Caravel User Guide

Resources

Getting Started

Quickstart: https://caravel-user-project.readthedocs.io/en/latest/

OpenMPW: https://efabless.com/kb-articles/creating-your-first-open-mpw-or-chipignite-project

Simulation: https://caravel-user-project.readthedocs.io/en/latest/#running-full-chip-simulation

Tools

KLayout: https://www.klayout.de/build.html

Docker Desktop: https://www.docker.com/products/docker-desktop/

GTKWave: https://sourceforge.net/projects/gtkwave/files/gtkwave-3.3.100-bin-win64/

Magic: http://opencircuitdesign.com/magic/

Reference

User Project: https://caravel-user-project.readthedocs.io/en/latest/

Harness: https://caravel-harness.readthedocs.io/en/latest/

MGMT SoC: https://caravel-mgmt-soc-litex.readthedocs.io/en/latest/

Wishbone Bus: https://cdn.opencores.org/downloads/wbspec_b4.pdf

PDK DRC Rules: https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html

OpenRAM: https://vlsi.jp/OpenMPWSRAM_eng.html#using-sram-with-openmpw

Bring-up: https://github.com/efabless/caravel_board/tree/main

Project Setup

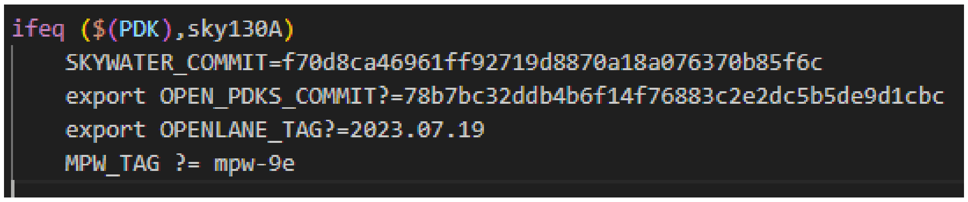
Clone the required caravel_user_project repo from caravel github:
https://github.com/efabless/caravel_user_project

Before running any simulations or hardening, the following three commands must be run in the root of the caravel repository:

```
export OPENLANE_ROOT=$(pwd)/dependencies/openlane_src

export PDK_ROOT=$(pwd)/dependencies/pdks

export PDK=sky130A
```

NOTE: This must be done EVERY TIME anything is run, NOT just on setup for the export commands.

After the root paths are set, we can now build our project. Please ensure that the git commit tags in the root Makefile (seen below) are UP TO DATE with the most recent MPW shuttle branch for the required project submission.



```
ifeq ($(PDK),sky130A)
    SKYWATER_COMMIT=f70d8ca46961ff92719d8870a18a076370b85f6c
    export OPEN_PDKS_COMMIT?=78b7bc32ddb4b6f14f76883c2e2dc5b5de9d1cbc
    export OPENLANE_TAG?=2023.07.19
    MPW_TAG ?= mpw-9e
```

Skywater130A MPW9 Git Commit Tags

Run ‘make setup’ to install the following:

- Skywater pdk (make pdk-with-volare)
- Openlane (make openlane)
- Submission precheck (make precheck)

While other dependencies and checks are made in make setup, all the required builds will occur here, ensuring you DO NOT need to run any other make command to build the project. If there is an issue with your project build, it is encouraged to delete the dependencies folder and rerun make setup. If that does not work, delete the cloned repsository and start fresh with another ‘make setup’, ensuring the most up to date commit tags are given in the Makefile, as pulled from the user_project repo.

RTL Design

A functional RTL design can be written in Verilog and placed under the /verilog/ folder in the caravel repository. Inside of this folder, the following actions should be made to set up a new RTL design:

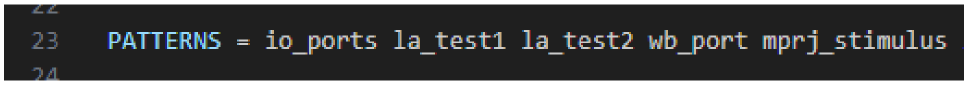
1. /verilog/rtl
    a. Add the new Verilog designs, can also be in own folder path under /verilog/rtl
2. /verilog/includes/includes.gl.caravel_user_project
    a. Add "-v $(USER_PROJECT_VERILOG)/gl/design/MODULE.v"
3. /verilog/includes/includes.gl+sdf.caravel_user_project
    a. Add "$USER_PROJECT_VERILOG/gl/design/MODULE.v"
4. /verilog/includes/includes.rtl.caravel_user_project
    a. Add "-v $(USER_PROJECT_VERILOG)/rtl/design/MODULE.v"

NOTE: For each of the verilog files under /verilog/includes, there MUST be an empty line after the last entry, or the make files for the caravel repository will break.

RTL Simulations

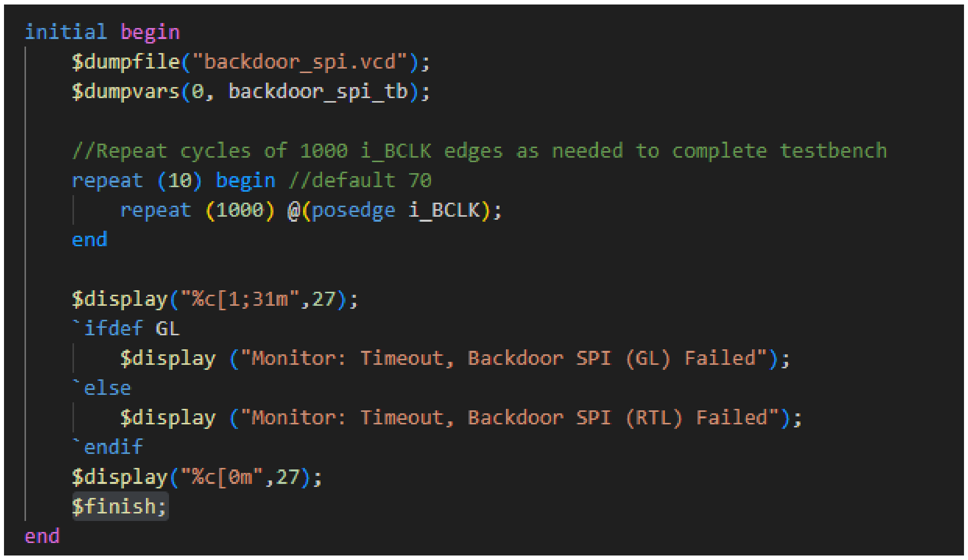
To setup a Verilog testbench in the caravel repository, follow the steps below:

