
CIIC Harness

efabless

Nov 11, 2021

CONTENTS

1	Table of contents	3
2	Overview	5
3	Prerequisites	7
4	Install Caravel	9
5	Caravel Integration	11
5.1	Repo Integration	11
5.2	Verilog Integration	11
5.3	Layout Integration	12
6	Building the PDK	13
7	Running Full Chip Simulation	15
8	User Project Wrapper Requirements	17
9	Hardening the User Project using OpenLane	19
9.1	OpenLane Installation	19
9.2	Hardening Options	19
9.3	Running OpenLane	20
10	Running MPW Precheck Locally	21
11	Other Miscellaneous Targets	23
12	Checklist for Open-MPW Submission	25

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

INSTALL CARAVEL

To setup caravel, run the following:

```
# If unset, CARAVEL_ROOT will be set to $(pwd)/caravel
# If you want to install caravel at a different location, run "export CARAVEL_ROOT=
↳<caravel-path>"
export CARAVEL_ROOT=$(pwd)/caravel

# Disable submodule installation if needed by, run "export SUBMODULE=0"

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

To update the installed caravel to the latest, run:

```
make update_caravel
```

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

5.1 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`.

5.2 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.

5.3 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

You have two options for building the pdk:

- Build the pdk natively.

Make sure you have [Magic VLSI Layout Tool](#) installed on your machine before building the pdk. The pdk build is tested with magic version 8.3.209.

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

# you can optionally specify skywater-pdk and open-pdks commit used
# by setting and exporting SKYWATER_COMMIT and OPEN_PDKS_COMMIT
# if you do not set them, they default to the last verified commits tested for this_
↳project

make pdk
```

- Build the pdk using openlane's docker image which has magic installed.

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

# you can optionally specify skywater-pdk and open-pdks commit used
# by setting and exporting SKYWATER_COMMIT and OPEN_PDKS_COMMIT
# if you do not set them, they default to the last verified commits tested for this_
↳project

make pdk-nonnative
```


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>
export CARAVEL_ROOT=$(pwd)/caravel
# specify simulation mode: RTL/GL
export SIM=RTL
# Run RTL simulation on IO ports testbench, make verify-io_ports
make verify-<testbench-name>
```

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>
export CARAVEL_ROOT=$(pwd)/caravel
# specify simulation mode: RTL/GL
export SIM=GL
# Run RTL simulation on IO ports testbench, make verify-io_ports
make verify-<testbench-name>
```

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

These fixed configurations are specified [here](#) .

However, 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

9.1 OpenLane Installation

You will need to install openlane by running the following

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

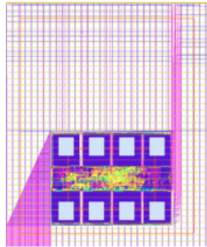
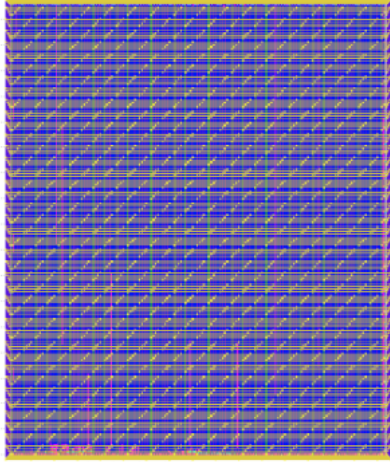
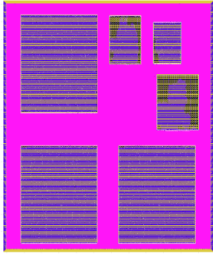
# you can optionally specify the openlane tag to use
# by running: export OPENLANE_TAG=<openlane-tag>
# if you do not set the tag, it defaults to the last verified tag tested for this project

make openlane
```

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

9.2 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](#).

9.3 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:

```
# 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 Specify `CARAVEL_ROOT` before running any of the following,

```
# export CARAVEL_ROOT=$(pwd)/caravel
export CARAVEL_ROOT=<path-to-caravel>
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.

Specify `CARAVEL_ROOT` before running any of the following,

```
# export CARAVEL_ROOT=$(pwd)/caravel  
export CARAVEL_ROOT=<path-to-caravel>
```

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`