EIT, Lund University

IC Project 1, ETIN35

RISC-V SoC for ML: From RTL to GDSII

PULPino

Author(s)
Kristoffer Westring
Kristoffer.westring@eit.lth.se

Lund University

Revision History

Revision	Date	Author(s)	Description
1.0	March 1, 2024	KW	Created
1.01	March 5, 2024	SCM, WM	Revised
1.1	March 25, 2024	KW	Revised, added 2.3, 2.2.1. Modified 2.2. Added Appendix.
1.11	April 4, 2024	KW	Removed Work-In-Progress.
1.12	Feb 25, 2025	KW	Project name changed, new TAs added.
1.2	Mar 6, 2025	KW	Clarified tasks, added new tasks.

Contents

1 Information

The PULPino platform is an open-source microcontroller unit (MCU) built around the RISC-V instruction set architecture (ISA), developed at ETH, Zurich. The PULPino platform contains a single RISC-V core that suppports integer instructions (RV32I), multiplication/division instructions (RV32M), and compressed instructions (RV32C).

The PULP platform is provided with a number of peripherals, Fig. 1, of which some can be used to communicate with the outside world. Two of these peripherals, `SPISlave` and `Adv. DebugUnit`, can be used to program and debug the SoC.

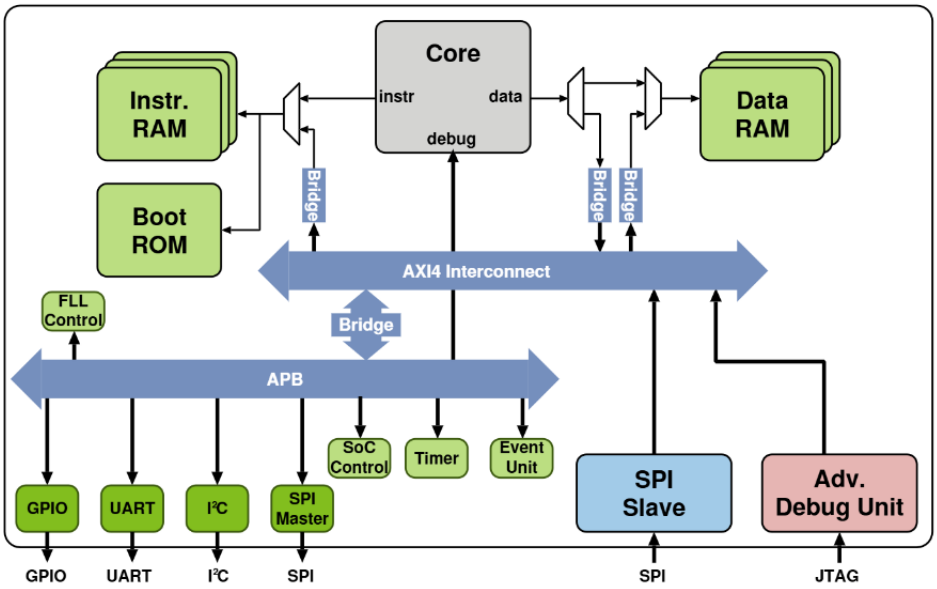


Figure 1: PULPino overview.

For more information on the PULPino platform, including its architecture, features, and how to get started with development, refer to the following resources:

- PULPino platform GitHub repository: `https://github.com/pulp-platform/pulpino`
- PULPino datasheet: `https://pulp-platform.org/docs/pulpino_datasheet.pdf`
- Local PULPino files: `/usr/local-eit/cad2/riscv/pulpino`

To gain a deeper understanding of the communication protocols used in PULPino, it is recommended to read about the APB (Advanced Peripheral Bus) protocol. A basic understanding of the bus protocol is required during implementation of a new peripheral.

- Detailed APB protocol specification: `http://web.eecs.umich.edu/~prabal/teaching/eecs373-f12/readings/ARM_AMBA3_APB.pdf`

Important!

The project can grow quite large, make sure to delete unnecessary files. This usually makes itself known by throwing a **Disk Quota Exceeded** message in the terminal. If you run out of disk space, try:

- Run **quota -s** to check your current disk quota. Remove files accordingly.
- Place your project on the local disk, however it is **not** backed up! You can create `/export/space/nobackup/<your-username>`. In this directory the you should not be limited by your quota. You will **only** be able to access this directory on the computer from which you created the directory. Make sure to backup your projects either on a USB stick or on GitHub or a similar service.

