

ece 260c. Lab 0 Welcome & Tools Setup

Your Name / Your PID

Welcome to ECE 260C. In this lab, you will get started with the tools that you will later use in this course.

Section I Basic Setup

In this section, you will follow the Software Setup guide, [linked here](#), from beginning to end. If you ever switch or reset machines, you should revisit this guide.

Q1.1

Get the container shell running interactively as noted in the guide. Run the following command and then paste a screenshot here:

```
echo <your PID>
```

Now, we need to load OpenROAD-flow-scripts, a comprehensive, configurable flow for OpenROAD that includes many example designs and PDKs. For this course, we will work with the [IHP 130](#) open-source manufacturable process. First, however, we should sanity check ORFS.

For your convenience, **you should create a directory in the container's default root** called /work. cd into it and run the orfs_copy commands to load ORFS with a custom directory name:

```
mkdir /work
cd /work
orfs_copy orfs_lab0 # This is a convenience command provided by the
```

container to save time. You can also download ORFS using a git clone.
`cd orfs_lab0/flow` # The actual flow is a subdirectory of the ORFS repo.

When we work with ORFS, we stay within the `flow/` directory so paths, unless otherwise specified, are relative to `flow/`.

Here, you can run `ls` and see the major elements of the flow:

- `Makefile` – this is what you will use to invoke the flow.
- `scripts` – this contains the flow’s actual behavior. These scripts contain mostly `tcl` commands for using OpenROAD and Yosys. They are invoked by the `Makefile`.
- `platforms` – this contains bundled PDKs, plus any special scripts needed to make them work, including `ihp-sgl3g2`.
- `designs` – For each platform, this includes configurations for a set of bundled test designs that you can run ORFS on.
 - `designs/src` – the common Verilog for the bundled test designs within ORFS

Q1.2 Understanding the `Makefile` and the directory structure, pick a design and platform of your choice, and find the file named `config.mk`, then run

```
cat config.mk
```

Paste a screenshot here and describe the parameters you see, looking them up if need be in the [ORFS Docs](#) Do you recognize any from previous courses?

If the `config.mk` contains many parameters or custom scripting, such as in `flow/designs/asap7/aes/config.mk`, describe only a few of the simpler ones.

For your reference, you can always refer to [these same files on GitHub](#) without spinning up the shell session.

Section II Running the Flow

For the next part of our sanity check, we can try **running an actual design** .

By default, running make without any parameters will build the [GCD design on the Nangate45 process](#). GCD is useful for testing because it's small and will thus finish implementation quickly – we will use it here.

In place of Nangate45, however, which is a non-manufacturable academic PDK, we will use IHP 130.

Run the following to perform the flow:

```
DESIGN_CONFIG=./designs/ihp-sg13g2/gcd/config.mk make
```

Depending on your machine, this can take up to 10 minutes. Please keep in mind that you need GUI working for the flow to complete without error. Please review the end of the output log carefully to ensure it's successful.

When the run is successfully completed, you can `ls` again. You'll notice 4 new directories each with nested subdirectories organized by platform, design, and a variant (defaults to "base") i.e. `./logs/ihp-sg13g2/gcd/base`:

- **logs** – contains the tool logs and machine-readable metrics from OpenROAD and yosys, organized by stage of the flow
- **objects** – this typically contains PDK data copied in temporarily by the flow
- **reports** – this contains Quality-of-Results reports from different stages of the flow. Also contains rendered images
- **results** – this contains the most vital data to the flow, separated by stage:
 - synthesized netlist Verilog files
 - flow-generated tcl scripts for IO placement
 - timing/parasitics .sdc/.spef files
 - OpenDB database files that you can reload into OpenROAD
 - The all-important `6_final.def` and `6_final.gds`, the final result of the implementation.

Q2.1 Now, we can view our final report by running:

```
cat reports/ihp-sg13g2/gcd/base/6_finish.rpt | less
```

With the less command, you can just press enter or use the arrow keys to scroll around the report.

What are your WNS and TNS values? Does the design, as it is, pass timing?

Q2.2 Another useful comm and for shortening reports is tail -n <N>, which retrieves the last N lines. Run it like so:

```
cat reports/ihp-sg13g2/gcd/base/6_finish.rpt | tail -n 30
```

and paste a screenshot here.

Before we begin to deep-dive into some of the most useful flow commands, let's clean up our environment and prepare ourselves to make edits.

Typically, whenever we edit our design or perhaps upgrade OpenROAD, we need to perform a full clean run . To do this, we run the following with the same DESIGN_CONFIG as before (please fill in the design path in place of ...):

```
DESIGN_CONFIG=.../config.mk make clean_all
```

All the intermediate data from the aforementioned 4 folders (specifically, the subfolders containing our specific platform-design-variant combo) will be cleared out.

From this point on, remember to include the same DESIGN_CONFIG before all make commands otherwise ORFS will default to trying to work with nangate45/gcd/base.

