
CIIC Harness

efabless

May 28, 2022

CONTENTS

1	Dependencies	3
2	Starting your project	5
3	Caravel User Project Detailed Overview	7
4	Table of contents	9
5	Overview	11
6	Prerequisites	13
7	Install Caravel	15
8	Caravel Integration	17
8.1	Repo Integration	17
8.2	Verilog Integration	17
8.3	Layout Integration	18
9	Building the PDK	19
10	Running Full Chip Simulation	21
11	User Project Wrapper Requirements	23
12	Hardening the User Project using OpenLane	25
12.1	OpenLane Installation	25
12.2	Hardening Options	25
12.3	Running OpenLane	26
13	Running MPW Precheck Locally	27
14	Other Miscellaneous Targets	29
15	Checklist for Open-MPW Submission	31

DEPENDENCIES

- Docker: [Linux](#) || [Windows](#) || [Mac with Intel Chip](#) || [Mac with M1 Chip](#)
 - Python 3.6+ with PIP
-

STARTING YOUR PROJECT

1. To start the project you need to first create an empty Git project on Github and make sure your repo is public and includes a README
2. Open your Terminal. Create an empty folder to use as your Caravel workspace, and navigate to it.

```
# Create a directory and call it anything you want
mkdir -p caravel_tutorial

# navigate into the directory
cd caravel_tutorial
```

3. Clone caravel_user_project and setup the git environment as follows

```
# Make sure that "caravel_example" matches the empty github repo name in step 1
git clone -b mpw-6b https://github.com/efabless/caravel_user_project caravel_example
cd caravel_example
git remote rename origin upstream

# You need to put your empty github repo URL from step 1
git remote add origin <your github repo URL>

# Create a new branch, you can name it anything
git checkout -b <my_branch>
git push -u origin <my_branch>
```

4. Now that your git environment is setup, it's time to setup your local environment by running.

```
# make sure to change <directory_name> with the directory you created in step 2
# in this case it is caravel_tutorial
export OPENLANE_ROOT=~/<directory_name>/openlane # you need to export this whenever you
↳ start a new shell

export PDK_ROOT=~/<directory_name>/pdk # you need to export this whenever you
↳ start a new shell

make setup
```

- This command will setup your environment by installing the following:
 - caravel_lite (a lite version of caravel)
 - management core for simulation
 - openlane to harden your design

– pdk

5. Now you can start hardening your design

- To start hardening your project you need - RTL verilog model for your design for OpenLane to harden - A subdirectory for each macro in your project under `openlane/` directory, each subdirectory should include openlane configuration files for the macro

```
make <module_name>
```

For an example of hardening a project please refer to [user_project_example](#)

6. Integrate modules into the `user_project_wrapper`

- Change the environment variables `VERILOG_FILES_BLACKBOX`, `EXTRA_LEFS` and `EXTRA_GDS_FILES` in `openlane/user_project_wrapper/config.tcl` to point to your module
- Instantiate your module(s) in `verilog/rtl/user_project_wrapper.v`
- Harden the `user_project_wrapper` including your module(s), using this command:

```
make user_project_wrapper
```

7. Run simulation on your design

- You need to include your `rtl/gl/gl+sdf` files in `verilog/includes/includes.<rtl/gl/gl+sdf>.caravel_user_project`

NOTE: You shouldn't include the files inside the verilog code

```
# you can then run RTL simulations using
make verify-<testbench-name>-rtl

# OR GL simulation using
make verify-<testbench-name>-gl

# for example
make verify-io_ports-rtl
```

8. Run the precheck locally

```
make precheck
make run-precheck
```

9. You are done! now go to www.efabless.com to submit your project!

CARAVEL USER PROJECT DETAILED OVERVIEW

TABLE OF CONTENTS

- *Overview*
- *Install Caravel*
- *Caravel Integration*
 - *Repo Integration*
 - *Verilog Integration*
 - *Layout Integration*
- *Running Full Chip Simulation*
- *User Project Wrapper Requirements*
- *Hardening the User Project using Openlane*
- *Checklist for Open-MPW Submission*

OVERVIEW

This repo contains a sample user project that utilizes the `caravel` chip user space. The user project is a simple counter that showcases how to make use of `caravel`'s user space utilities like IO pads, logic analyzer probes, and wishbone port. The repo also demonstrates the recommended structure for the open-mpw shuttle projects.