  Remember that you have signed an NDA and you are **NOT** allowed to move any technology specific files outside of the lab computers! This includes behavioural models for the SRAM! Keep this in mind if you plan on using GitHub or similar version control service.

This information is crucial for the completion of your project.

2 Getting started

In order to get started with the project, you will need to locate the necessary files, and get a basic understanding of the structure of the project. The project is quite large and you don't need to have a deep understanding of every module, but a general knowledge of the structure is required.

2.1 Setting up the project

All the necessary files for this project are located in `/usr/local-eit/cad2/riscv/`. You will need to copy the file containing the environment variables setup, `setup2022.efd`, the Modelsim initialization file containing the libraries, `modelsim_st65.ini`, as well as the PULPino directory, `pulpino`.

```
user@computer:<project_path>$ mkdir etin35_project
user@computer:<project_path>$ cd etin35_project
user@computer:<project_path>/etin35_project$ cp -r /usr/local-eit/cad2/riscv/
    setup2022.efd /usr/local-eit/cad2/riscv/modelsim_st65.ini /usr/local-eit/cad2/
    riscv/pulpino ./
```

Each time you start a new terminal, or open a new tab in the terminal, you will have to source `setup2022.efd` if you want to execute Modelsim, synthesize, or compile something.

```
source setup2022.efd
```

If you get an error regarding syntax error, enter the bash terminal by typing **bash** before sourcing the setup file.

You will also need to update the RTL files by pulling them from GitHub remote repository. For this, run the following script below.

```
user@computer:<project_path>/$ cd pulpino/
user@computer:<project_path>/pulpino$ python update-ips.py
```

If you get an error regarding **remote**, when running **python update-ips.py**, see the fix shown in Listing 2.

Listing 1: Potential error when running **update-ips.py**.

```
Using remote git server https://github.com, remote is https://github.com/pulp-
    platform
From https://github.com/pulp-platform/IPApproX
* branch          verilator -> FETCH_HEAD
Traceback (most recent call last):
File "update-ips.py", line 91, in <module>
ipdb.update_ips(remote = remote)
TypeError: update_ips() got an unexpected keyword argument 'remote'
```

Open the Python-script with the same name in a text editor, and make the following changes

Listing 2: Snippet from **update-ips.py** to resolve error.

```
# updates the IPs from the git repo
#ipdb.update_ips(remote = remote) # Before changes
ipdb.update_ips()
```

At this point, the python script should work and the IP:s should be pulled. The terminal will show print whether or not each ip has been successfully updated. The ip `adv_dbg_if` will not be updated, but this **should not** be an issue.

2.2 RTL simulation

A `helloworld`-project has been created for you, which will be used as the source for the simulation. Before you can simulate you need to start the RISCV GNU Compiler Collection (**RISCV GCC**). This will create a makefile for you, which is used to simplify the compilation and simulation startup.

Listing 3: Setting up the build directory

```
cd sw/
mkdir build
cp cmake_configure.riscv.gcc.sh build/
cd build/
source cmake_configure.riscv.gcc.sh
make helloworld.read
make vcompile
make helloworld.vsim (you can also try: make helloworld.vsimc)
```

It is recommended that you open `cmake_configure.riscv.gcc.sh` and `Makefile` in a text editor to understand what knobs you can turn and what the commands executed above, actually does.

Try making your own application and see if you are able to run the code in the simulation! For more information about this, see Ch. 2.3. It is highly recommended that you check this chapter as you are likely to encounter issues!

Open the file source file for the `helloworld` project. Make changes to the printf statement.

Question!

Try to recompile the source files for the project `helloworld`!

```
make helloworld.read
make helloworld.vsimc
```

Questions:

What happens when you simulate the RTL? Does it work? How do we resolve the issue?
This will be explained on the following page!

What does the command `make helloworld.read` do?

What does the command `make helloworld.vsimc` do?

Why didn't we use `make vcompile` this time? When do we need to use this command?

For more information about what the commands to and how the software directory is structured, read the file `<project_path>/pulpino/sw/build/README.md`

Important!