Now, if you run ls logs/ihp-sg13g2/gcd/base, you'll notice that the directory still exists but is completely empty. Similarly, in the results directory, only some leftover .tcl or .sdc files remain that will be cleanly overwritten the next time you run the flow.

Q2.3 Paste a screenshot here of the results directory.

Section III Editing Flow Parameters

For this section, you'll need an editor. We highly recommend you follow the Software Setup guide to use Dev Containers in Visual Studio Code.

If you have your own preferred editor, like `vim` or `nano`, you may set it up but keep in mind you will also need an image viewer such as `sxiv` (`vim`, `nano`, and `sxiv` are installed in the container).

The following lab instructions will assume you're using VS Code but, when asked for screenshots, you can paste your equivalent.

You've already seen creating or editing designs centered around the `config.mk` file. Now, we'll edit the configuration for the IHP 130 GCD design we've been using. Alongside a basic set of flow environment variables you see here, more are described in the [ORFS docs](#).

For the following exercise, **we'll modify the aspect ratio (AR) of the core**, which describes the relationship between the height and the width from 0.0 to 1.0 – i.e. an AR of 0.5 means the height of the core is half the width of the core. Typically, this AR, alongside `CORE_UTILIZATION` – the percentage indicating how much of the core's total area should be taken up by standard cells, are used to size a core area.

Look for the flow variable controlling Aspect Ratio in the ORFS docs linked above, then add it in the same `export <VARIABLE> = VALUE` format seen in the rest of the `config.mk` file.

The value you use for this exercise **will actually be determined by your PID** using a simple algorithm: use the last two digits of your PID to form a decimal number. For example, if your PID is A35971234 → your AR would be 0.34.

If the number is less than 0.25, however, add 0.3 to it so if your PID is A16808614 then use the value 0.44.

Q3.1

If you found a number from your PID, write down your PID here, underlining the two digits you used here:

If a number like this cannot be made from your PID or if the later run fails, select a custom AR value and note it down here:

Q3.2 Paste a screenshot of your editor here with your modified `ihp-sg13g2/gcd/config.mk` displayed, showing your augmented AR variable.

Now, making sure you `cleaned`, run the make command again:

```
DESIGN_CONFIG=./designs/ihp-sg13g2/gcd/config.mk make
```

Q3.3 Now, with the flow run, let's use VS Code to view the final PNG render in `reports/ihp-sg13g2/gcd/base/final_all.webp.png`
Paste a screenshot of the viewer window here.

You can look at the other images rendered out to `reports` but, for more advanced interaction, we will now look at two useful commands (remember to use the correct design path in place of ...):

```
DESIGN_CONFIG=.../config.mk make open_final  
DESIGN_CONFIG=.../config.mk make gui_final
```

These two commands allow you to open CLI (command-line interface) and GUI OpenROAD, respectively, with the correct DB and timing data automatically loaded. **It's also possible to load the database at different stages** by replacing the `final` suffix with `floorplan/place/cts/route/grt/final`.

Q3.4 Open the GUI for the final design and paste a screenshot here.

Sometimes, you may need to resize the OpenROAD window to make it render correctly.

With the GUI open, you have many options. In the main design viewer, you can left-click on cells/macros, nets, tracks, and ports. You can also right-click-drag to zoom in. **You may need to do this for your design depending on Aspect Ratio** . You can hit F to zoom back out and fit the whole chip on the screen.

Q3.5 What do you notice about the pin placements? Are they done by an engineer or by the tool? Is there any method to their placement?

Click on a few cells, generically called Instances, to see their database entries on the right side in the Inspector. If you click on a wire net, you can see all the connected cells listed.

Here are some key cell entries visible in this view :

- Block/module are used in hierarchical design, which we will discuss later.
- Master is the specific cell in use. For this PDK, all masters are prefixed with sg13g2_. You can click on this entry to get details about the master itself.
- ITerms are the actual connections to the cell, power and clock included where applicable.

Hover over or click on these to see the nets they are connected to.

- Placement Status/Don't Touch are used by the actual OpenROAD application components to mark what should be modified. For example, setting Placement Status to FIRM before running Global Placement will keep the placer from moving that cell around. We will take advantage of this in a future lab.
- Clicking on Timing/Power allows you to access a report including arrival times, power activities, and total power, as well as references to the Liberty entry/library.
- Source for some D Flip Flops will show the register in Verilog source that the FF backs.

Q3.6 Select a random cell and paste a screenshot here of the Inspector.

The Display Control area on the left-side. Allows you to hide or show different layers and views. You can use this to get clearer insight into how the design was placed and routed. For example, you can hide Layers to see just where Instances were placed or you can hide the top metal layers to see the design without the large power delivery network obscuring cells.

You can also control whether a layer is clickable (the rightmost column of checkboxes). When a layer is deselected here (for example, the Metal5 vertical power wire), you can click on the elements under it.

