Caravel Tutorial Document

Version of Software Tools:

Windows 11

Ubuntu 20.04

Docker 23.0.1

Python 3 with pip 3.8.10

1. Overview of the Efabless Process Steps....4
2. Windows Subsystem for Linux Installation (WSL2) on Windows 11....6
2.1 Opening Power Shell....6
2.2 Installing WSL2....7
3. Setting Up Dependencies for the Caravel Repository....8
3.1 Updating Ubuntu to latest version....8
3.2 Installing Docker in Ubuntu....10
3.3 Troubleshooting....12
3.4 Installing Python 3 with PIP....13
3.5 Installing klayout (Optional)....13
4. Getting Started with the Caravel Repo....14
4.1 Cloning your Project Repository into Linux....14
4.2 Setting up Local Caravel Environment....14
4.3 Before hardening user project....15
4.4 Hardening user project to test open lane setup: Example - User_adder....15
4.5 Before Hardening the user project wrapper....16
4.6 Creating user project wrapper....17
4.7 Running test bench simulations on your design....18
4.8 Example....18
4.9 Running final checks on your project before submission....19
5. Getting started with the Wishbone Bus....21
5.1 What is the wishbone bus....21
5.2 Using the wishbone bus from the Management SoC....21
5.3 Writing to the user area....21
5.4 Reading from the user area....22
6. Getting started with the Logic Analyzer....24
6.1 What is the logic analyzer....24
6.2 How to use the logic analyzer in the user area....24
6.3 How to use the logic analyzer from the management SoC....24
7. How to use interrupts....25
7.1 What is an interrupt....25
7.2 How to trigger an interrupt from the user area....25

7.3 How to receive an interrupt in the SoC....25
8. How to use klayout to view GDS files....27
9. How to submit your project design to Efabless....30
10. Caravel PCB daughter & carrier boards you receive from Efabless....33
10.1. Introduction....33
10.2.0. Overview of caravel_board....33
10.2.1. Power management section....33
10.2.2. FPGA section....33
10.2.3. GPIO section....33
10.2.4. JTAG section....33
10.3.0. Overview of caravel_Nucleo....34
10.3.1. ST-Link debugger....34
10.3.2. USB interface....34
10.3.3. Buttons and LEDs....34
10.4.0 How the parts on the board connect to each other?....34
10.4.1. FPGA connections....34
10.4.2. Power management connections....34
10.4.3. GPIO connections....35
10.4.4. JTAG connections....35
10.4.5. ST-Link connections....35
10.4.6. Button and LED connections....35
10.5. How the caravel_board connects to the user area and how to test it?
35
10.6. Conclusion....36
10.7. Full form of all abbreviations....37
11. References....37

1. Overview of the Efabless Process Steps

The Caravel project is a long project that will take time and dedication to complete. In this section an overview of the step-by-step process for the caravel project will be explained from start to finish.

1) The very first thing that will need to be done is to familiarize yourself with the Caravel harness.
    a) You will want to study all of the components that make up the Caravel harness and their functionalities. This can be done by reading through the following website that contains information about the user harness.
    b) Link: Efabless Caravel “harness” SoC — Caravel Harness documentation (caravel-harness.readthedocs.io)

2) You will have to download all necessary efabless tools and familiarize yourself with how they work.
    a) how to do so is explained in the sections below.

3) You will need an open-source Git repository.
    a) This is to store the caravel project as well as give detailed documentation on what your project does and how it works in the form of a README.
    b) If you are part of a senior design team the ETG will set up your repository for you, but it will be up to you to update your repository as necessary.

4) Next you will need to come up with a chip design idea that you believe will be able to fit within the user area of the Caravel harness slowly making a high level RTL design of your project Idea to implement into the user area.

5) After coming up with an idea and creating a viable RTL for it you will need to slowly add all of the components/modules that comprise your design idea into the user area
    a) Be sure to set up test benches testing each module individually to check for functionality before implementing them into the caravel harness.
    b) Then slowly integrate each module into the caravel harness setting up test benches to check for functionality between each of the modules you have as you implement them all into the user area.

6) After successfully integrating your design into the user area next you will need to harden your design inside of your lynux environment.
    a) Explanation on how to do so is in section 4.3.

7) Once you have successfully hardened your design you will need to instantiate it in the project wrapper.

8) After you have instantiated your design inside of the wrapper you will harden the project wrapper.
    a) Explanation on how to do so is in section 4.4.

9) After hardening and creating the new project wrapper you will test its functionality using the test benches given in the caravel repository as well as any personal test benches you design for your project to ensure that it is working properly.
    a) Explanation on how to do so is in section 4.5.

10) Once your project has passed all test benches the last thing to do is to run the final checks on it. Your project must pass these prechecks before you attempt to submit your design to Efabless
    a) Explanation on how to do so is in section 4.6.

11) After your project has passed the final prechecks you will submit your project to efabless. Keep in mind that it may take several hours for your design to be fully submitted to efabless due to the final tests that will need to be performed on your project submission on efabless submission page.
    a) Explanation on how to submit your project and what needs to be done is in section 8.

12) The next thing that needs to be done is to prepare a bring-up plan for your project. You will need to come up with steps to test the physical PCB board & all componnents on it, as well as testing your overall project's functionality. Essentially coming up with tests that have hardware and software aspects to it to ensure that everything is functioning properly.
    a) To understand what to expect to be on the PCB boards that you will receive you may review all of section 9 of this document.

13) Eventually you will get your project design back in the form of a die soldered to a PCB board. You will obtain a carrier board as well as a daughter board. The only thing left to do is to execute your bring-up plan to see how well your project turned out.

2. Windows Subsystem for Linux Installation (WSL2) on Windows 11

Source: [Install WSL | Microsoft Learn]

2.1 Opening Power Shell

First open PowerShell on your windows computer, you can do so by clicking in the search bar at the bottom of your windows interface circled in green as seen in figure 1.

Figure 1. Windows search bar

A search window, as displayed in figure 2 should appear.

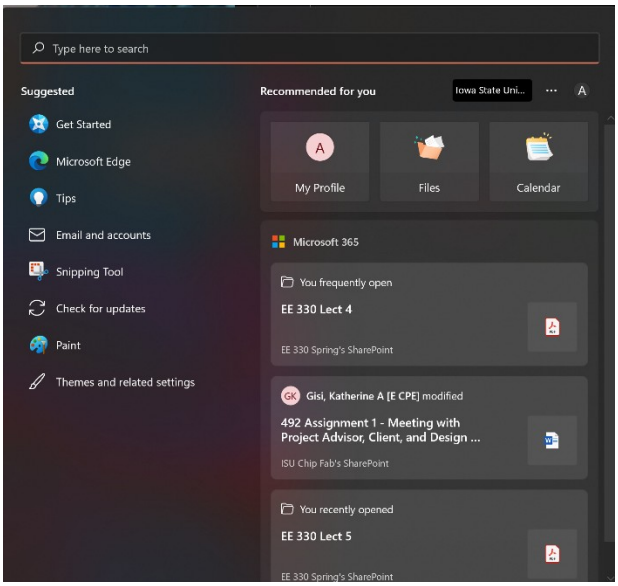


Figure 2. Windows search window

Afterwards type in PowerShell into your search window and open it as seen in figure 3.

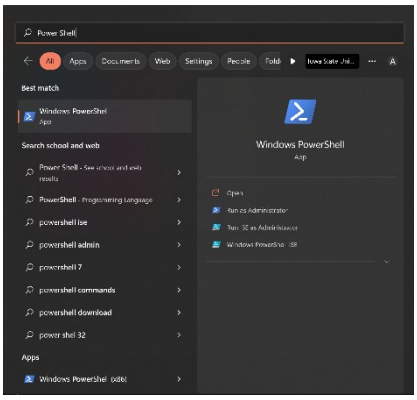


Figure 3. PowerShell App

2.2 Installing WSL2

Once PowerShell is up and running type in the following command into the command line and run it refer to figure 4.

Command: wsl --install

This will enable all the necessary features in order to run Linux on your machine. It will also install the Ubuntu distribution of Linux on your machine. You will be prompted to create a username and password for Ubuntu, it is very important that you remember your password as you will need it every time you open Ubuntu and attempt to run the first command.