In order to rerun a project, you need to clean up some directories. Otherwise you will get an error. This error be resolved by running the command `make clean`, while being inside the `<project_path>/pulpino/sw/build/` directory. An example of the error that occurs, and the solution, is shown below

```
user@computer:~$ cd <project_path>/pulpino/sw/build
user@computer:<project_path>/pulpino/sw/build$ make helloworld.vsimc
[  0%] Built target string
[  0%] Built target Arduino_core
[ 33%] Built target Arduino_separate
[ 33%] Built target bench
[ 66%] Built target sys
[ 66%] Built target crt0
[ 66%] Built target helloworld.elf
[ 66%] Generating vectors/stim.txt
[ 66%] Built target helloworld.stim.txt
[ 66%] Generating modelsim.ini
make[3]: *** [apps/helloworld/modelsim.ini] Error 1
make[2]: *** [apps/helloworld/CMakeFiles/helloworld.links.dir/all] Error 2
make[1]: *** [apps/helloworld/CMakeFiles/helloworld.vsimc.dir/rule] Error 2
make: *** [helloworld.vsimc] Error 2
user@computer:<project_path>/pulpino/sw/build$ make clean
user@computer:<project_path>/pulpino/sw/build$ make helloworld.vsimc
..
..
[100%] Running helloworld in ModelSim
Hello World!!!!!
[100%] Built target helloworld.vsimc
```

2.2.1 Scripting the simulation

As have been repeated numerous times in this document, scripting is the love language for hardware designers. There are a lot of different types of scripting languages, some of them you may already be familiar with, like Python, Bash, and JavaScript. During this project, you will encounter Tcl and Bash. Scripting is a great way to reduce the headache of keeping track of all different Makefiles and commands, and can save you a lot of time!
Note that this it is not mandatory to create this script, but is likely to help you in your further endeavours as hardware designers.
To start off, what do we need to do in order to simulate the RTL?

- ☐ Set up environment variables, to allow us to use the required software, like Modelsim
- ☐ Clean the project

- □ Compile the C-project
- □ Compile the RTL
- □ Run the simulation

In the directory `<project_path>/pulpino/sw/build/`, create a new file called `my_first_script.sh`. Open the file with a text editor and add the following

Listing 4: my_first_script.sh, a script to perform the steps required to compile and simulate.

```
#!/bin/bash

# Source the setup file
source ../../../setup2022.efd

# Clean
make clean

# The name of your C project to be compiled. Change this to the the project that you
    want to compile & run!
PROJECT_NAME=my_first_project

# Execute the make commands using the project name variable
make "${PROJECT_NAME}.read"
make vcompile
make "${PROJECT_NAME}.vsimc"
```

Now we need to change the permissions on the file, to make it executable! This is done via the `chmod +x filename` command. The permissions only needs to changed once after the file has been created. Next time you run the script, you should only have to type `./my_first_script.sh`. *Tips! The TAB key on your keyboard can help you to autocomplete file names, commands etc.*

Open a terminal and go to the directory containing your script, then run the script!

Listing 5: Changing permissions and running the script.

```
user@computer:$ cd <project_path>/pulpino/sw/build
user@computer:<project_path>/pulpino/sw/build$ chmod +x my_first_script.sh
user@computer:<project_path>/pulpino/sw/build$ ./my_first_script.sh
```

2.3 Adding a new C-project

All of the software projects are located in the directory `<project_dir>/riscv/pulpino/sw/apps`. Note that this directory differs from `<project_dir>/riscv/pulpino/sw/build/apps`, which contains the compiled projects.

At this point, you have simulated the `helloworld` project, and you're ready to create your first project.

```
user@computer:$ cd <project_path>/pulpino/sw/apps
user@computer:<project_path>/pulpino/sw/apps$ mkdir my_first_project
user@computer:<project_path>/pulpino/sw/apps$ cd my_first_project
user@computer:<project_path>/pulpino/sw/apps/my_first_project$ pluma my_first_project.c
user@computer:<project_path>/pulpino/sw/apps/my_first_project$ pluma my_first_function.h
user@computer:<project_path>/pulpino/sw/apps/my_first_project$ pluma my_first_function.c
user@computer:<project_path>/pulpino/sw/apps/my_first_project$ pluma CMakeLists.txt
user@computer:<project_path>/pulpino/sw/apps/my_first_project$ cd ..
user@computer:<project_path>/pulpino/sw/apps$ pluma CMakeLists.txt
```

Note! There are multiple `CMakeLists.txt` files that you need to edit. Make sure you are keeping track of which is which! Your file structure should look like Fig. 2. Note that only the relevant files and directories are shown in this figure.

