CIIC Harness

efabless

Mar 26, 2023

CONTENTS

CHAPTER

ONE

TABLE OF CONTENTS

CHAPTER

TWO

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.

CHAPTER

THREE

PREREQUISITES

- Docker: Linux || Windows || Mac with Intel Chip || Mac with M1 Chip
- Python 3.6+ with PIP

CHAPTER

FOUR

QUICKSTART

4.1 Starting your project

1. To start the project you first need to create a new repository based on the caravel_user_project template and make sure your repo is public and includes a README.
   - Follow https://github.com/efabless/caravel_user_project/generate to create a new repository.
   - Clone the reposity using the following command:

   ```
   git clone <your github repo URL>
   ```

2. To setup your local environment run:

```
cd <project_name> # project_name is the name of your repo

mkdir dependencies

export OPENLANE_ROOT=$(pwd)/dependencies/openlane_src # you need to export this␣
↪whenever you start a new shell

export PDK_ROOT=$(pwd)/dependencies/pdks # you need to export this whenever you␣
↪start a new shell

# export the PDK variant depending on your shuttle, if you don't know leave it to␣
↪the default

# for sky130 MPW shuttles....
export PDK=sky130A

# for the gf180 GFMPW shuttles...
export PDK=gf180mcuC



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

1. Now you can start hardening your design
   - To start hardening you 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 *Hardening the User Project using OpenLane*.
     .
2. 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
     ```
3. 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

     # OR for GL+SDF simulation using
     # sdf annotated simulation is slow
     make verify-<testbench-name>-gl-sdf

     # for example
     make verify-io_ports-rtl
     ```
4. Run opensta on your design
   - Extract spefs for `user_project_wrapper` and macros inside it:

     ```
     make extract-parasitics
     ```
   - Create spef mapping file that maps instance names to spef files:

     ```
     make create-spef-mapping
     ```
   - Run opensta:

     ```
     make caravel-sta
     ```

     NOTE: To update timing scripts run `make setup-timing-scripts`

5. Run the precheck locally

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

6. You are done! now go to https://efabless.com/open_shuttle_program/ to submit your project!

CHAPTER

FIVE

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 GPIO Configuration

You are required to specify the power-on default configuration for each GPIO in Caravel. The default configuration provide the state the GPIO will come up on power up. The configuration can be changed by the management SoC during firmware execution.

Configuration settings define whether the GPIO is configured to connect to the user project area or the managment SoC. They also determine whether IOs are inputs or outputs, digital or analog, as well as whether pull-up or pull-down resistors are configured for inputs.

GPIOs are configured by assigning predefined values for each IO in the file verilog/rtl/user_defines.v in your project.

You need to assigned configuration values for GPIO[5] thru GPIO[37].

GPIO[0] thru GPIO[4] are preset and cannot be changed.

The following values are redefined for assigning to GPIOs.

- GPIO_MODE_MGMT_STD_INPUT_NOPULL
- GPIO_MODE_MGMT_STD_INPUT_PULLDOWN
- GPIO_MODE_MGMT_STD_INPUT_PULLUP
- GPIO_MODE_MGMT_STD_OUTPUT
- GPIO_MODE_MGMT_STD_BIDIRECTIONAL
- GPIO_MODE_MGMT_STD_ANALOG
- GPIO_MODE_USER_STD_INPUT_NOPULL
- GPIO_MODE_USER_STD_INPUT_PULLDOWN
- GPIO_MODE_USER_STD_INPUT_PULLUP
- GPIO_MODE_USER_STD_OUTPUT
- GPIO_MODE_USER_STD_BIDIRECTIONAL
- GPIO_MODE_USER_STD_OUT_MONITORED
- GPIO_MODE_USER_STD_ANALOG

MPW_Prececk includes a check to confirm each GPIO is assigned a valid value.

5.4 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* .

CHAPTER

SIX

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>
make verify-<testbench-name>-rtl

# For example
make verify-io_ports-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>
make verify-<testbench-name>-gl

# For example
make verify-io_ports-gl
```

To make sure that your design is timing clean, one way is running sdf annotated gate-level simulation Run the sdf annotated gate-level simulation by:

```
export PDK_ROOT=<pdk-installation-path>
make verify-<testbench-name>-gl-sdf

# For example
make verify-io_ports-gl-sdf
```

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.

CHAPTER

SEVEN

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.

CHAPTER

EIGHT

HARDENING THE USER PROJECT USING OPENLANE

8.1 OpenLane Installation

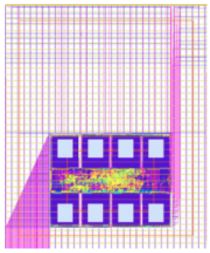
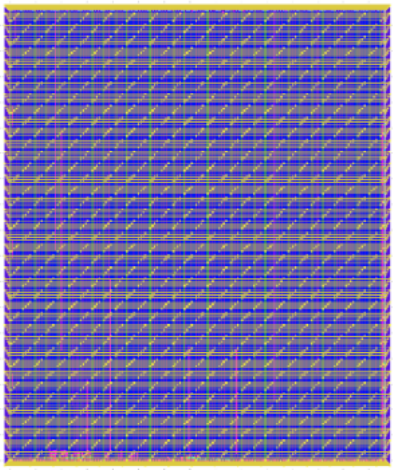
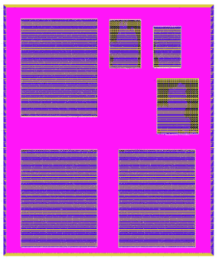
You will need to install openlane by running the following

```
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 verfied tag tested for this project

make openlane
```

For detailed instructions on the openlane and the pdk installation refer to README.

8.2 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.

8.3 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.

CHAPTER

NINE

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.

CHAPTER

TEN

RUNNING TIMING ANALYSIS ON EXISTING PROJECTS

Start by updating the Makefile for your project. Starting in the project root…

```
curl -k https://raw.githubusercontent.com/efabless/caravel_user_project/main/Makefile >␣
↪Makefile

make setup-timing-scripts

make install

make install_mcw
```

This will update Caravel design files and install the scripts for running timing.

Then, you can run then run timing by the following…

```
make extract-parasitics

make create-spef-mapping

make caravel-sta
```

A summary of timing results is provided at the end of the flow.

CHAPTER

ELEVEN

OTHER MISCELLANEOUS TARGETS

The makefile provides a number of useful that 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
```

CHAPTER

TWELVE

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