Figure 4. Installation of wsl through PowerShell

3. Setting Up Dependencies for the Caravel Repository

3.1 Updating Ubuntu to latest version

The first thing that will need to be done is to make sure that your system is running on the latest version of Ubuntu. To do so you will have to open up Ubuntu by typing it in the same search bar the same way that you opened PowerShell. After doing so type the following command into the command line and run it. Before Ubuntu executes the command, you may be prompted to type in your password that you created in the previous step. Refer to figure 5.

Command: sudo apt update

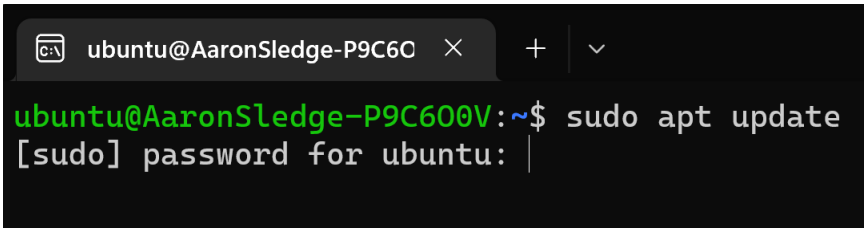


Figure 5. Ubuntu

As can be seen in figure 5 you will be prompted to type in your password before Ubuntu executes the above-mentioned command. It is important to note that as you type in your password Ubuntu will not display any of the characters that you type in, this is for security reasons. Understand that Ubuntu is receiving the characters that you are typing there is nothing wrong with your terminal.

After typing in the correct password Ubuntu will execute the command that you entered prior to it prompting you with a request for your password as can be seen in figure 6.

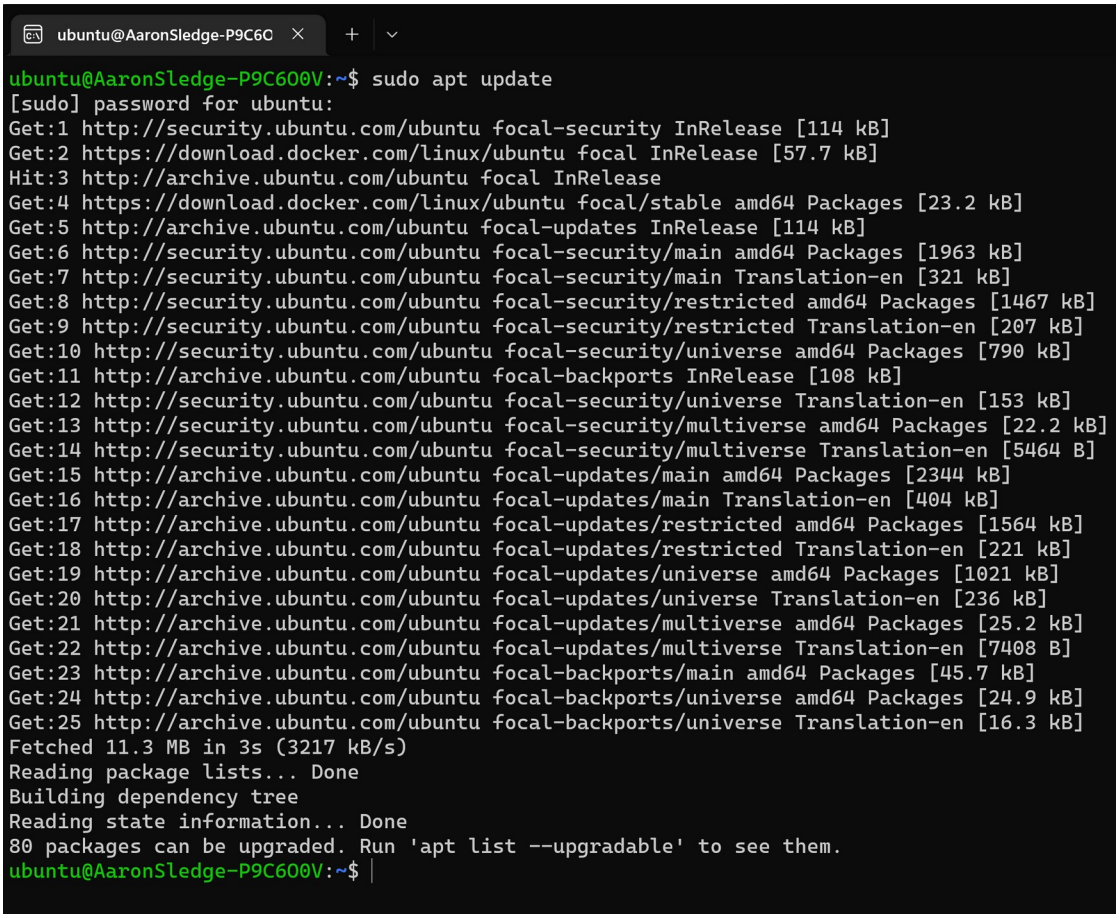


Figure 6. Execution of first command

After Ubuntu has executed the “sudo apt update” command in the second to last line in figure 6 it will tell you the number of packages you are able to upgrade as well as how to view the list of upgradable packages if needed.

Type the following command into the command line and run it. This command will upgrade all of the packages listed from the prior command. You will be promoted with a question asking if you wish to continue where you can enter “Y” for yes and “n” for no, refer to figure 7.

Command: sudo apt upgrade

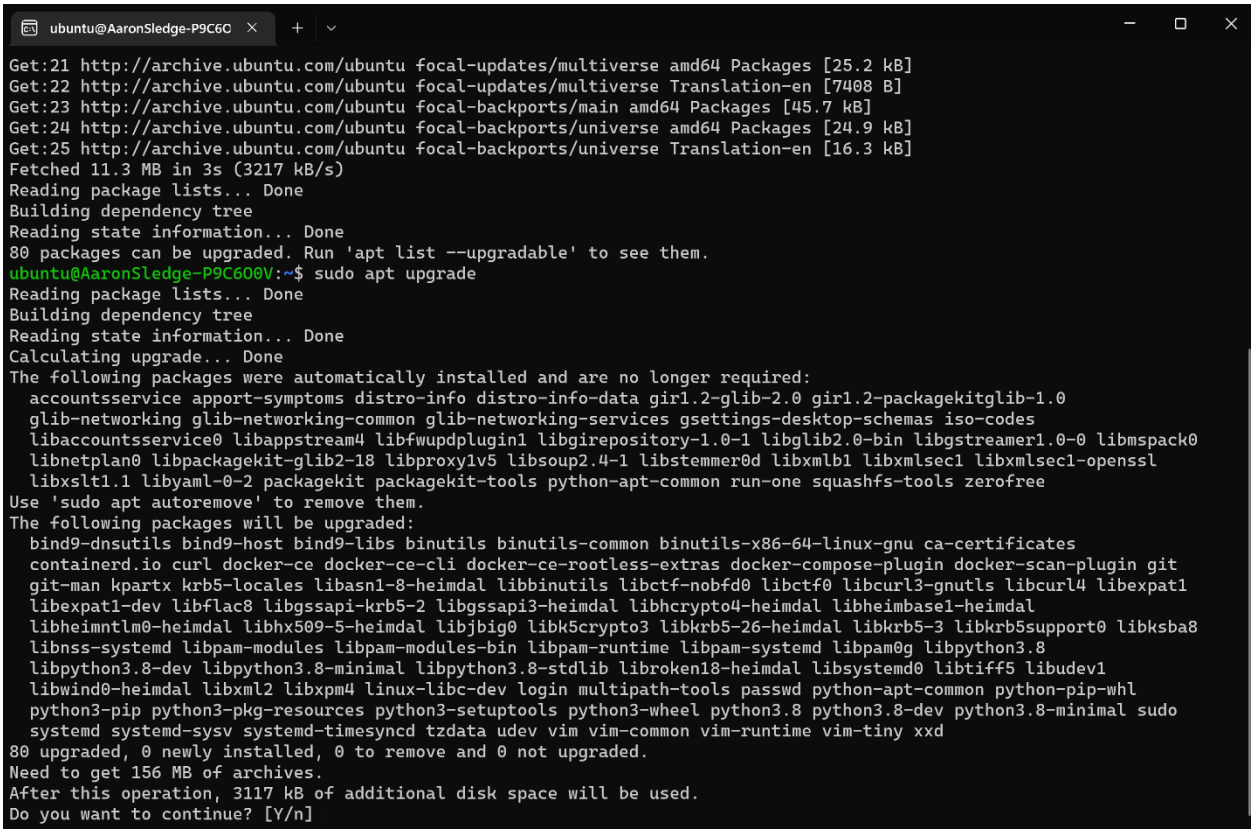


Figure 7. Ubuntu executing sudo apt upgrade

Enter in 'Y' and Ubuntu will upgrade your packages. Depending on the number of packages that need to be upgraded it could take Ubuntu anywhere from a few seconds to a few minutes to finish upgrading the packages. Once Ubuntu has finished it will display "done" in the second to last line, indicating that the upgrading process was completed and successful.