Write your main project code in `my_first_project.c`, which contains the main function, the entry point of your program. Declare interfaces to your functions in the header file `my_first_function.h` and define the actual functionality in `my_first_function.c`.
In `my_first_function.c`, implement a function that performs a task of your choice. Ensure your function provides a visible output (like printing a message) to verify its execution when called from the main function.

In `<project_path>/pulpino/sw/apps/my_first_project/CMakeLists.txt` you need to add the application by specifying the name of the project and the source file(s), as shown below.

Listing 6: /sw/apps/my_first_project/CMakeLists.txt, example entry

```
set(SOURCES my_first_project.c my_first_function.c)
add_application(my_first_project "${SOURCES}")
```

Finally, we need to add the recently created project directory, to the `CMakeLists.txt` that specifies the directories to be added to the compilation. Open the file existing file `<project_path>/pulpino/sw/apps/CMakeLists.txt` and scroll down to the bottom of the document. Add the following line to let CMake know about the add the subdirectory to

Listing 7: /sw/apps/CMakeLists.txt, example entry

```
add_subdirectory(my_first_project)
```

Below is a directory tree showing some relevant files for the previous steps.

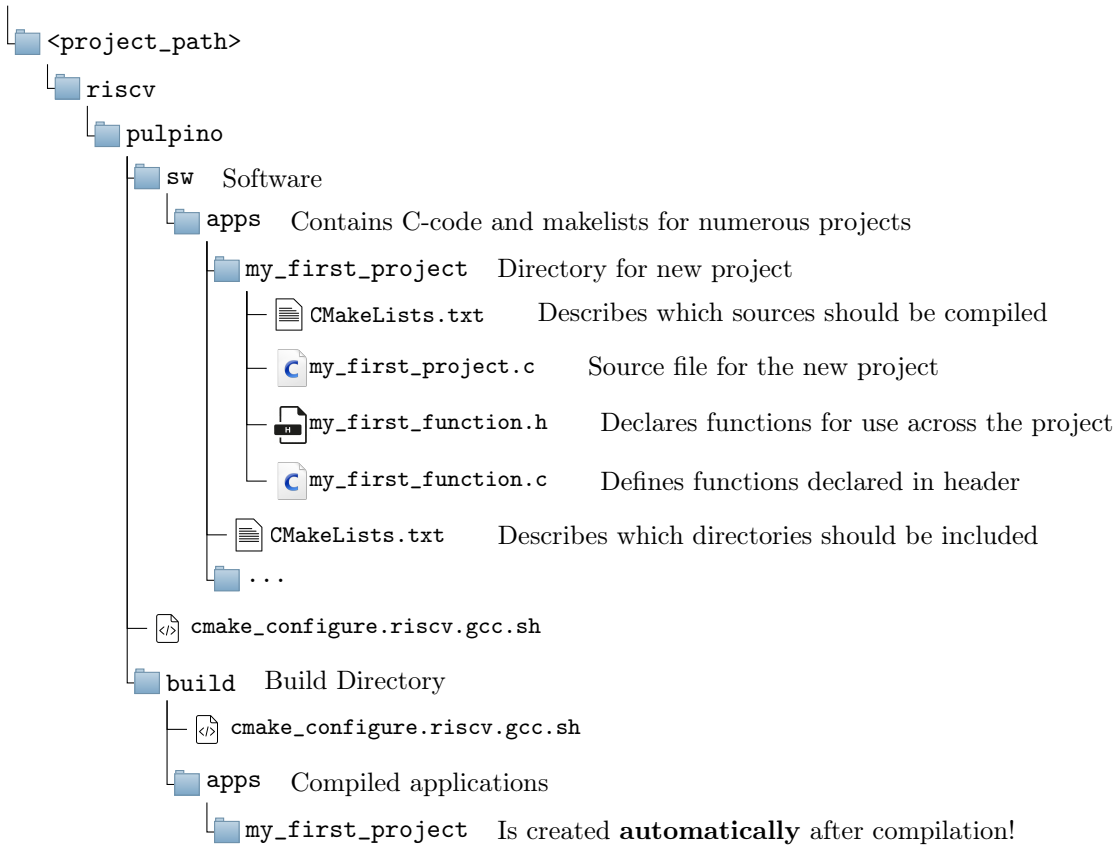


Figure 2: Directory tree showing the structure for adding a new C-project.