PREREQUISITES

- Docker: [Linux](#) || [Windows](#) || [Mac with Intel Chip](#) || [Mac with M1 Chip](#)
- Python 3.6+ with PIP

INSTALL CARAVEL

To setup caravel, run the following:

```
git clone https://github.com/efabless/caravel_user_project.git
cd caravel_user_project

make install
```

To remove caravel, run

```
make uninstall
```

By default `caravel-lite` is installed. To install the full version of caravel, run this prior to calling make install.

```
export CARAVEL_LITE=0
```


CARAVEL INTEGRATION

8.1 Repo Integration

Caravel files are kept separate from the user project by having caravel as submodule. The submodule commit should point to the latest of caravel/caravel-lite master/main branch. The following files should have a symbolic link to [caravel's](#) corresponding files:

- **Openlane Makefile:** This provides an easier way for running openlane to harden your macros. Refer to *[Hardening the User Project Macro using Openlane](#)*. Also, the makefile retains the openlane summary reports under the signoff directory.
- **Pin order** file for the user wrapper: The hardened user project wrapper macro must have the same pin order specified in caravel's repo. Failing to adhere to the same order will fail the gds integration of the macro with caravel's back-end.

The symbolic links are automatically set when you run `make install`.

8.2 Verilog Integration

You need to create a wrapper around your macro that adheres to the template at [user_project_wrapper](#). The wrapper top module must be named `user_project_wrapper` and must have the same input and output ports as the golden wrapper [template](#). The wrapper gives access to the user space utilities provided by caravel like IO ports, logic analyzer probes, and wishbone bus connection to the management SoC.

For this sample project, the user macro makes use of:

- The IO ports for displaying the count register values on the IO pads.
- The LA probes for supplying an optional reset and clock signals and for setting an initial value for the count register.
- The wishbone port for reading/writing the count value through the management SoC.

Refer to [user_project_wrapper](#) for more information.

8.3 Layout Integration

The caravel layout is pre-designed with an empty golden wrapper in the user space. You only need to provide us with a valid `user_project_wrapper` GDS file. And, as part of the tapeout process, your hardened `user_project_wrapper` will be inserted into a vanilla caravel layout to get the final layout shipped for fabrication.

To make sure that this integration process goes smoothly without having any DRC or LVS issues, your hardened `user_project_wrapper` must adhere to a number of requirements listed at [*User Project Wrapper Requirements*](#).

BUILDING THE PDK

For more information about volare click [here](#)

```
# set PDK_ROOT to the path you wish to use for the pdk
export PDK_ROOT=<pdk-installation-path>

# use volare to download the pdk
# To change the default pdk version you can export OPEN_PDKS_COMMIT=<pdk_commit>
make pdk-with-volare
```


RUNNING FULL CHIP SIMULATION

First, you will need to install the simulation environment, by

```
make simenv
```

This will pull a docker image with the needed tools installed.

Then, run the RTL simulation by

```
export PDK_ROOT=<pdk-installation-path>
# Run RTL simulation on IO ports testbench, make verify-io_ports
make verify-<testbench-name>-rtl
```

Once you have the physical implementation done and you have the gate-level netlists ready, it is crucial to run full gate-level simulations to make sure that your design works as intended after running the physical implementation.

Run the gate-level simulation by:

```
export PDK_ROOT=<pdk-installation-path>
# Run RTL simulation on IO ports testbench, make verify-io_ports
make verify-<testbench-name>-gl
```

This sample project comes with four example testbenches to test the IO port connection, wishbone interface, and logic analyzer. The test-benches are under the [verilog/dv](#) directory. For more information on setting up the simulation environment and the available testbenches for this sample project, refer to [README](#).

USER PROJECT WRAPPER REQUIREMENTS

Your hardened `user_project_wrapper` must match the `golden user_project_wrapper` in the following:

- Area (2.920um x 3.520um)
- Top module name "user_project_wrapper"
- Pin Placement
- Pin Sizes
- Core Rings Width and Offset
- PDN Vertical and Horizontal Straps Width

You are allowed to change the following if you need to:

- PDN Vertical and Horizontal Pitch & Offset

To make sure that you adhere to these requirements, we run an exclusive-or (XOR) check between your hardened `user_project_wrapper` GDS and the golden wrapper GDS after processing both layouts to include only the boundary (pins and core rings). This check is done as part of the `mpw-precheck` tool.