3.2 Installing Docker in Ubuntu

Source: Install Docker Engine on Ubuntu | Docker Documentation

Docker for WSL

First you must be in your home directory, to do so enter the following command into the command line and run it. This will return you to your home directory.

Command: cd

Afterwards you will have to update the apt package index, enter in the following command in the command line and run it. The output should look like figure 8 below.

Command: sudo apt-get update

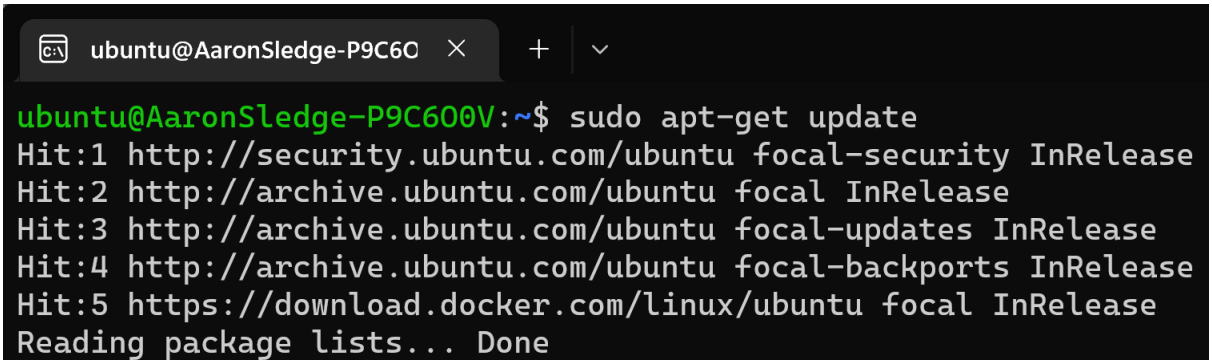


Figure 8. Output of sudo apt-get update command

If Ubuntu is unable to run the previous command then your default unmask could be incorrectly configured. If that is the case, then type in the following commands in the command line and run each one separately.

Command: sudo chmod a+r /etc/spt/keyrings/docker .gpg

Command: sudo apt-get update

Next you will have to install the Docker Engine, container, and Docker Compose. Type in the following command in the command line and run it. The following command will install the latest version of docker on your machine.

Command: sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin

To verify that docker installed correctly type in the following command line and run it. The output should look like figure 8 down below, if it does then you have successfully downloaded docker into your Linux environment.

Command: sudo docker run hello-world

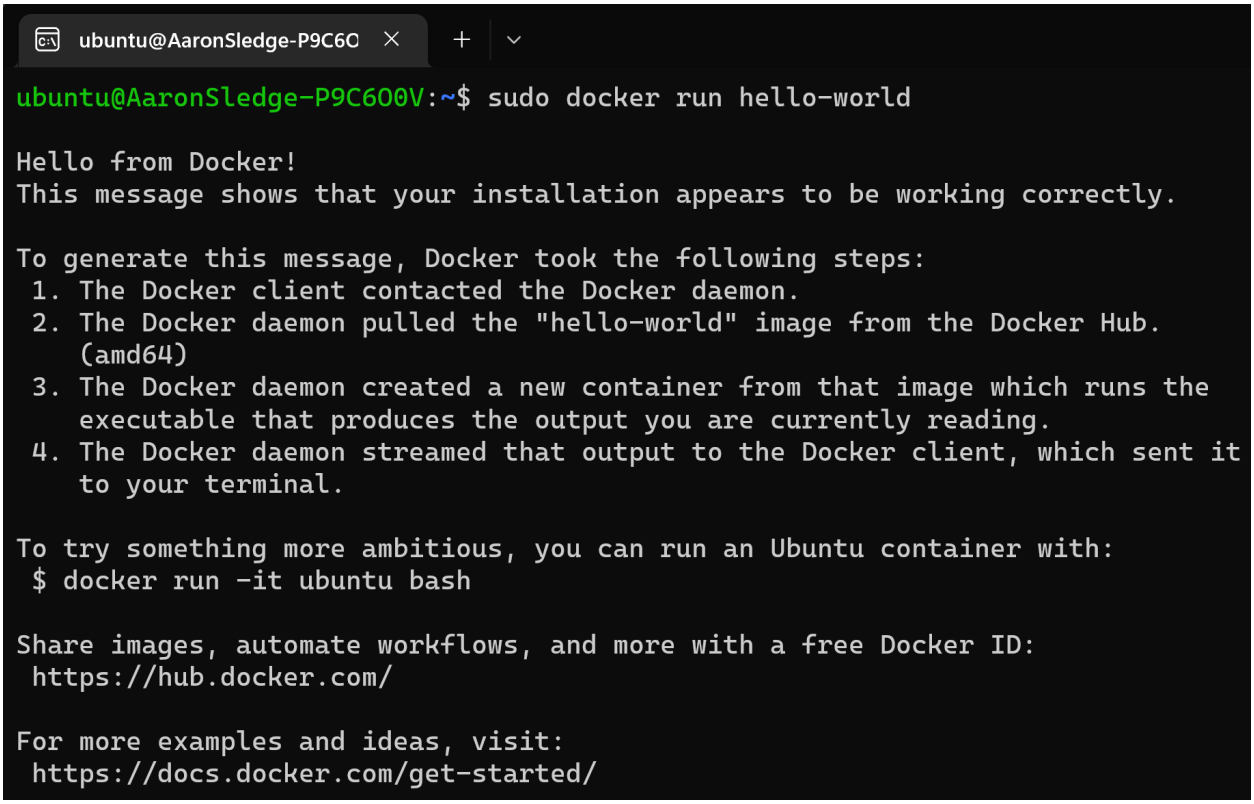


Figure 9. Correct Docker Installation Output

3.3 Troubleshooting

Based on past experience, WSL may or may not work easily with docker. The recommended way to use WSL and docker is to install Docker Desktop for Windows, then enable a setting to connect to WSL; however, there have also been issues where Docker Desktop does not start correctly on Windows either. To avoid using Docker Desktop, you can try following the normal install steps for a traditional Ubuntu installation. This has its own issues, specifically with GPG key errors and the daemon not starting correctly by default.

GPG key error:

`sudo apt-get update` may fail with an error about gpg keys. To fix this, you can run the following from the directory “/etc/apt/keyrings”:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

Then try `sudo apt-get update` again

Docker daemon not running:

The docker daemon may not start after installing docker, so `sudo docker run hello-world` give an issue with not being able to communicate with the docker daemon. To solve this, try running `sudo dockerd &`. This runs the command in the background, but the output will still be sent to the terminal. You should still be able to run `sudo docker run hello-world`. If that does not work, then enter in the command below:

Command: sudo service docker start

Your daemon error should be fixed by now.

3.4 Installing Python 3 with PIP

Source: [How to Install Python Pip on Ubuntu 20.04 | Linuxize]

Installing Python 3 with Pip is the final dependency needed in order to use the caravel (repository). Type the following command into the command line. This will install the latest version of Python 3 along with PIP at the same time.

Command: Sudo apt install python3-pip

After running the previous command type the following command in the command line to verify that the installation executed correctly.

Command: pip3 --version

The output should look like figure 9 down below, the versions number may vary.

