Verilog HDL extension.

Optional Extension: Bracket Pair Colorizer 2

Another very useful extension is [Bracket Pair Colorizer 2](#). A very common issue in Verilog is matching "begin...end" within control structures such as if statements (think like brackets '{}' in C or Java). This extension simply will make matching begin and end keywords distinct and matching colors: possibly helping to avoid pesky syntax errors. Simply install in the same manner as for the previous extensions.

Optional Extension: Live Share

The course has several large partner projects later in the semester. The extension [Live Share](#) comes in very handy as it allows to to have a single, simultaneous programming session (similar to google docs) all within VSCode. Helpful documentation for the extension can be found [here](#).

Installing: Docker

Installing Docker is fairly straightforward. Go to the [Docker Website](#) and download the application corresponding to your operating system. Follow the instructions to install, restart your computer, and launch Docker Desktop when you log back in. For Windows users, you can safely ignore or reject any mentions of WSL upon installation.

NOTE: Docker Desktop must *always* be running in order when you want to start a container.

Installing: Command Line Tools

In order to provide some level of ease-of-use for this class, a few scripts have been developed. This should be the only part of installation that will be platform specific.

Unix-like Systems (MacOS/Linux)

First, make sure Docker is running. Download the provided bash script specified for posix/unix in module on the course canvas page. Open terminal on a mac or bash on Linux and navigate to the directory containing the script *ece206* (typically your downloads folder). Once there, type the commands:

```
1 chmod +x ./ece206
2 ./ece206 init
```

This will create a directory (`~/ece206`) in your home folder in which *all* project directories must be located in order to be available within the Docker container. It will also pull the latest version of the course's Docker image from the web and you may be prompted to allow the process web access. Restart your shell session, and type:

```
1 ece206 --help
```

If you get a bunch of text describing the script in return, you may move on.

Windows Systems

First, make sure Docker is running. Download the provided PowerShell script specified for windows in module on the course canvas page. Open PowerShell (default on systems running Windows 7 and above) and navigate to the directory containing the script `ece206.ps1` (typically your downloads folder). Once there, type the commands:

```
1 Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
2 code ./ece206.ps1 $PROFILE
```

The first command will allow any scripts made on your computer to execute, for more information see Microsoft's description [here](#). The last command should open two files in VSCode. Copy the contents of `ece206.ps1` into the other profile document and save it. Restart PowerShell and run the following command:

```
1 ece206 init
```

This will create a directory (`$HOME\ece206`) in your home folder in which *all* project directories must be located in order to be available within the Docker container. It will also pull the latest version of the course's Docker image from the web and you may be prompted to allow the process web access. Restart your shell session, and type:

```
1 ece206 --help
```

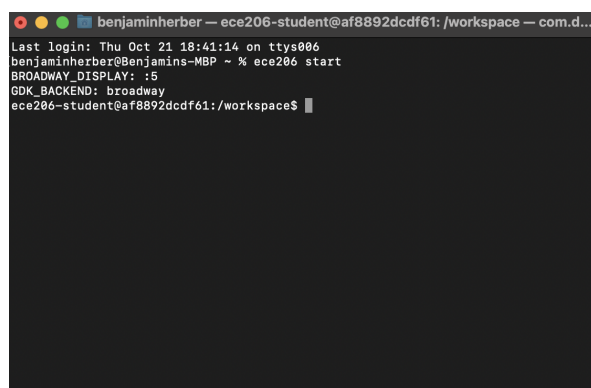
If you get a bunch of text describing the script in return, you may move on.

Verification

Time to test if everything is working! Make sure Docker is running. Then, open your shell program (Terminal/Bash/PowerShell) and type the following:

```
1 ece206 start
```

You should now roughly see the following:

A terminal window titled 'benjaminherber — ece206-student@af8892dcdf61: /workspace — com.d...' showing the output of the 'ece206 start' command. The output includes the last login time, the user 'benjaminherber@Benjamins-MBP', the command 'ece206 start', environment variables 'BROADWAY_DISPLAY: :5' and 'GDK_BACKEND: broadway', and the current shell prompt 'ece206-student@af8892dcdf61: /workspace\$'.

Once here we can start tinkering with the software included in the image (iVerilog, Verilator, and GTKWave). You can exit the container once done by either typing *exit* or pressing *ctrl+D*.

iVerilog

Test iVerilog by running:

```
1 iverilog -V
```

You should see some legal text and other information ending with roughly the following:

```

benjaminherber — ece206-student@af8892dcdf61: /workspace — com.d...
You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

FLAGS DLL vvp.tgt
vvp.tgt: Icarus Verilog VVP Code Generator 11.0 (stable) ()

Copyright (c) 2001-2020 Stephen Williams (steve@icarus.com)

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License along
with this program; if not, write to the Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

ece206-student@af8892dcdf61:/workspace$

```

GTKWave

Test GTKWave by running the following:

```
1 gtkwave --version
```

You should see the following:

```

benjaminherber — ece206-student@af8892dcdf61: /workspace — com.d...
ece206-student@af8892dcdf61:/workspace$ gtkwave --version
GTKWave Analyzer v3.3.111 (w)1999-2020 BSI

This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
ece206-student@af8892dcdf61:/workspace$