1. /verilog/dv/Makefile
    a. Add MODULE name to PATTERNS list in makefile
2. /verilog/dv
    a. Copy an existing testcase folder
    b. Rename the new folder to MODULE
3. /verilog/dv/MODULE
    a. Rename the copied testbench verilog file and c file to MODULE
    b. Delete all C code from Module.c if only running RTL simulation
4. /verilog/dv/MODULE/MODULE_tb.v
    a. Set name for .vcd file by editing $dumpfile("MODULE.vcd");
    b. Set name for dumpvars by editing $dumpvars(0, MODULE_tb)
    c. Add cmd displays with $display("STRING") in the testbench if desired



```
PATTERNS = io_ports la_test1 la_test2 wb_port mprj_stimulus
```

/verilog/dv/Makefile PATTERNS



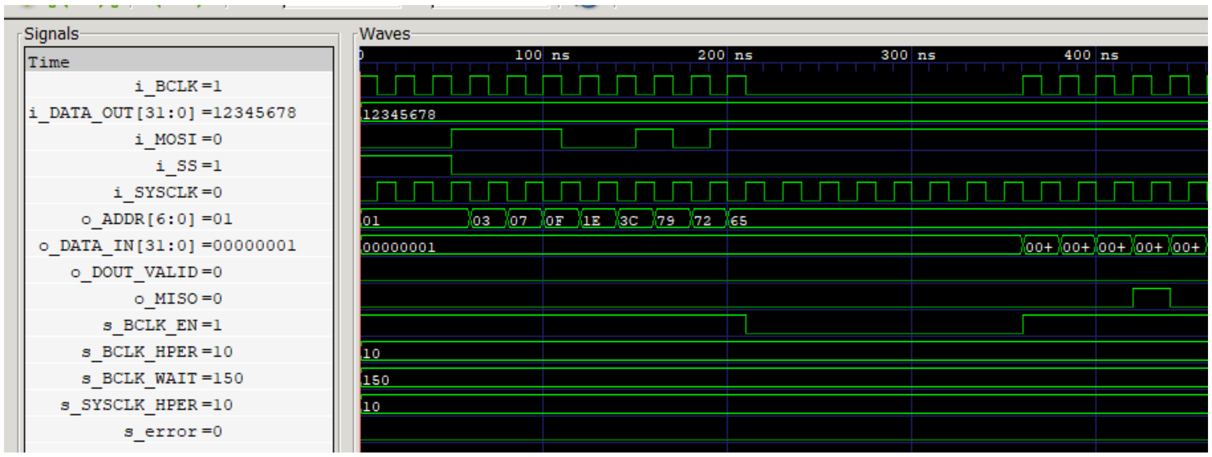
```
initial begin
    $dumpfile("backdoor_spi.vcd");
    $dumpvars(0, backdoor_spi_tb);

    //Repeat cycles of 1000 i_BCLK edges as needed to complete testbench
    repeat (10) begin //default 70
        repeat (1000) @(posedge i_BCLK);
    end

    $display("%c[1;31m",27);
    `ifdef GL
        $display ("Monitor: Timeout, Backdoor SPI (GL) Failed");
    `else
        $display ("Monitor: Timeout, Backdoor SPI (RTL) Failed");
    `endif
    $display("%c[0m",27);
    $finish;
end
```

/verilog/dv/MODULE/MODULE_tb.v dumpfile, dumpvars, display examples

To run a RTL level simulation from root: "make verify-<module>-rtl"

After the RTL simulation is run, you can view the waveform results using the Open-Source tool GTKWave. After opening GTKWave, select File, Open New Tab, and then navigate to the

/verilog/dv/MODULE path. Inside of your module's dv folder, a .vcd file will have been generated, as specified with the $dumpfile command and name in your testbench. With the vcd file opened, signals can now be appended to be viewed.



RTL Simulation GTKWave Sample

Module Hardening

Functional modules are hardened under the /openlane/ folder in the Caravel repository. The supplied top level design user_project_wrapper comes with a set of requirements to pass the eFabless precheck and fabrication and should not be edited. The only Verilog files to add to the top-level module are the top-level functional Verilog files or hardened macros, as referenced below.

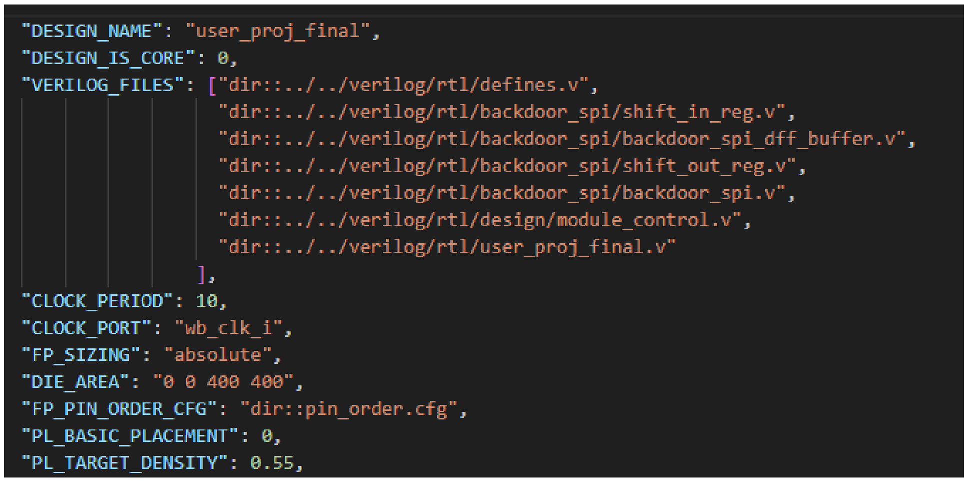
It is encouraged to harden your own macros individually, to verify both that they can harden on their own and pass through synthesis, but also in case you want to pass in your hardened design as a drop in macro, which can reduce the amount of resynthesizing of the top-level wrapper.