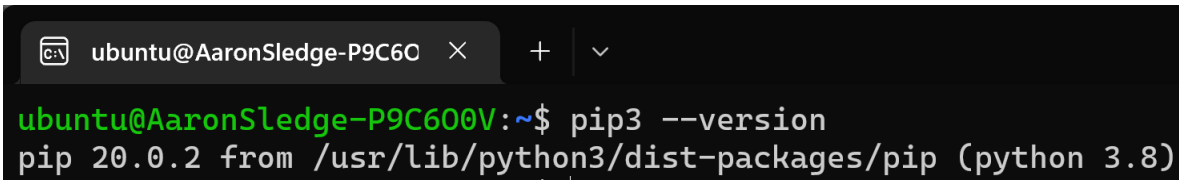


Figure 10. Pip and Python version check output

3.5 Installing klayout (Optional)

All though this software tool is not required in order to be able to use the caravel environment it is a nice visual aid that will help with understanding how much space your design occupies in the user project area. Type and run the following commands:

1. sudo apt update
2. sudo apt upgrade
3. sudo apt –y install klayout

It should only take your terminal a few minutes at most to download the klayout software tool. Again it is not necessary but can prove to be extremely useful when it comes to understanding how much of the user area space is taken up by your design.

4. Getting Started with the Caravel Repo

Source: [caravel_user_project/quickstart.rst at main · efabless/caravel_user_project · GitHub]

Before you are able to do anything with the Caravel repository you must first set up the local environment inside of Linux inside of your project repository.

4.1 Cloning your Project Repository into Linux

First you must clone your project repository given to you by the ETG into Linux. Type the following command into your command line and run it.

Command: git clone <your repo URL>

Upon running that command, you may be prompted to give your username and password in order to clone your repo. The username should be your school email and the password should be your school password. After entering that Linux should clone your repository without issue.

4.2 Setting up Local Caravel Environment

The first thing that you will need to do inside of Linux is to enter into your project repository. Type the following command into your command line and run it to enter your project repository.

Command: cd ~/<your project name>

Next you will need to run the following commands one at a time.

Commands:

1. mkdir dependencies (you only need to do this once)
2. export OPENLANE_ROOT=$(pwd)/dependencies/openlane_src
3. export PDK_ROOT=$(pwd)/dependencies/pdks
4. Make setup

If you are running into an error with the communication with daemon from docker please refer to the trouble shooting section under installing docker in Ubuntu.

Commands 2 & 3 will have to be exported every time that you start a new shell (every time you reopen your terminal). After the fourth command is executed, the following tools will be installed:

- Caravel_lite
- Management core for simulation
- Openlane to harden your design
- Pdk

These installations should only take a few minutes to complete.

4.3 Before hardening user project

Source: MPW-6 Walkthrough - YouTube (17:20 & 21:36)

Before you go to harden your project there are a few files that have to be modified so that the make command knows what to harden. The first file that has to be changed is in the user project example directory and is called the config.tcl file. You may find and change the config file under caravel_user_project/openlane/user_proj_example/config.tcl or you may make your own directory inside of openlane and create your own config file. Below I will show you how to modify the config file that is already present in the directory that I described above.

1. In the config.tcl file you will change the “DESIGN_NAME” section to the name of your project, so change line 21 by replacing user_proj_example with your project name.

2. Next you will have to change line 25, changing the user_proj_example.v file to your projects .v file or files depending on how many different Verilog scripts your project has.

3. If you have an issue with hardening your project you may have to play around with lines 34 and 39. For line 34 only change the right two numbers for the die area. So only change the 900 and or the 600. And for line 39 you must choose a number that is no greater then 1. But to keep the number as low as possible is desirable as you may receive a number of errors or violations for trying to make the placement target density too high.

```
20
21  set ::env(DESIGN_NAME) user_proj_example
22
23  set ::env(VERILOG_FILES) "\
24      $::env(CARAVEL_ROOT)/verilog/rtl/defines.v \
25      $script_dir/../../verilog/rtl/user_proj_example.v"
26
27  set ::env(DESIGN_IS_CORE) 0
28
29  set ::env(CLOCK_PORT) "wb_clk_i"
30  set ::env(CLOCK_NET) "counter.clk"
31  set ::env(CLOCK_PERIOD) "10"
32
33  set ::env(FP_SIZING) absolute
34  set ::env(DIE_AREA) "0 0 900 600"
35
36  set ::env(FP_PIN_ORDER_CFG) $script_dir/pin_order.cfg
37
38  set ::env(PL_BASIC_PLACEMENT) 0
39  set ::env(PL_TARGET_DENSITY) 0.05
40
```

Figure 11. config.tcl file for user project example

4.4 Hardening user project to test open lane setup: Example - User_adder

Testing your openlane set up is essential due to the fact that you must be able to harden your design before you are able to do anything else regarding too project submission to Efabless. Type and run the following command:

Command: make user_adder

Your terminal will then run through 48 steps, by the end of the process the output of your terminal should look the same as figure 12 below.

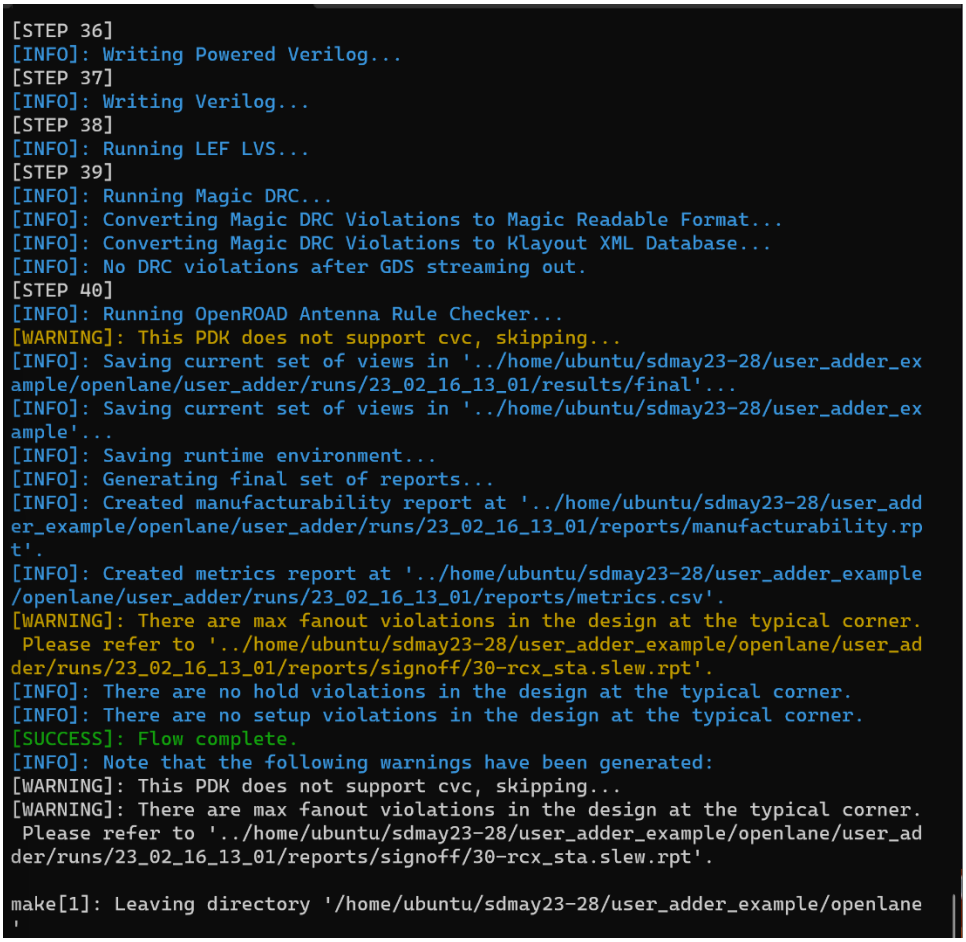


Figure 12. Output of hardening of user adder successfully

As can be seen in figure 12 there are two warnings given, however you can ignore these warnings for now because they will not hinder you any time soon.

4.5 Before Hardening the user project wrapper

Before you go on to harden your project inside of the project wrapper you once again have to change the contents of the corresponding config file or create your own. For this example, I will describe how to change the config file that is already present in the caravel user project. The location of the config.tcl file is in caravel_user_project/openlane/user_proj_wrapper/config.tcl.

1. You will have to edit line 42 changing the mprj clock to whatever the name is of the clock that you made for your project.