Another useful feature are heatmaps, which provide key spatial statistics. Open the Heatmaps section and select one at a time (except IR Drop, which may not be populated). If a heatmap is ever too granular, too coarse, or including elements you do not want, you can go to Tools > Heatmaps and change the grid size or other options. For our purposes, the default 10x10 heatmap is sufficient.

Q3.7 Select a random heatmap (except Placement Density or IR Drop) and paste a screenshot here. Discuss what it suggests about the design.

Q3.8 Compare a “hot” area (i.e. a dark red grid square) to surrounding cold(er) ones – what differences can you see between their interior instances and/or nets?

Remember, you can toggle Layers in Display Control to more easily select instances or certain nets.

Return Display Control to how it was before (all Layers shown, heatmaps off, Timing Paths on).

Now, go to the Timing Report tab on the right panel of the GUI (where Inspector is). Here, you can see paths with the lowest slack (or highest WNS) for both Setup and Hold.

In the Setup section, Click on the Slack column header to sort from lowest to highest and click on the top entry. Assuming you're zoomed in enough, you should be able to see the clock path (including buffers) highlighted in blue and datapath highlighted in red (this can be hard to see in the default color scheme without hiding layers).

Q3.9 Without layers hidden, paste a screenshot of the app with the timing path visible and the Timing Report visible. Based on the slack, does the design meet timing at these constraints?

While you have OpenROAD open, you can also check out the following (no need to screenshot):

- The Charts tab in the same right panel as Inspector and Timing Report
- The Find tool
- The Clock Tree Viewer– enable this in the menubar under Windows > Clock Tree Viewer. From there, you can select the Clock Tree Viewer tab in the same right panel. Select “default” and click Update.

This screen shows you the spread of the clock tree as it enters from a port (the red triangle), is buffered (the blue triangles), and is distributed to sets of FFs (highlighted in red at the bottom at the chart). You can click each of these entries to see them in the design and in Inspector.

You can now close out OpenROAD. If prompted whether you want to Hide GUI or Exit, select Exit.

Keep in mind that many of the features you see here are available in the CLI.

If you look at `6_finish.rpt` again (as you saw in **Q2.2**), you can see the timing commands that were invoked to give results similar to the GUI Timing Report.

Similarly, ORFS exports several views, including the IR drop heatmap, as png files in the same directory, as you saw in **Q3.2**.

As we continue this course, you will be able to write your own scripts that take advantage of these same features but often you will also be able to work on top of ORFS.

Section IV Submitting Your Design

Congratulations on running your customized design! In this final section, you will prepare some files for submission to GitHub Classroom. You'll use GitHub Classroom for your later labs and for your final project so this is a chance to figure it out.

First, complete the login procedure as noted in the *Continued Usage– Saving your Files with Git* section in the Software Setup guide. Then, cd back into your /work directory (or, if you didn't create one, your root directory / or home directory ~)

Now go to the assignment:

[Click here to unlock the assignment in GitHub Classroom.](#)

Follow the instructions to join the class and create your submission repo. **Please be attentive when selecting your name.**

When your repo is created, you can clone it by running:

```
gh repo clone ABKCourses/e260c-lab0-YourUsername
```

If this doesn't work, make sure you're using the same account you logged in with.

If you used the wrong account, misselected your name, or could not find your name during the Class Join process, email a TA as soon as you can.

Now, you should see a repo that's empty except for maybe a README.md. You can use the cp command to copy files from ORFS like so (always verify with ls after):

```
cp /work/orfs_lab0/flow/... . # using relative paths, copying into  
your current dir  
cp /work/orfs_lab0/flow/... /work/e260c-lab0-YourUsername # using  
absolute paths
```

Please copy the following files from the flow subdirectories into the root of the repo (you should have no directories in your repo):

- designs/ihp-sg13g2/gcd/config.mk
- reports/ihp-sg13g2/gcd/base/6_finish.rpt

- reports/ihp-sg13g2/gcd/base/cts_core_clock_layout.webp.png
- logs/ihp-sg13g2/gcd/base/6_report.json
- results/ihp-sg13g2/gcd/base/6_final.gds

Now, you need to stage, commit, and push these additions. Using VS Code, you can go to the Source Control tool on the left sidebar, hover over Changes, and click the plus button to stage all the files you just added at once.

Then, type in a commit message such as “Submitting my assignment” and hit Commit. Finally, hit Sync Changes.

If you want to use command -line Git, you can run the following within the repo directory:

```
git add .  
git commit -m "Submitting my assignment"  
git push
```

Tip: When you work on multiple machines or with teammates, ensure you’re communicating and syncing changes regularly to avoid merge conflicts when editing the same files.

When you’ve successfully pushed the changes from VS Code or the shell, refresh the GitHub repo webpage and verify that your commit, with all the required files, is present.

Take this lab document and export it as PDF.

<end>