To setup a new module for hardening, follow the steps below:

1. /openlane/
   a. Copy an existing sample hardening configuration under /openlane/ that is NOT user_proj_wrapper
   b. Rename the copied config folder to the same MODULE name as your rtl and dv design
2. /openlane/MODULE
   a. Edit config.json with Desired parameters below
   b. Delete pin_order.cfg if not specifying north/east/south/west side for pin placement
   c. Delete macro.cfg if not placing pre-hardened macro into hardened design

The following parameters inside config.json SHOULD be changed, IF NOT user_project_wrapper:

1. **DESIGN_NAME**: name of functional verilog module
2. **VERILOG_FILES**: list of related Verilog files, including submodules instantiated in hardened design
3. **CLOCK_PORT**: Clock input to module for timing analysis
4. **CLOCK_NET**: Clock Net to module for timing analysis
5. **FP_SIZING**: Determines how length/width of module is determined
   a. Set to "Absolute" for a fixed sized based on DIE_AREA
   b. Set to "Relative" for optimized size. NOTE: Does not work for designs small enough if you have too many I/o ports
   c. Default is "Relative"
6. **DIE_AREA**: Determines length and width of hardened module if FP_SIZING set to Absolute
   a. Set to "x0 y0 x1 y1", or "0 0 1000 1000" for area corners
   b. Unit is µm
7. **PL_TARGET_DENSITY**: percentage from 0 to 1 of how DENSE cells are in area
   a. Most designs harden up to around 0.6
   b. 1 = closely dense, 0 = widely spread

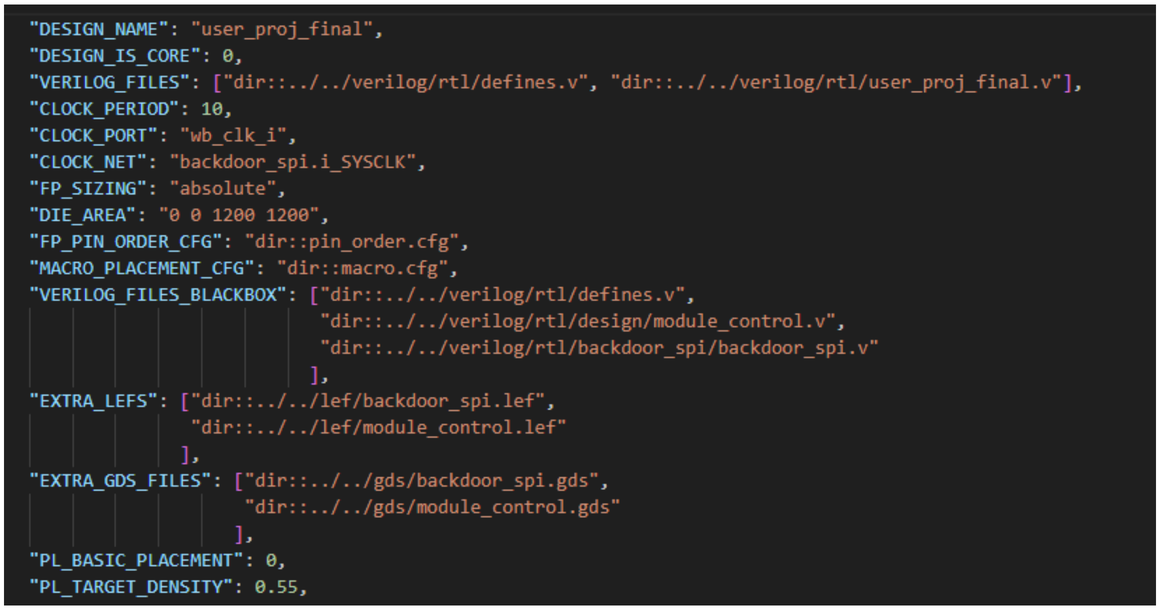


```
"DESIGN_NAME": "user_proj_final",
"DESIGN_IS_CORE": 0,
"VERILOG_FILES": ["dir::../../verilog/rtl/defines.v",
                  "dir::../../verilog/rtl/backdoor_spi/shift_in_reg.v",
                  "dir::../../verilog/rtl/backdoor_spi/backdoor_spi_dff_buffer.v",
                  "dir::../../verilog/rtl/backdoor_spi/shift_out_reg.v",
                  "dir::../../verilog/rtl/backdoor_spi/backdoor_spi.v",
                  "dir::../../verilog/rtl/design/module_control.v",
                  "dir::../../verilog/rtl/user_proj_final.v"
                 ],
"CLOCK_PERIOD": 10,
"CLOCK_PORT": "wb_clk_i",
"FP_SIZING": "absolute",
"DIE_AREA": "0 0 400 400",
"FP_PIN_ORDER_CFG": "dir::pin_order.cfg",
"PL_BASIC_PLACEMENT": 0,
"PL_TARGET_DENSITY": 0.55,
```

Sample config.json WITHOUT pre-hardened macros

If you are using pre-hardened modules, you should also edit the following config lines:

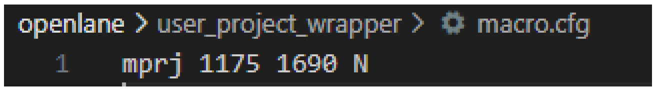
1. **VERILOG_FILES_BLACKBOX**: List each pre-hardened verilog design
2. **EXTRA_LEFS**: List each pre-hardened LEF file /lef/
3. **EXTRA_LIBS**: List each pre-hardened GDS file under /lib/
4. **EXTRA_GDS_FILES**: List each pre-hardened GDS file under /gds/
5. **MACRO_PLACEMENT_CFG**: Set to "dir::macro.cfg" for macro placement



```
"DESIGN_NAME": "user_proj_final",
"DESIGN_IS_CORE": 0,
"VERILOG_FILES": ["dir::../../verilog/rtl/defines.v", "dir::../../verilog/rtl/user_proj_final.v"],
"CLOCK_PERIOD": 10,
"CLOCK_PORT": "wb_clk_i",
"CLOCK_NET": "backdoor_spi.i_SYSCLK",
"FP_SIZING": "absolute",
"DIE_AREA": "0 0 1200 1200",
"FP_PIN_ORDER_CFG": "dir::pin_order.cfg",
"MACRO_PLACEMENT_CFG": "dir::macro.cfg",
"VERILOG_FILES_BLACKBOX": ["dir::../../verilog/rtl/defines.v",
                           "dir::../../verilog/rtl/design/module_control.v",
                           "dir::../../verilog/rtl/backdoor_spi/backdoor_spi.v"
                          ],
"EXTRA_LEFS": ["dir::../../lef/backdoor_spi.lef",
               "dir::../../lef/module_control.lef"
              ],
"EXTRA_GDS_FILES": ["dir::../../gds/backdoor_spi.gds",
                    "dir::../../gds/module_control.gds"
                   ],
"PL_BASIC_PLACEMENT": 0,
"PL_TARGET_DENSITY": 0.55,
```

Sample config.json WITH pre-hardened macros

NOTE: If you are using pre-hardened macros, you MUST specify macro placement in a macro.cfg config file under the folder for your openlane hardening config.

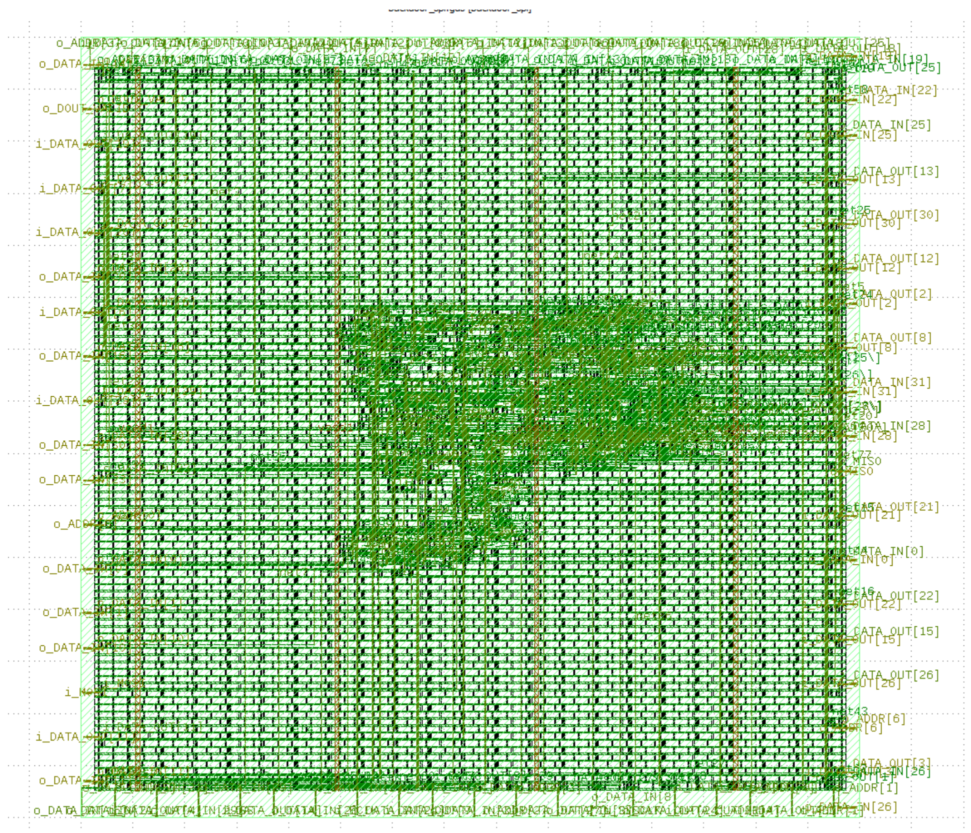


```
openlane > user_project_wrapper > macro.cfg
mprj 1175 1690 N
```

Sample /openlane/user_project_wrapper/macro.cfg

Useful Links: https://github.com/The-OpenROAD-Project/OpenLane/blob/master/docs/source/reference/configuration.md

The generated GDS files can be viewed in KLayout. After opening KLayout, select File, Open, and then navigate to the caravel/gds/MODULE.gds to open the hardened macro.



Sample GDS View in KLayout

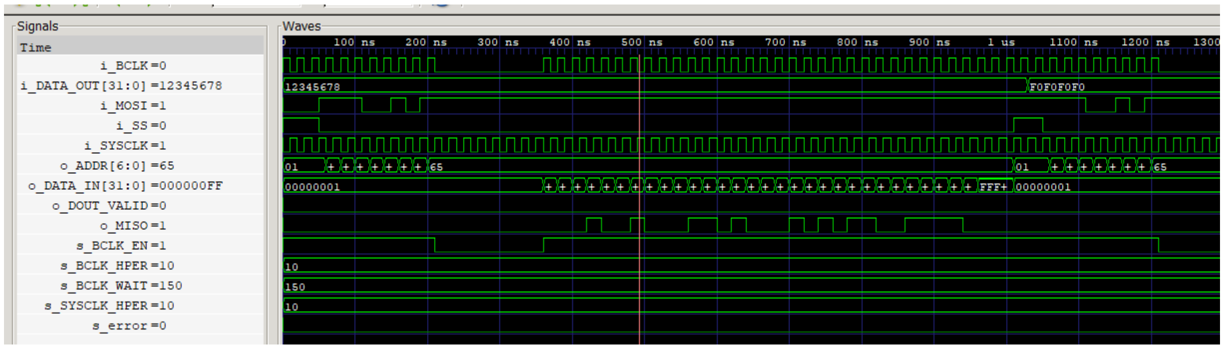
Gate Level Simulations

Gate Level simulations are very easy to run if your design has followed the above steps for RTL implementation, RTL simulation, and Module Hardening. At this point, the only thing left to do is run the make verify command with the gl tag, as seen below.