2. Change lines 57,60, & 63 to your projects corresponding file types.

```
32
33  # User Configurations
34
35  ## Source Verilog Files
36  set ::env(VERILOG_FILES) "\
37      $::env(CARAVEL_ROOT)/verilog/rtl/defines.v \
38      $script_dir/../../verilog/rtl/user_project_wrapper.v"
39
40  ## Clock configurations
41  set ::env(CLOCK_PORT) "user_clock2"
42  set ::env(CLOCK_NET) "mprj.clk"
43
44  set ::env(CLOCK_PERIOD) "10"
45
46  ## Internal Macros
47  ### Macro PDN Connections
48  set ::env(FP_PDN_MACRO_HOOKS) "\
49      mprj vccd1 vssd1 vccd1 vssd1"
50
51  ### Macro Placement
52  set ::env(MACRO_PLACEMENT_CFG) $script_dir/macro.cfg
53
54  ### Black-box verilog and views
55  set ::env(VERILOG_FILES_BLACKBOX) "\
56      $::env(CARAVEL_ROOT)/verilog/rtl/defines.v \
57      $script_dir/../../verilog/rtl/user_proj_example.v"
58
59  set ::env(EXTRA_LEFS) "\
60      $script_dir/../../lef/user_proj_example.lef"
61
62  set ::env(EXTRA_GDS_FILES) "\
63      $script_dir/../../gds/user_proj_example.gds"
64
```

Figure 13. Config file fore user project wrapper

4.6 Creating user project wrapper

Before you are able to run the final checks you must first take your harden design and place it inside of the caravel project wrapper. After doing so type and run the following command to create the user project wrapper inside of your environment.

Command: make user_project_wrapper

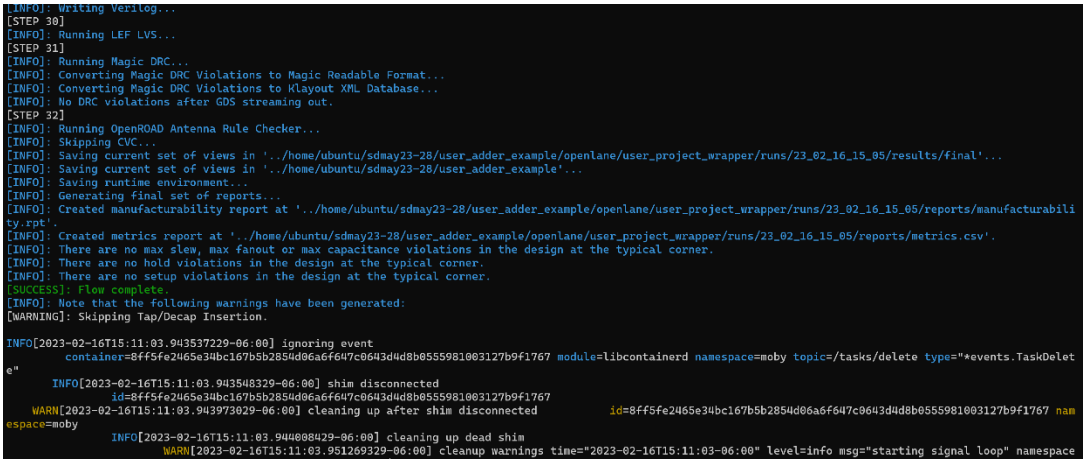


Figure 14. Output of making the user wrapper

4.7 Running test bench simulations on your design

After hardening your design inside of the user project wrapper there are only a few more things that will need to be done before your design is ready for submission to Efabless. First you will be instructed on how to run personal test benches for the user adder project so that you can familiarize yourself with how the output of the tests should look like inside of the terminal. However, the example will only use personal test benches because the user adder project will fail all other test benches that your project must pass before submission to Efabless. The other test benches that your project will need to pass are for the io ports, mprj, the logic analyzer, and wishbone bus which will be addressed after the example with the user adder project. Your project will need to be tested at both rtl and gl and passed at both levels before your design is ready for submission.

4.8 Example

There are two personal test benches to run on the user adder, the commands for each of those tests are as follows:

Command: make verify-user_adder_test1-<rtl or gl>

The output for the previous command should look similar to figure 15 down below

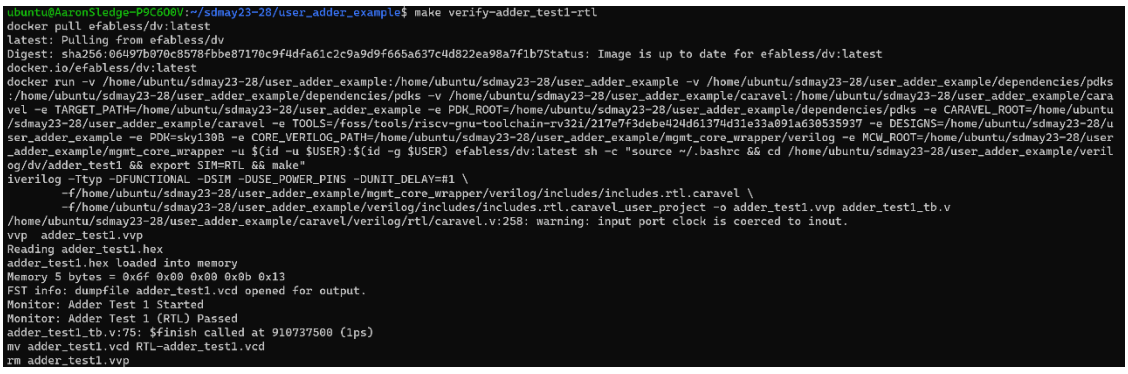


Figure 15. Output of make verify-user_adder_test1

Command: make verify-user_adder2-<rtl or gl>

The output for the previous command should look similar to figure 16 down below

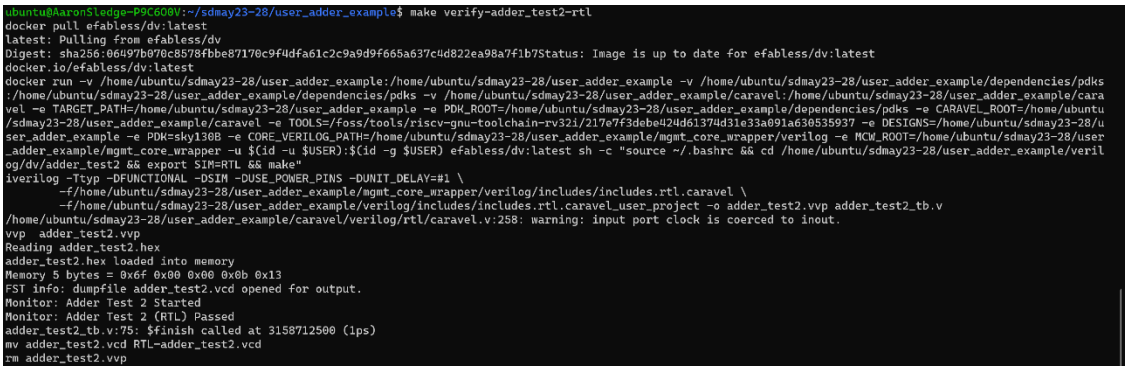


Figure 16. Output of make verify-user_adder_test2

There are a number of other tests that your design will have to pass before submission. However, you will not be able to use the user adder project to run the tests down below. The tests will time out if you try to run them using the user adder project and fail the test.

To run each test individually you will have to enter in one of the following commands:

Command: make verify-io_ports-<rtl or gl>

Command: make verify-mprj_stimulus-<rtl or gl>

Command: make verify-la_test1-<rtl or gl>

Command: make verify-la_test2-<rtl or gl>

Command: make verify-wb_port -<rtl or gl>

4.9 Running final checks on your project before submission

The final thing that you will have to do is to pass all of the prechecks set out by Caravel. There are 13 in total that your design will have to pass before your project is ready for submission to caravel:

1. License check: caravel will check your directory and make sure that there are no prohibited license files within it.

2. Make file: caravel will check your make file in your directory and ensure that it is valid.

3. Default: caravel will check to see if you have edited the 'README.md' file to ensure that you do not maintain the default 'README.md' file that comes with the caravel repository.

4. Documentation: caravel will check to see if there is appropriate documentation in your directory.

5. Consistency: Caravel will check the following.

A) Hierarchy - Module user_project_wrapper is instantiated in caravel.