```

GTKWave in Browser

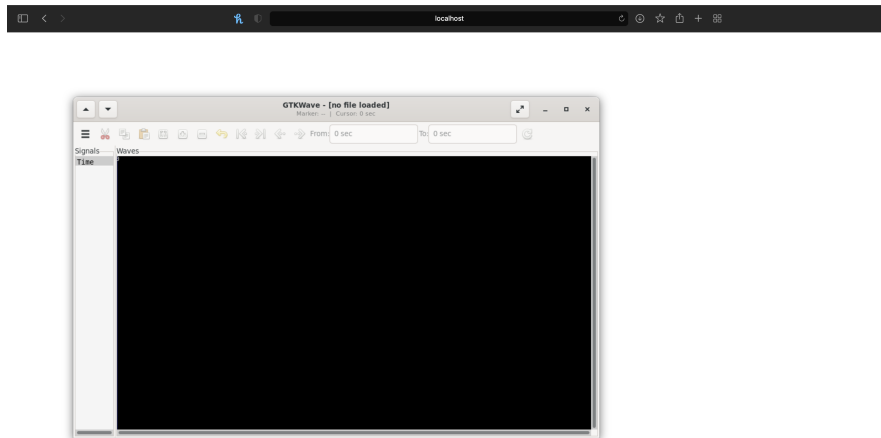
The way that we are able to access the GTKWave GUI is through a web server project called **broadway**. Everything should be handled for you such as port-forwarding etc. To test access, simply launch GTKWave without a waveform:

```
1 gtkwave
```

Now open your favorite browser (Chrome/Safari/Firefox/etc.) and type in the address bar:

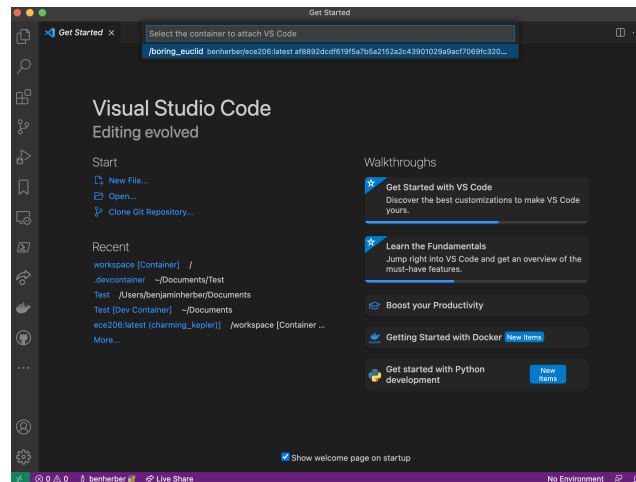
```
1 localhost:8085
```

You should see the following:

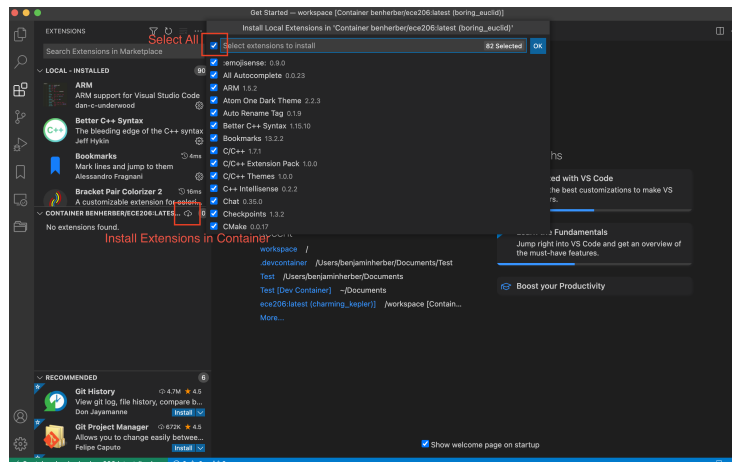


How to Connect VSCode to a Container

In order for linting to work, VSCode needs to be able to find the iVerilog or Verilator binaries depend on which you choose to use. First, make sure that you have an active container open (i.e. you have ran and haven't closed *ece206 start*). Then launch VSCode and press *ctrl/cmd+shift+P*. Then type in the command palette; *Remote-Containers: Attach to Running Container...* and press enter/return. You should see something like the following:



If you have multiple options, select the one containing *benherber/ece206*. It may take a moment for VSCode to connect and install the VSCode server within the container, but once it completes, we need to navigate to the mounted directory. In order to do so, click on the File in the NavBar and press *Open Folder* (you may need to go to the toolbar *File*→*Open Folder...*). and enter */workspace* in the prompt. Then press *OK*. After this, any files you have in *~/ece206* or *\$HOME/ece206* on MacOS/Linux or Windows respectively should appear in the file explorer on the side. Also, you'll need to enable extensions in the container. You can do this by going to the extensions tab on the toolbar and pressing the download button, selecting the preferred extensions, and pressing *OK*. The relevant buttons are shown in the following:



Congrats you did it!