NOTE: To run gate level simulations, your design must be successfully hardened. This is because a netlist is generated based on the SkyWater PDK Standard Cell Library, where each of the functional logic gates are pulled from.

To run a GL level simulation from root: "make verify-<module>-gl"

GTKWave can be used to verify your gate level design also, which can be used with the vcd file GL-MODULE.vcd.



GL Simulation GTKWave Sample

Custom Cell Layout – Magic

Magic

Rationale: Instead of trying to draw the custom cell directly in magic which seemed very imprecise drawing each rectangle by hand, I opted to write a tcl script which would draw rectangles at specific coordinates. In retrospect, the .mag format is simple enough, I could probably have just written the .mag file by hand. However, then the tcl script got too unwieldy, so I created a python script to generate the .tcl file so I could use variables to define the box coordinates easier. This will probably make someone used to TCL cringe, but it worked. Magic must also be built from source, APT version is too old (https://github.com/RTimothyEdwards/magic)

Resources

https://isn.ucsd.edu/courses/beng207/lectures/Tim_Edwards_2021_slides.pdf

https://skywater-pdk.readthedocs.io/en/main/rules/periphery.html#npc

https://skywater-pdk.readthedocs.io/en/main/rules/layers.html

Generation flow

1. Run `make setup` to fetch the pdk
2. Copy dependencies/pdks/sky130A/libs.tech/magic/sky130A.magicrc to `.magicrc` in the cell generation working directory (`cell_gen`).
3. Run the python script to generate the .tcl file: `python3 NAND.py`
4. Open Magic by `cd`ing into cell_gen and running `magic`. Ensure the PDK environment variables are set.
5. Run the tcl script: `source NAND.tcl`. Make sure you're completely zoomed out when you run that, sometimes if you have zoomed into a certain portion of the design, only that part of the design actually gets erased correctly.
6. Update the JSON file in openlane (see below), and create a blackbox verilog file (see Creating the Blackbox below)
7. Run the flow (`make user_proj_final`). The GDS and LEF files were automatically inserted into openlane, and the .mag file has been saved for future reference.

LEF Class

There are two main LEF classes, BLOCK and CELL. The custom cell was designated as a CELL, so it will be placed in a row automatically by the detailed placer as expected. BLOCK is used for hardened macros (SRAM, user_proj_final, etc) to designate it as a block that should not be placed in a row. There are size constraints on a CELL that are not present in a BLOCK (must fit in an existing row), but BLOCKS cannot be packed as densely.

Creating the Blackbox

A blackbox verilog file is required to represent the custom cell. The inputs do not all have to be used, but it must have all four power pins (and the LEF/GDS should have all four power pins as well). These pin names must match exactly the names in the LEF/GDS file. A template from our NAND gate is below. One very important feature is the `/// sta-blackbox` which identifies this as a black box for static timing analysis and LVS check. List this in the openlane JSON file with the gds/lef files (VERILOG_FILES_BLACKBOX, EXTRA_LEFS, and EXTRA_GDS_FILES).

```
`default_nettype none

/// sta-blackbox

`celldefine
module NAND (
   output X,
   input A,
   input B,
   `ifdef USE_POWER_PINS
      inout VPWR,
      inout VGND,
      inout VPB,
      inout VNB
   `endif
);
   assign X = ~(A & B);

endmodule
`endcelldefine

`default_nettype wire
```

Tech Issues

The provided SKY130A tech file has an issue that makes all custom cells fail DRC. By DRC rules, two diff layers must have a gap of 27µm. NSDM/PSDM layers must have either no gap or a gap larger than some amount. In the standard cells, the NSDM/PSDM layers are 13.5µm from the end of the DIFF layers, but in the current SKY130A tech file, the NSDM/PSDM layers are auto-generated as a bounding box with a border of 12.5µm. Therefore, if you put the DIFF layers close enough for the PSDM layers to have no gap, the DIFF layers are too close, but if you put the DIFF layers further away, there is a gap in the NSDM layers. To fix this, edit the sky130A tech file in magic to change the 125's to 135's. Because this is re-generated after make setup, you will have to change this whenever using magic after make setup has been run.

templayer basePSDM pdiffres,mvpdiffres grow 15 or xhrpoly,uhrpoly,xpc grow 110 bloat-or allpactivetap * 125 allnactivenontap 0	templayer basePSDM pdiffres,mvpdiffres grow 25 or xhrpoly,uhrpoly,xpc grow 110 bloat-or allpactivetap * 135 allnactivenontap 0

bloat-or allpactivenontap * 125 allnactivetap 0 templayer baseNSDM ndiffres,mvndiffres grow 125 bloat-or allnactivetap * 125 allpactivenontap 0 bloat-or allnactivenontap * 125 allpactivetap 0	bloat-or allpactivenontap * 135 allnactivetap 0 templayer baseNSDM ndiffres,mvndiffres grow 135 bloat-or allnactivetap * 135 allpactivenontap 0 bloat-or allnactivenontap * 135 allpactivetap 0

Custom Cell Layout – XSchem

XSchem

XSchem is a schematic capture tool which will use the base models in the sky130PDK to allow you to simulate a custom cell from basic building blocks. To use XSchem, install from https://xschem.sourceforge.io/stefan/index.html, determine a working directory, and copy the dependencies/pdks/sky130A/libs.tech/xschem/xschemrc file into `.xschemrc`. Ensure the PDK_ROOT environment variable is set, then run `xschem` from the folder with `.xschemrc`. This will load the sky130A PDK into XSchem.