B) Complexity - Netlist caravel contains at least 8 instances.

C) Modeling - Netlist caravel is structural.

D) Submodule Hooks - All module ports for user_project_wrapper are correctly connected in the top- level netlist caravel.

E) Power Connections - All instances in caravel are connected to power.

F) Ports - Netlist user_project_wrapper ports match the golden wrapper ports.

G) Complexity - Netlist user_project_wrapper contains at least 1 instance.

H) Modeling - Netlist user_project_wrapper is structural.

I) Layout - The GDS layout for user_project_wrapper matches the provided structural netlist.

J) Power Connections - All instances in user_project_wrapper are connected to power.

K) Port Types - Netlist user_project_wrapper port types match the golden wrapper port types.

6. XOR: Checks for total XOR differences if looking for more information on this test check the file in the directory that the pre-check suggests.

7. Magic DRC: Checks for any violations in the design rule check

8. Klayout FEOL: Checks for DRC violations

9. Klayout BEOL: Checks for DRC violations

10. Klayout Offgrid: Checks for DRC violations

11. Klayout Metal Minimum Clear Area Density: Checks for DRC violations

12. Klayout Pin Label Purposes Overlapping Drawing: Checks for DRC violations

13. Klayout ZeroArea: Checks for DRC violations

Each one of these checks will have to be passed before you are able to submit your project to Efabless. In order to execute these tests, enter the following commands into your terminal:

Commands: make precheck (you only need to do this once)

Commands: make run-precheck

If your project passes all of the prechecks the output in your terminal should appear similar to figure 17 below.

```
'
{{SUCCESS}} All Checks Passed !!!
```

Figure 17. Output of make run-percheck

After this step is complete you must make sure that your repository is up to date and that you push all necessary files or edits to previous file to your repo. You now should have a solid base of understanding of the caravel repository, how to opperate it, as well as what will need to be done for your project.

5. Getting started with the Wishbone Bus

5.1 What is the wishbone bus

The wishbone bus is a 32-bit bus connecting the user project wrapper with the management SoC. It includes the following user project wrapper ports:

Port Description	Port name
Clock	wb_clk_i
Reset	wb_rst_i
Strobe	wbs_stb_i
Cycle	wbs_cyc_i
Write enable	wbs_we_i
Select	wbs_sel_i
Data in (to user area)	wbs_dat_i
Data out (to SoC)	wbs_dat_o
Address	wb_adr_i
Acknowledgement	wbs_ack_o

Note: Addresses must be in 4-byte increments

5.2 Using the wishbone bus from the Management SoC

In a caravel testbench, the management SoC can communicate using the wishbone using the following steps:

1. Enable wishbone: reg_wb_enable = 1;
2. Write to bus: <32-bit address> = <32-bit data>; (the user area may respond to any address from 0x3000_0000 to 0x7FFF_FFFF)

0x 2f 00 00 08	IRQ 7 input source (reg_irq7_source)
0x 30 00 00 00	User area base. A user project may define additional Wishbone responder modules starting at this address.
0x 80 00 00 00	QSPI controller

3. Read from bus: uint32_t in_data = <32-bit address>;

5.3 Writing to the user area

The user area can respond to a write request using the following process:

```
// writes
always @(posedge clk) begin
    if (rst) begin
        wbs_ack_o <= 0;
    end else begin
        wbs_ack_o = 1'b0;
        if(valid && !wbs_ack_o && wbs_adr_i == 32'h3000_0000) begin
            wbs_ack_o = 1'b1;
            if (wstrb[0]) lower_bits <= wbs_dat_i[7:0];
            if (wstrb[1]) next_bits <= wbs_dat_i[15:8];
        end
    end
end
```

where 'valid = wbs_cyc_i && wbs_stb_i'.

You should check if the address on the wishbone matches your component's address before reading/writing/acknowledging the data.

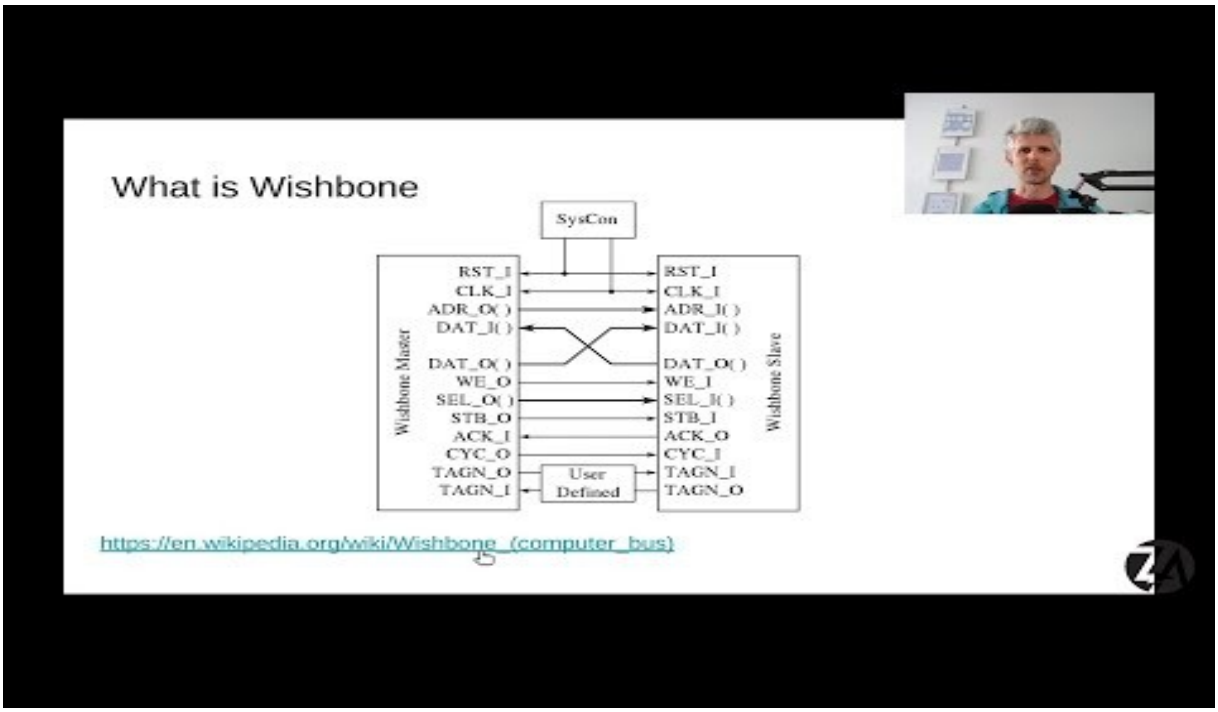
5.4 Reading from the user area

The user area can respond to a read request using the following process:

```
// reads
always @(posedge clk) begin
    if (rst) begin
        wbs_ack_o <= 0;
    end else begin
        wbs_ack_o = 1'b0;
        if(valid && !wbs_ack_o && wbs_adr_i == 32'h3000_0004) begin
            wbs_ack_o = 1'b1;
            wbs_dat_o <= bits;
        end
    end
end
```

The caravel harness github repository includes the user_proj_example, which has some interaction with the wishbone, as well as some tests in the design verification (DV) verilog directory.

Video: [Caravel Wishbone bus demo - YouTube]



Necessary Repositories:

1. [GitHub - mattvenn/wishbone_buttons_leds at caravel]
2. [caravel_user_project/README.md at wishbone-demo · mattvenn/caravel_user_project · GitHub]

6. Getting started with the Logic Analyzer

6.1 What is the logic analyzer

The logic analyzer is a 128-bit port going to/from the user area and management SoC. It allows the SoC to "probe" any desired signals from the user area after fabrication (but they cannot be changed once fabricated, so choose wisely). This is useful for debugging signals during the bring-up process, possibly to discover bugs in the hardware, or to verify that they are working correctly.