HARDENING THE USER PROJECT USING OPENLANE

12.1 OpenLane Installation

You will need to install openlane by running the following

```
export OPENLANE_ROOT=<openlane-installation-path>

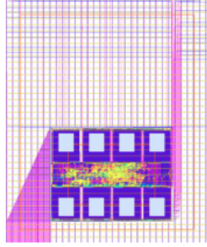
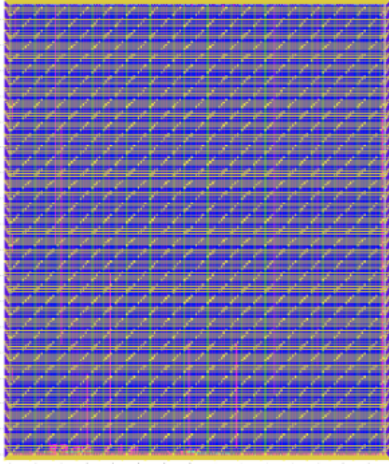
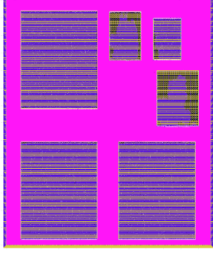
# you can optionally specify the openlane tag to use
# by running: export OPENLANE_TAG=<openlane-tag>
# if you do not set the tag, it defaults to the last verified tag tested for this project

make openlane
```

For detailed instructions on the openlane and the pdk installation refer to [README](#).

12.2 Hardening Options

There are three options for hardening the user project macro using openlane:

Option 1	Option 2	Option 3
Hardening the user macro(s) first, then inserting it in the user project wrapper with no standard cells on the top level	Flattening the user macro(s) with the user_project_wrapper	Placing multiple macros in the wrapper along with standard cells on the top level
		
ex: caravel_user_project		ex: caravel_ibex

For more details on hardening macros using openlane, refer to [README](#).

12.3 Running OpenLane

For this sample project, we went for the first option where the user macro is hardened first, then it is inserted in the user project wrapper without having any standard cells on the top level.

To reproduce hardening this project, run the following:

```
# DO NOT cd into openlane

# Run openlane to harden user_proj_example
make user_proj_example
# Run openlane to harden user_project_wrapper
make user_project_wrapper
```

For more information on the openlane flow, check [README](#).

RUNNING MPW PRECHECK LOCALLY

You can install the `mpw-precheck` by running

```
# By default, this install the precheck in your home directory
# To change the installtion path, run "export PRECHECK_ROOT=<precheck installation path>"
make precheck
```

This will clone the precheck repo and pull the latest precheck docker image.

Then, you can run the precheck by running

```
make run-precheck
```

This will run all the precheck checks on your project and will produce the logs under the `checks` directory.

OTHER MISCELLANEOUS TARGETS

The makefile provides a number of useful targets that can run LVS, DRC, and XOR checks on your hardened design outside of openlane's flow.

Run `make help` to display available targets.

Run lvs on the mag view,

```
make lvs-<macro_name>
```

Run lvs on the gds,

```
make lvs-gds-<macro_name>
```

Run lvs on the maglef,

```
make lvs-maglef-<macro_name>
```

Run drc using magic,

```
make drc-<macro_name>
```

Run antenna check using magic,

```
make antenna-<macro_name>
```

Run XOR check,

```
make xor-wrapper
```


CHECKLIST FOR OPEN-MPW SUBMISSION

- ✓ The project repo adheres to the same directory structure in this repo.
- ✓ The project repo contain info.yaml at the project root.
- ✓ Top level macro is named `user_project_wrapper`.
- ✓ Full Chip Simulation passes for RTL and GL (gate-level)
- ✓ The hardened Macros are LVS and DRC clean
- ✓ The project contains a gate-level netlist for `user_project_wrapper` at `verilog/gl/user_project_wrapper.v`
- ✓ The hardened `user_project_wrapper` adheres to the same pin order specified at [pin_order](#)
- ✓ The hardened `user_project_wrapper` adheres to the fixed wrapper configuration specified at [fixed_wrapper_cfgs](#)
- ✓ XOR check passes with zero total difference.
- ✓ Openlane summary reports are retained under `./signoff/`
- ✓ The design passes the [mpw-precheck](#)