Useful Keyboard Shortcuts

U = Undo

Shift + U = Redo

Shift + I = Insert

C = Copy

M = Move

W = Wire

Shift + W = Snap Wire

Resources on Generating Simulations

https://xschem.sourceforge.io/stefan/xschem_man/graphs.html

https://xschem.sourceforge.io/stefan/xschem_man/tutorial_run_simulation.html

NGSpice Manual

https://ngspice.sourceforge.io/docs/ngspice-41-manual.pdf

SKY130 Inverter Reference

http://web02.gonzaga.edu/faculty/talarico/vlsi/xschemTut.html

Running a simulation

Generate a netlist by clicking the Netlist button in the upper right corner, or press `n`.

Run the simulation by clicking the Simulate button in the upper right corner, or run ngspice manually.

Load the waveform by holding ctrl and clicking the Load Waves button to load the waves from the .raw file produced by the simulation.

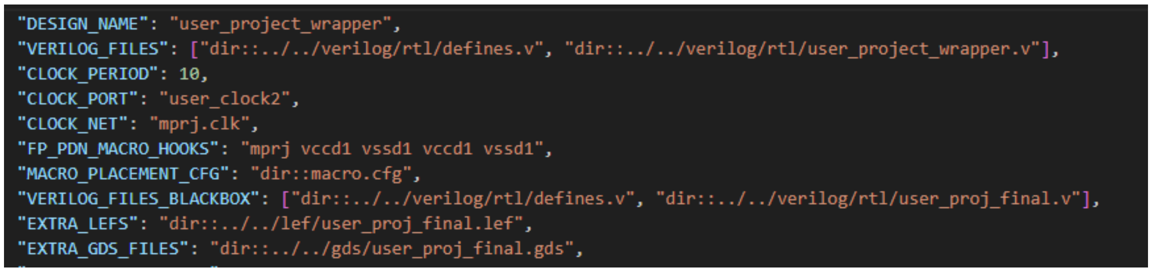
Double-click the graph body to change the nets shown. Digital mode will stack the graphs as separate waveforms instead of overlapping in space.

Top Level Wrapper Design

Once all the individual functional designs are complete, you will need to instantiate your design inside of the user_project_wrapper Verilog file. It is recommended that you create a wrapper module to be instantiated inside of the user_project_wrapper, so that you can harden this design WITHOUT the set requirements of the user_project_wrapper, which is used as the top level design for the submission and precheck.

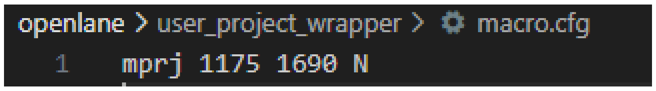
The user_project_wrapper has its own specific set of hardening configurations set up under the path /openlane/user_project_wrapper. Changing this should be taken with care, since this is used to generate the submission precheck and is used as the top level result for your fabricated design. What you SHOULD edit is the following:

1. /openlane/user_project_wrapper/config.json
    a. If NOT using pre-hardened macro:
        i. VERILOG_FILES: include top level module design in wrapper
    b. If using pre-hardened macro:
        i. VERILOG_FILES_BLACKBOX: include top level module design in wrapper
        ii. EXTRA_LEFS: include top level module design in wrapper
        iii. EXTRA_GDS_FILES: include top level module design in wrapper
2. /openlane/user_project_wrapper/macro.cfg
    a. If using pre-hardened macro:
        i. Specify center location of prehardened macro



```
"DESIGN_NAME": "user_project_wrapper",
"VERILOG_FILES": ["dir::../../verilog/rtl/defines.v", "dir::../../verilog/rtl/user_project_wrapper.v"],
"CLOCK_PERIOD": 10,
"CLOCK_PORT": "user_clock2",
"CLOCK_NET": "mprj.clk",
"FP_PDN_MACRO_HOOKS": "mprj vccd1 vssd1 vccd1 vssd1",
"MACRO_PLACEMENT_CFG": "dir::macro.cfg",
"VERILOG_FILES_BLACKBOX": ["dir::../../verilog/rtl/defines.v", "dir::../../verilog/rtl/user_proj_final.v"],
"EXTRA_LEFS": "dir::../../lef/user_proj_final.lef",
"EXTRA_GDS_FILES": "dir::../../gds/user_proj_final.gds",
```

Sample /openlane/user_project_wrapper/config.json



```
mprj 1175 1690 N
```

Sample /openlane/user_project_wrapper/macro.cfg

SRAM Usage and Integration

eFabless SRAM

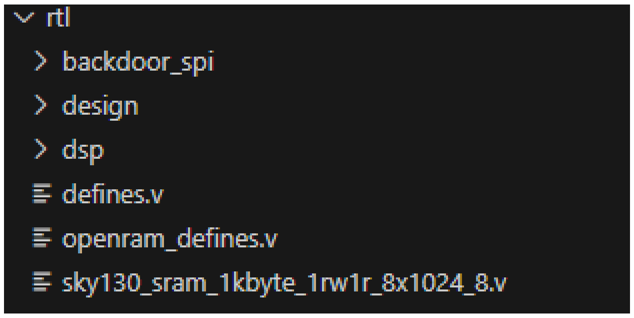
Utilizing onboard SRAM in the user area can have massive benefits over the DRAM integrated with the management microcontroller. The provider for the memory designs used in the eFabless OpenPDK is another open-source project called OpenRAM. This open-source project is a memory compiler that is capable of building memory macros that you can utilize in your designs. The sddec23-06 did not do a lot of work with the OpenRAM project other than using their precompiled macros. There is a great deal of customization that is available in the project and is something worth looking at in the future.