6.2 How to use the logic analyzer in the user area

The logic analyzer can be used by connecting the 128 bits to any of the desired signals in your design. This can be seen in the user_proj_example caravel module.

```
// Assuming LA probes [63:32] are for controlling the count register
assign la_write = ~la_oenb[63:32] & ~{BITS{valid}};

always @(posedge clk) begin
    if (reset) begin
        count <= 0;
        ready <= 0;
    end else begin
        ready <= 1'b0;
        if (~|la_write) begin
            count <= count + 1;
        end
        if (valid && !ready) begin
            ready <= 1'b1;
            rdata <= count;
            if (wstrb[0]) count[7:0]   <= wdata[7:0];
            if (wstrb[1]) count[15:8]  <= wdata[15:8];
        end else if (|la_write) begin
            count <= la_write & la_input;
        end
    end
end
```

Here, the logic analyzer is used to set the counter value. It takes the or-reduction of la_write (or'ing all bits together, |x operator) and if it is 1, sets the count to the la_write and'ed with la_input.

6.3 How to use the logic analyzer from the management SoC

For details on how to use the logic analyzer from the SoC, refer to the la_test1 dv directory in the base caravel repository.

```
// Configure LA probes [31:0], [127:64] as inputs to the cpu
    // Configure LA probes [63:32] as outputs from the cpu
    reg_la0_oenb = reg_la0_iena = 0x00000000;    // [31:0]
    reg_la1_oenb = reg_la1_iena = 0xFFFFFFFF;    // [63:32]
    reg_la2_oenb = reg_la2_iena = 0x00000000;    // [95:64]
    reg_la3_oenb = reg_la3_iena = 0x00000000;    // [127:96]

    // Flag start of the test
    reg_mprj_datal = 0xAB400000;

    // Set Counter value to zero through LA probes [63:32]
    reg_la1_data = 0x00000000;
```

First, the logic analyzer probes are configured to be inputs or outputs, then the data is written (or read) from the data register.

7. How to use interrupts

7.1 What is an interrupt

An interrupt is a signal from the hardware that pauses the current processor execution and begins execution somewhere else. This is commonly used to avoid blocking code, where the CPU continuously polls the hardware for data. Instead, the hardware can alert the CPU when a condition is met, such as a timer reaching a specified value.

7.2 How to trigger an interrupt from the user area

An interrupt can be triggered from the 3 irq bits available to the user area. Each of these bits corresponds to an interrupt. The interrupt should be high for one clock cycle, and low otherwise.

7.3 How to receive an interrupt in the SoC

Interrupts can be enabled in the SoC in the following way:

```c
#include <defs.h>
#include <stub.c>
#include <irq_vex.h>

extern uint16_t flag;

void main()
{
    irq_setmask(0);
    irq_setie(1);
    flag = 0;

    irq_setmask(irq_getmask() | (1 << USER_IRQ_0_INTERRUPT));
    // irq_setmask(irq_getmask() | (1 << TIMER0_INTERRUPT));

    // Configure GPIO upper bits to assert the test code
    reg_mprj_io_35 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_34 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_33 = GPIO_MODE_MGMT_STD_OUTPUT;
    reg_mprj_io_32 = GPIO_MODE_MGMT_STD_OUTPUT;

    // Apply the GPIO configuration
    reg_mprj_xfer = 1;
    while (reg_mprj_xfer == 1);

    reg_mprj_irq = 0b001;

    reg_mprj_datah = 0x5;       // Signal start of test
    reg_mprj_datal = 0;

    reg_user_irq_enable = 1;
    reg_user0_irq_en = 1;
```

The irq_vex.h header adds the functions/macros for irq_setmask, irq_setie, etc. The code starts by setting the mask to 0 and enabling interrupt 1. The external flag variable is the variable used by the interrupt handler. This handler is part of the existing simulation infrastructure, so it is not necessary to write it. The interrupt handler must be placed at a specific location in the SoC's program memory, so it is simpler to use the example provided by the harness repository by default. The second irq_setmask() sets the correct bit for the irq we are triggering in hardware. In this case, it is the 0th interrupt (irq bit 0 in the user area). Next, the req_user_irq_enable register should have the corresponding bit set for the interrupt you are using. Finally, the req_userX_irq_en must be set to 1 (where X is the irq index). In simulation, the flag variable will be set to 1 when the interrupt is triggered.

8. How to use klayout to view GDS files

Klayout can be a very useful tool when it comes to seeing the cells that you have added to the user project wrapper. So, in this section of the tutorial document, you will be instructed on how to view the project wrappers as well as how to interpret the cells. Klayout is only useful after you have hardened your design inside of the user_project_wrapper. The first thing you need to do is to enter the directory of your project where you hardened your design. Then type in the following command:

Command: find –name '*klayout.gds'

This command will find all of the files within the directory you are in that contain the "klayout.gds" attachment. The output will look similar but not the same as figure 18 down below.

```
ubuntu@AaronSledge-P9C6O0V:~/sdmay23-28/user_adder_example/openlane$ find -name '*klayout.gds'
./user_project_wrapper/runs/23_02_11_11_48/results/signoff/user_project_wrapper.klayout.gds
./user_project_wrapper/runs/23_02_23_15_26/results/signoff/user_project_wrapper.klayout.gds
./user_project_wrapper/runs/23_02_16_15_05/results/signoff/user_project_wrapper.klayout.gds
./user_project_wrapper/runs/23_02_23_14_53/results/signoff/user_project_wrapper.klayout.gds
./user_adder/runs/23_02_23_14_41/results/signoff/user_adder.klayout.gds
./user_adder/runs/23_02_16_14_24/results/signoff/user_adder.klayout.gds
./user_adder/runs/23_02_11_11_38/results/signoff/user_adder.klayout.gds
./user_adder/runs/23_02_23_15_16/results/signoff/user_adder.klayout.gds
./user_adder/runs/23_02_11_13_21/results/signoff/user_adder.klayout.gds
./user_adder/runs/23_02_11_11_24/results/signoff/user_adder.klayout.gds
./user_adder/runs/22_11_15_14_51/results/signoff/user_adder.klayout.gds
./user_adder/runs/23_02_16_13_01/results/signoff/user_adder.klayout.gds
```

Figure 18. Finding all klayout gds files

As can be seen from figure '#' above you are able to look at just the user area or the user project wrapper as a whole. The file names starting with ***./user_project_wrapper*** are the GDS files that contain your project in the user area inside of the user project wrapper. The file names that start with ***./user_adder*** are the GDS files that will show the user area with the user adder inside of it. For either one of the two the way to tell which file is the most recent is by looking at the numbers in the files. Reference figure 19 below for which numbers to look at.

Figure 19. Display of date and time GDS file was created

The meaning behind what each number represents is listed below:

1. 23 – stands for the year the GDS file was made
2. 02 – stands for the month the GDS file was made
3. 16 – stands for the day the GDS file was made
4. 15 – stands for the hour the GDS file was made in military time

5. 05 – stands for the minute the GDS file was made

Highlight and copy the most recent version of the file you made, or whichever version you want to view inside of klayout. Use the following command and paste the file name that you previously copied and want to view it inside of klayout.

Command: klayout < Name of GDS File >

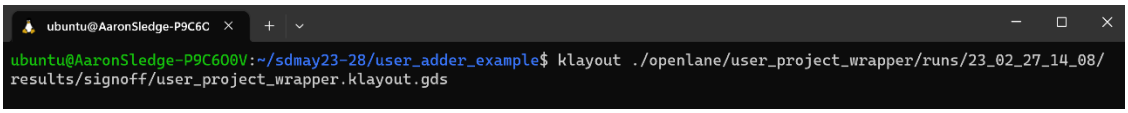


Figure 20. Display of Klayout command with desired file

Once you enter the command a window should pop up with the layout of the caravel harness. See figure 21 below for reference.