3 Starting the project

At this point, you need to start planning out the necessary work required to fulfill the tasks in the project, see Ch. 4. Some of the files in the project may not be synthesizable; it is your job to understand why and how this can be resolved.

If you have not done so yet, you need to look through the directories and understand which part of the modules does what.

Start off by swapping the systems existing memories to the SRAM provided, see Ch. A.2, which is 2048×8. Figure out where the RAM is instantiated, and create a wrapper so that you
8192 can swap the memory to fit the 8096×32 RAM of the existing architecture. Take note of which functions the memories require, e.g., *is it byte-enabled? Does it require an enable signal? etc.* The Pulpino system contains both a Data and an Instruction memory. Both of these should be changed.

After swapping the memories, make sure that the RTL simulation is still valid, when using the behavioral model of the memories.

4 Tasks

4.1 Grade 3

1. Run the *Hello World* program as an RTL simulation to verify functionality
2. Using the 2048×8-bit SRAM module, create a 32 kB, 32-bit wide memory with byte-enable functionality, compatible with the Pulpino platform's requirements. Both the Data and Instruction memory in the Pulpino platform should be changed.
   - Remember to verify the design after making changes to the system!
3. Switch from FLL to External Clock Source
4. Remove unnecessary peripherals to limit the number of pads to 32
5. Define separate clock domains for every clock signal in the ASIC flow
   - Somewhere in the design, there is at least one clock domain crossing, configure the constraints appropriately. Are they synchronous or asynchronous to each other?

6. Synthesize the design
   - The synthesis flow should be scripted to ensure reproducibility
7. Place and route the design
   - The PnR flow should be cleanly scripted, it is suggested that you creating multiple scripts for the different phases e.g.,
     - init.tcl
     - place.tcl
     - cts.tcl
     - postcts.tcl
     - route.tcl
     - postroute.tcl
   - Aim for a reasonable utilization. If your density is low, remove unnecessary pads and reduce the area. If you are core limited, add more PG pads
   - Minimum pad pitch, 90 μm
   - Target the maximum feasible operating frequency
     - Note the critical path. You should be able to understand and explain why it is not possible to achieve a higher frequency without changing the design
8. Simulation
   - RTL, initial verification
   - Gate level, synthesis verification (Post-synthesis)
   - Layout, implementation verification (Post-layout)
9. Write a C-program the performs the identical operations as your matrix multiplication unit, on the core
10. Benchmark matrix multiplication on RISC-V vs accelerator
    - Performance
    - Power (Primetime)

4.2 Grade 4

1. All tasks required for grade 3.
2. Replace the a peripheral with the matrix multiplier accelerator, created as a part of this course
   - This requires the creation of a wrapper from which the accelerator can be attached to the APB bus [1].

[1] APB Protocol: `https://developer.arm.com/documentation/ihi0024/latest/`

3. Write an application in C for integration and testing of the accelerator in the MCU
4. Configure the floorplan and re-run the ASIC flow
5. Verify your design through simulation
6. Benchmark matrix multiplication on RISC-V and compare it against accelerator as a peripheral
   - Performance
   - Power (Primetime)

Tip: Create an APB test bench prior to attaching the peripheral to the system. It may be easier to test the functionality of the accelerator outside of the SoC, though this is suggestion **not** a requirement for this course.

4.3 Grade 5

For grade 5, you do **not** need to perform the tasks for grade 4, i.e., implementing the matrix multiplier accelerator to the PULPino is not required.

1. All tasks required for grade 3 (Tasks for grade 4 not required, but are highly recommended to do)
2. Create a convolution accelerator for an IFM with size 28×28 compatible with the APB bus, with
   - ■ Filter size 3×3 and zero padding
   - ■ Filter size 5×5 and zero padding
   - ■ Filter size 3×3 and edge padding
   - ■ Filter size 3×3 and stride 2
   - ■ Filter size 5×5 and stride 2
   - □ Filter size 5×5 and max pooling (stride 2, window size 2×2) **OPTIONAL**