Sourcing the memory design

Now that you have decided to use SRAM in your design the first choice that you must make is where to source your prehardened SRAM macro. There are three primary locations where you can get these designs from. The first is to use the actual OpenRAM tool to generate your own custom design. This method will give you the most flexibility as it will allow you to directly create what you need for your specific design. The only drawback to this method is that it is another tool that you must learn to utilize the memory. The second location is in the OpenPDK that is downloaded when you setup your project. The PDK has four simple designs that you can utilize in your project. This is the most convenient method of getting memory into your project as it is included in the PDK already. However, the one downside to this method is that your memory selection is limited along with bugs existing in PDK version; some of the memories have been bugged for a while and it does not appear to be a priority for eFabless to push out the fixed designs by the OpenRAM team. The third and final way to source your memory designs is to use the designs in the second OpenRAM test chip https://github.com/VLSIDA/openram_testchip2. This was a test chip designed by the creators of OpenRAM and implements eleven different memory designs. This was the best method that I found for sourcing memory designs as it provided a wide range of designs to choose from. These designs also have the benefit of being designed and verified by the creators of the open-source project; the designers also manually went through the designs by hand to ensure that they would pass future drc and other hardening checks.

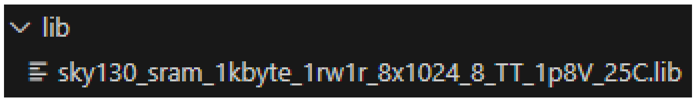
Bringing the files into your design

Once you have selected your design the next step is to bring all the needed files int your project. The four files that you need are the .gds .lef. .lib and .v files. They should all be place in the corresponding folders with the Verilog file placed in your rtl folder.

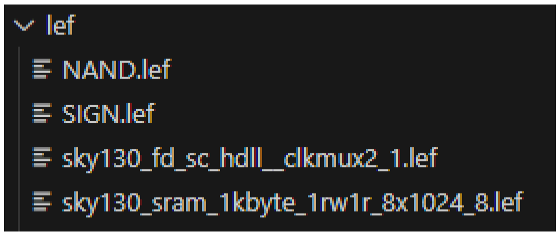


```
rtl
  backdoor_spi
  design
  dsp
  defines.v
  openram_defines.v
  sky130_sram_1kbyte_1rw1r_8x1024_8.v
```

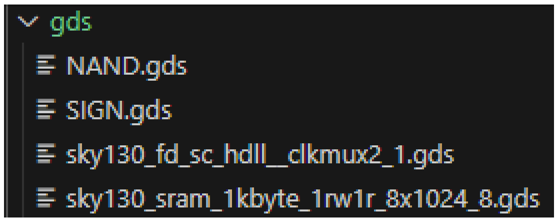
RTL location


```
lib
  sky130_sram_1kbyte_1rw1r_8x1024_8_TT_1p8V_25C.lib
```

Lib location


```
lef
  NAND.lef
  SIGN.lef
  sky130_fd_sc_hdll__clkmux2_1.lef
  sky130_sram_1kbyte_1rw1r_8x1024_8.lef
```

Lef location


```
gds
  NAND.gds
  SIGN.gds
  sky130_fd_sc_hdll__clkmux2_1.gds
  sky130_sram_1kbyte_1rw1r_8x1024_8.gds
```

Gds location

With all these folders added to your design you are ready to start integrating the memory into your rtl design.

RTL design

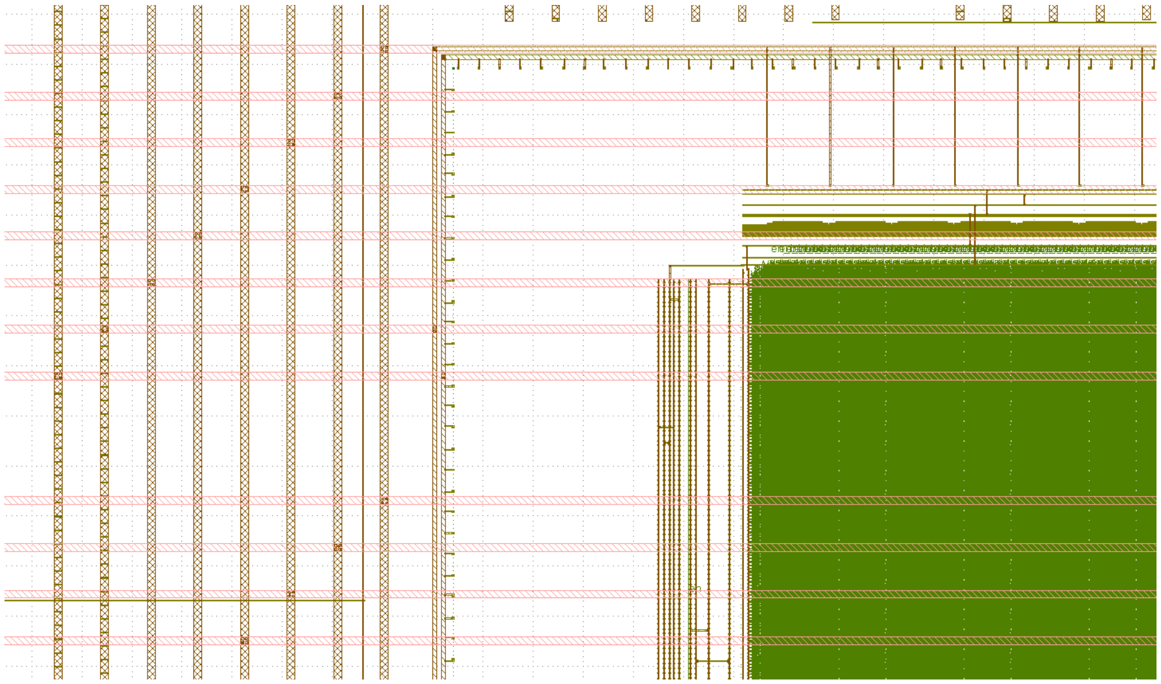
Implementing and using the memory is straight forward using the functional model provided in the design Verilog file. The first thing you must do though while designing with the memory is to place it in the user_project_wrapper and to interact with it there. The reasoning behind this will be explained in the hardening section as that is the time where the SRAM location matters. With the memory placed in the correct location you can start to hook up ports to the design. The first two that need to be connected are the power and ground connections called vccd1 and vssd1. These are important connections and need to be connected to either the vccd1 and vssd1 or vccd2 and vssd2 provided by the wrapper. The next ports to hook up are the clock ports to your desired clock signal. From there the next port to hook up is the chip select line to enable the memory. The next port to connect is the write enable port that will control whether the memory is reading from or writing to memory. From there we can hook up the address port to your address driver. The last two ports to hook up are the data in and data out ports that provide your data connection to the memory.

