CIIC Harness

efabless

Nov 25, 2022

CONTENTS

CHAPTER

ONE

TABLE OF CONTENTS

CHAPTER

TWO

OVERVIEW

This repo contains a sample user project that utilizes the caravan chip (analog version of caravel) user space. The user project is a simple power-on-reset that showcases how to make use of caravan's user space utilities like IO pads, logic analyzer probes, and wishbone port. The repo also demonstrates the recommended structure for the open-mpw **analog** projects.

CHAPTER

THREE

INSTALL CARAVEL

To setup caravel, run the following:

```
# By default, CARAVEL_ROOT is set to $(pwd)/caravel
# If you want to install caravel at a different location, run "export CARAVEL_ROOT=
↪<caravel-path>"
# Disable submodule installation if needed by, run "export SUBMODULE=0"

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

CHAPTER

FOUR

CARAVEL INTEGRATION

4.1 User Project: Power on Reset

This is an example user analog project which breaks out the power-on-reset circuit used by the management SoC for power-up behavior so that the circuit input and output can be independently controlled and measured.

The power-on-reset circuit itself is a simple, non-temperature-compensated analog delay calibrated to 15ms under nominal conditions, with a Schmitt trigger inverter to provide hysteresis around the trigger point to provide a clean output reset signal.

The circuit provides a single high-voltage (3.3V domain) sense-inverted reset signal “porb_h” and complementary low-voltage (1.8V domain) reset signals “por_l” and “porb_l”.

The only input to the circuit is the 3.3V domain power supply itself.

4.2 Verilog Integration

You need to create a wrapper around your macro that adheres to the template at user_analog_project_wrapper. The wrapper top module must be named `user_analog_project_wrapper` and must have the same input and output ports as the analog 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.

The verilog modules instantiated in the wrapper module should represent the analog project; they need not be more than empty blocks, but it is encouraged to write a simple behavioral description of the analog circuit in standard verilog, using real-valued wires when necessary. This allows the whole system to be run in a verilog testbench and verify the connectivity to the padframe and management SoC, even if the testbench C code does nothing more than set the mode of each GPIO pin. The example top-level verilog code emulates the behavior of the power-on-reset delay after applying a valid power supply to the circuit.

CHAPTER

FIVE

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
```

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.

To install the simulation environment locally, refer to README

Then, run the RTL and GL simulation by

```
export PDK_ROOT=<pdk-installation-path>
export CARAVEL_ROOT=$(pwd)/caravel
# specify simulation mode: RTL/GL
export SIM=RTL
# Run the mprj_por testbench, make verify-mprj_por
make verify-<testbench-name>
```

The verilog test-benches are under this directory verilog/dv.

CHAPTER

SEVEN

ANALOG DESIGN FLOW

The example project uses a very simple analog design flow with schematics made with xschem, simulation done using ngspice, layout done with magic, and LVS verification done with netgen. Sources for the power-on-reset circuit are in the "xschem/" directory, which also includes a schematic representing the wrapper with all of its ports, for use in a testbench circuit. There are several testbenches in the example, starting from tests of the component devices to a full test of the completed project inside the wrapper.

There is no automation in this project; the schematic and layout were done by hand, including both the power-on-reset block and the power and signal routing to the pins on the wrapper.

The power-on-reset circuit itself is simple and is not compensated for temperature or voltage variation. When the power supply reaches a sufficient level, the voltage divider sets the gate voltage on an nFET device to draw a current of nominally 240nA. The testbench "threshold_test_tb.spice" does a DC sweep to find the gate voltage that produces this value. Next, a cascaded current mirror divides down the current by a factor of (roughly) 400. The testbench current_test.spice checks the current division value. Finally, the output ~600pA from the end of the current mirror is accumulated on a capacitor until the value trips the input of the 3.3V Schmitt trigger buffer from the sky130_fd_sd_hvl library. The capacitor is sized to peg the nominal time to trigger at 15ms. The schematic "example_por_tb.sch" sets up the testbench for this timing test.

The output of the Schmitt trigger buffer becomes the high-voltage output, and is input to a standard buffer and inverter used as level shifters from the 3.3V domain to the 1.8V domain, producing complementary low-voltage outputs.

The user project is formed from two power-on-reset circuits, one of which is connected to the user area VDDA1 power supply, and the other of which is connected to one of the analog I/O pads, used as a power supply input and connected to its voltage ESD clamp circuit. The 3.3V domain outputs are connected directly to GPIO pads through the ESD (150 ohm series) connection. The 1.8V domain outputs are connected to GPIO pads through the usual I/O connections, with the corresponding user output enable (sense inverted) held low to keep the output always active.

The C code testbench is in "verilog/dv/mprj_por/mprj_por.c" and only sets the GPIO pins used to the correct state (user output function). The POR circuit outputs are monitored by the testbench verilog file "mprj_por_tb.v" which will fail if the connections are wrong or if the behavioral POR verilog does not work as intended.

Note that to properly test this circuit, the GPIO pins have to be configured for output to be seen and measured, implying that the management SoC power supply must be stable and the C program running off of the SPI flash before the user area power supplies are raised.

NOTE

When running spice extraction on the user_analog_project_wrapper layout, it is recommended to use *ext2spice short resistor*. This is to preserve all the different port names in the extracted netlist. In case you have two ports that are electrically shorted in the layout, the *short resistor* option will tell magic not to merge the two shorted ports instead it adds zero-ohm ideal resistors between the net names so that they can be kept as separate nets.

CHAPTER

EIGHT

RUNNING OPEN-MPW PRECHECK LOCALLY

You can install the 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 retain the logs under the `checks` directory.

CHAPTER

NINE

OTHER MISCELLANEOUS TARGETS

The makefile provides a number of useful that targets that can run compress, uncompress, and run XOR checks on your design.

Compress gds files and any file larger than 100MB (GH file size limit),

```
make compress
```

Uncompress files,

```
make uncompress
```

Specify CARAVEL_ROOT before running any of the following,

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

Run XOR check,

```
make xor-analog-wrapper
```

CHAPTER

TEN

CHECKLIST FOR OPEN-MPW SUBMISSION

|:heavy_check_mark:| The project repo adheres to the same directory structure in this repo.

|:heavy_check_mark:| The project repo contain info.yaml at the project root.

|:heavy_check_mark:| Top level macro is named `user_analog_project_wrapper`.

|:heavy_check_mark:| Full Chip Simulation passes for RTL and GL (gate-level)

|:heavy_check_mark:| The project contains a spice netlist for the `user_analog_project_wrapper` at net-gen/user_analog_project_wrapper.spice

|:heavy_check_mark:| The hardened Macros are LVS and DRC clean

|:heavy_check_mark:| The `user_analog_project_wrapper` adheres to empty wrapper template order specified at user_analog_project_wrapper_empty

|:heavy_check_mark:| XOR check passes with zero total difference.

|:heavy_check_mark:| Open-MPW-Precheck tool runs successfully.