   Note: The configuration is determined by the supervisor! Before initiating the work for grade 5, contact your supervisors to get the filter size of your convolution accelerator
   - ■ The accelerator design should optimize for either low power, or high performance
   - ■ To store the OFM and weights, implement a memory mapped SRAM to the accelerator
3. Replace the timer peripheral with the matrix multiplier accelerator, created during the first phase of this course
4. Write an application in C for integration and testing of the accelerator in the MCU
5. Configure the floorplan and re-run the ASIC flow
6. Verify the design through simulation
7. Benchmark convolution on RISC-V and compare it against convolution on your accelerator as a peripheral
   - ■ Performance
   - ■ Power (Primetime)

5 Contact details

The TAs for the project **RISC-V SoC for ML: From RTL to GDSII** in the course **ETIN35** for this period (*Spring 2025*), and their contact details are listed below. Besides the regularly scheduled group meetings, you can either contact the supervisors via email or Teams, or you can visit their room during office hours.

Sergio Castillo
Co-Supervisor
sergio.castillo_mohedano@eit.lth.se
Room Number: E:2339

Kristoffer Westring
Main Supervisor
kristoffer.westring@eit.lth.se
Room Number: E:2339

Fuad Mammadzada
Co-Supervisor
Mail: fu6315ma-s@student.lu.se
Room Number: E:2339

A Appendix

A.1 Cadence Documentation

You can find the documentation for the Genus and Innovus by typing `/usr/local-eit/cad2/cadence/gen161/bin/cdnshelp` respectively `/usr/local-eit/cad2/cadence/inn21/bin/cdnshelp` into a terminal. This will open a new window in which you can find documents explaining different parts of the backend flow. You can also search for commands for valuable information about e.g., arguments.