One niche thing about the memory is that it will have a three-cycle delay after the address is read in before the data is stored in the memory or able to be read from the memory. You must keep this in mind while designing with the memory. With this caveat in mind and having the memory ports hooked up you can implement it in your design however you see fit.

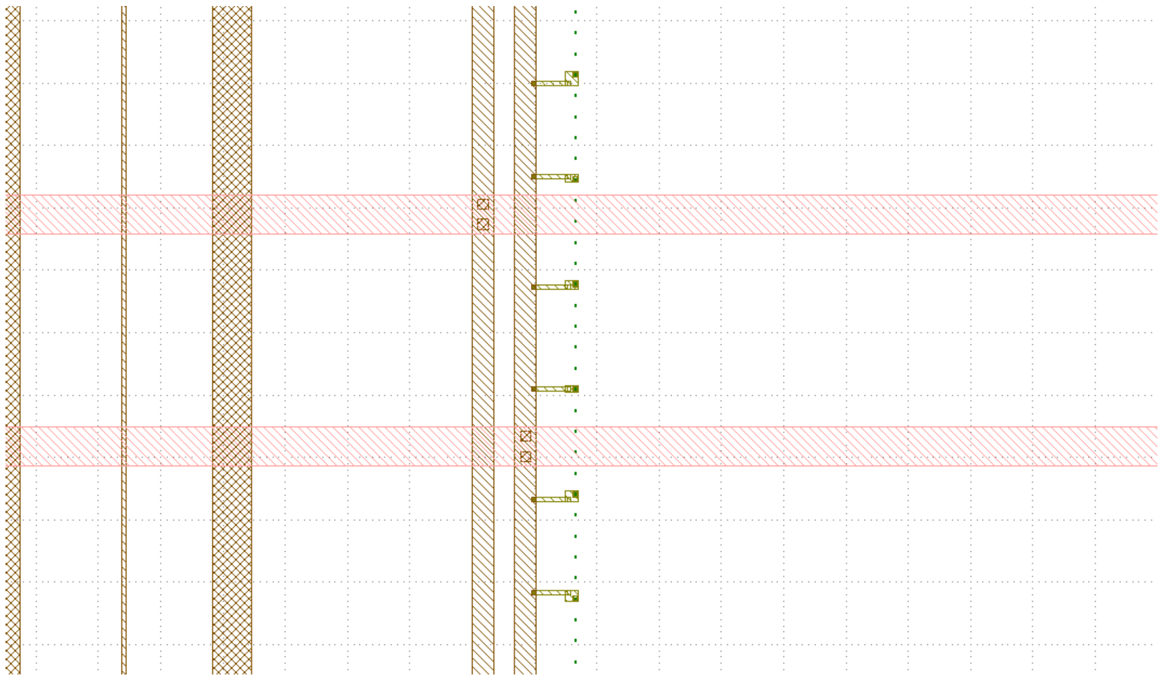
Hardening the SRAM

Once you have completed your rtl design with memory the next step is to go through the hardening process. The first step in this process is to modify / replace the config.json file for the user_project_wrapper hardening process. I would suggest that you use the config.json file from either sddec23-06's repo or from the second OpenRAM test chip https://github.com/VLSIDA/openram_testchip2. The reason for this is because there are multiple settings that need to be configured so that the memory will properly harden. During our initial attempts we were struggling to get the memory to properly hook up to the power rails. This issue was caused by the fact that we were instantiating the memory inside of another macro. This was problematic as the SRAM needs access to the top metal layer met5 for its power connections. When the macro was connected to the met5 layer it was able to be placed into the user_project_wrapper.

The main items that need to be set in the file are FP_PDN_CHECK_NODES, FP_PDN_ENABLE_RAILS, RUN_FILL_INSERTION, and RUN_TAP_DECAP_INSERTION. With all of these sets the memory should be able to properly harden and you should be able to see in the created gds file that the power rings of the memory module are connected to top met5 power layer.



Power Rail Connection



Zoomed In Power Rail Connection.

EFabless Submission Precheck

The MPW Precheck is a SEPERATE Github repository that can be cloned using the root makefile in the caravel repository. This is used to compare the hardened result of user_project_wrapper, our top-level module, against multiple different requirements for project submission. As long as the design fits all functional requirements, AND passes this precheck, a design is ready for submission.

To clone the MPW precheck github repository: ‘make precheck’

To run the MPW Precheck: ‘make run-precheck'

NOTE: The top-level wrapper user_project_wrapper must be hardened with ‘make user_project_wrapper’ before a precheck can occur.

Modifying project to pass precheck parameters:

1. Documentation Pass
   a. The root README.md file must be modified to ensure documentation has been updated
   b. For a minimum pass, delete all existing contents and add custom header title
2. GPIO Pass
   a. Update GPIO INIT config from INVALID for following file:
      i. sddec23-06/verilog/rtl/user_defines.v
   b. Can set to either inputs, outputs, bidirectional, or analog pins

Troubleshooting Notes

LINTER: Mixing positional and .*/named instantiation connection

- `ifdef USE_POWER_PINS must be the first items in the module port list, for some reason it gets angry if you put the power pins last.

Supported Verilog (* attributes *)

- https://github.com/The-OpenROAD-Project/yosys/blob/master/README.md#verilog-attributes-and-non-standard-features

What is maglef vs lef?

- Best guess is according to https://github.com/The-OpenROAD-Project/OpenLane/issues/1067, some of the older PDK's had a set of LEF issues, maglef is LEF files re-exported using Magic to hopefully fix most of those issues.