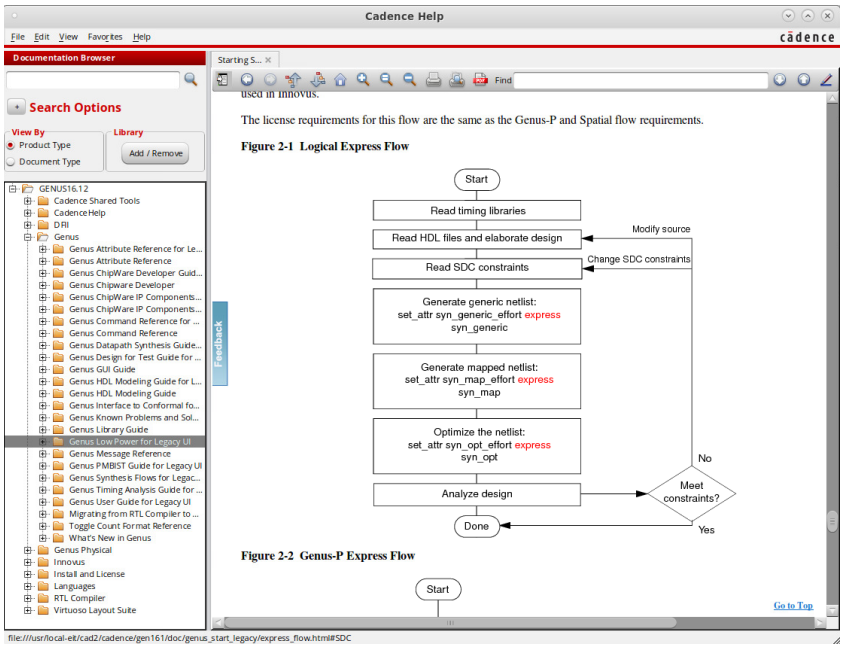
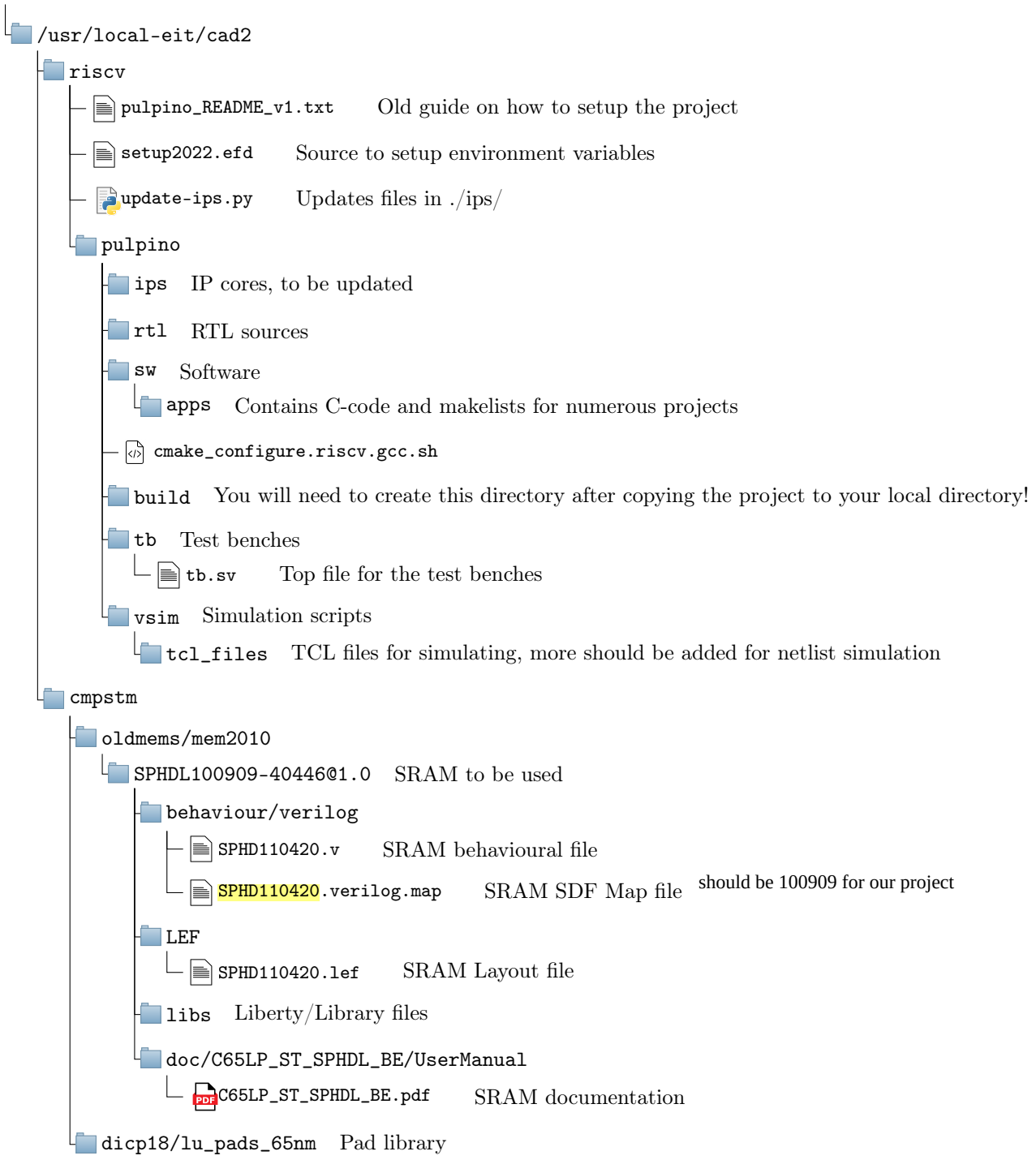


Figure 3: Cadence Help for Genus.

A.2 Relevant files and directories

This directory tree shows the location of *some* of the more important directories and files. Note that the list is **not** a complete list over which files you will need to modify and familiarize yourself with. It is instead meant to act like a starting reference to help you navigate the project. You will need to gain a deeper understanding from experience.


/usr/local-eit/cad2
riscv
pulpino_README_v1.txt
Old guide on how to setup the project
setup2022.efd
Source to setup environment variables
update-ips.py
Updates files in ./ips/
pulpino
ips
IP cores, to be updated
rtl
RTL sources
sw
Software
apps
Contains C-code and makelists for numerous projects
cmake_configure.riscv.gcc.sh
build
You will need to create this directory after copying the project to your local directory!
tb
Test benches
tb.sv
Top file for the test benches
vsim
Simulation scripts
tcl_files
TCL files for simulating, more should be added for netlist simulation
cmpstm
oldmems/mem2010
SPHDL100909-40446@1.0
SRAM to be used
behaviour/verilog
SPHD110420.v
SRAM behavioural file
SPHD110420.verilog.map
SRAM SDF Map file
should be 100909 for our project
LEF
SPHD110420.lef
SRAM Layout file
libs
Liberty/Library files
doc/C65LP_ST_SPHDL_BE/UserManual
C65LP_ST_SPHDL_BE.pdf
SRAM documentation
dicp18/lu_pads_65nm
Pad library