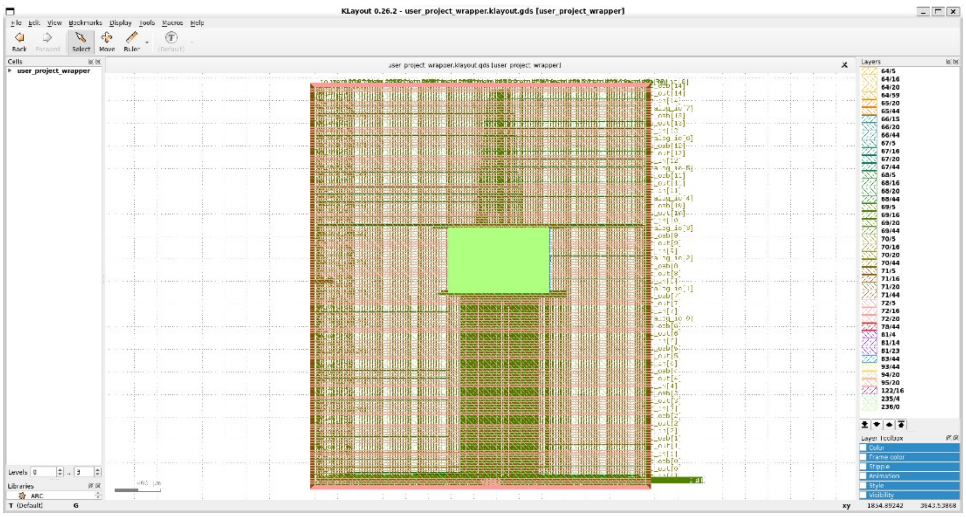


Figure 21. Display of user project wrapper in klayout

This is the view of the unlabeled layout of the caravel harness. If you would instead like for a less cluttered version of the layout with labels, then you should clone the following repository into your (Insert directory name) directory. Once that is done your layout for the caravel harness will look more like figure 22 down below.

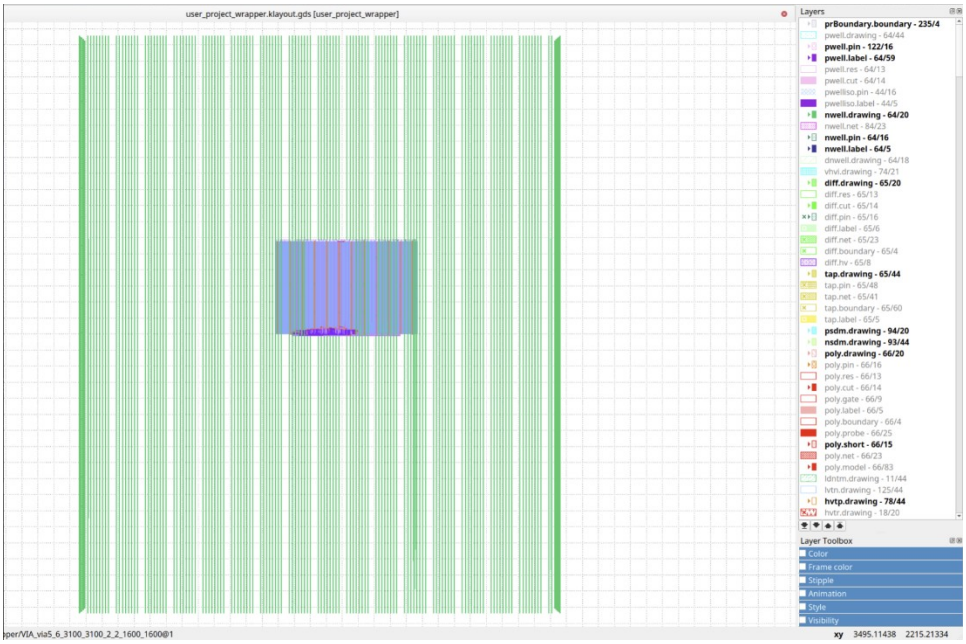


Figure 22. Display of filtered kaylout user project wrapper

9. How to submit your project design to Efabless

Source: MPW-6 Walkthrough - YouTube - 26 min till the end

After your project design has passed all necessary tests and checks in your environment the next step is to submit your design to Efabless. You will want to head to the following link and look for the next available MPW shuttle submission.

Link: Efabless

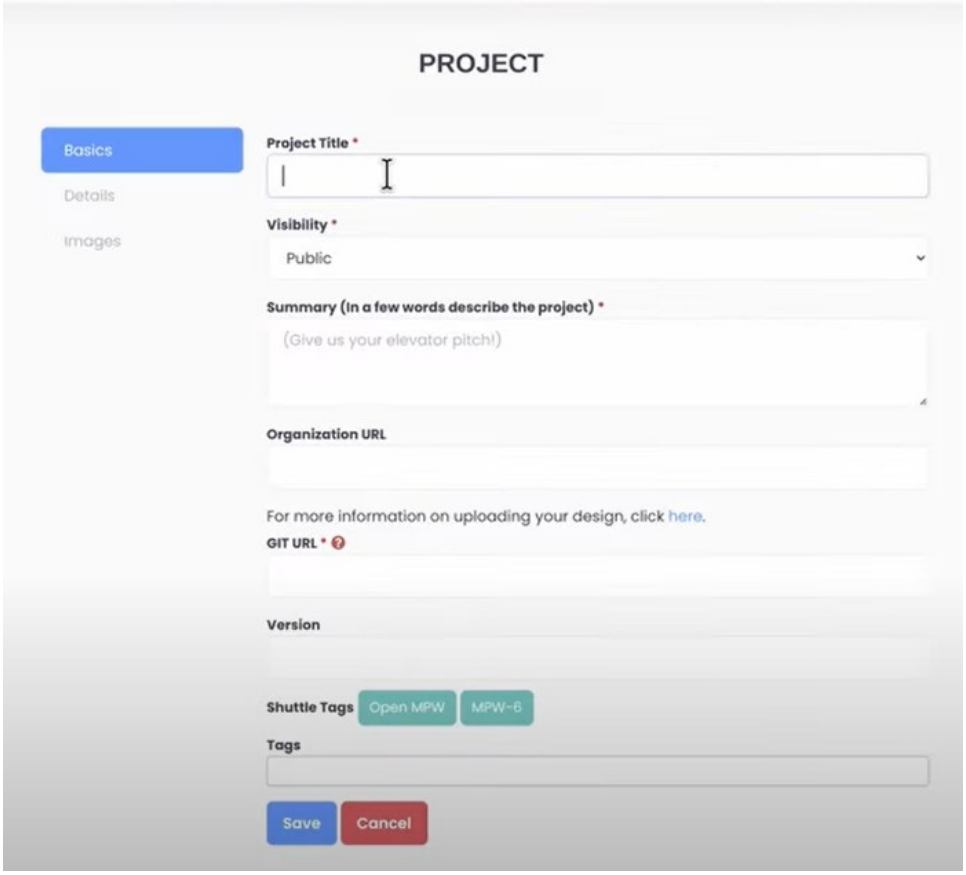


Figure 23. Efabless MPW submission page

You will need to fill out the form displayed in figure (#), the only thing that you have to make sure to do is that when you copy and paste the URL to your git repository in the 'GIT URL' is to add the '.git' at the end of it. The 'Version' & 'Tags' section do not need to be filled out for your first submission. Once done hit save, then you will be taken to the following page displayed in figure 24 below.

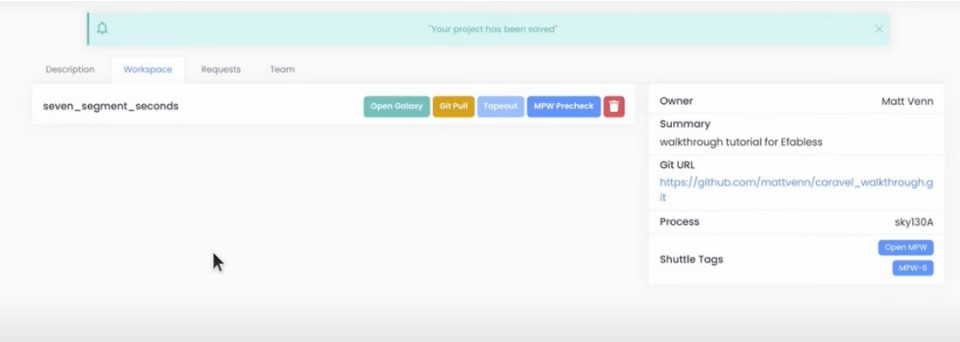


Figure 24. Efabless MPW workspace on submission page

Be sure to Navigate to the Workspace section in this page. Then you will need to click the 'Git Pull' button. The time it takes for the submission page to pull your repository will vary depending on how many people are trying to do the same thing at the same time. But once the page has successfully pulled your repository the page will notify you that it has successfully pulled your repository.

Once that is finished the next step is to click on the MPW Precheck button and give it a 'Job Name' any name will do. It will take roughly 5 minutes for the test to complete but once it does and it succeeds your workspace window should appear like figure 25 below.

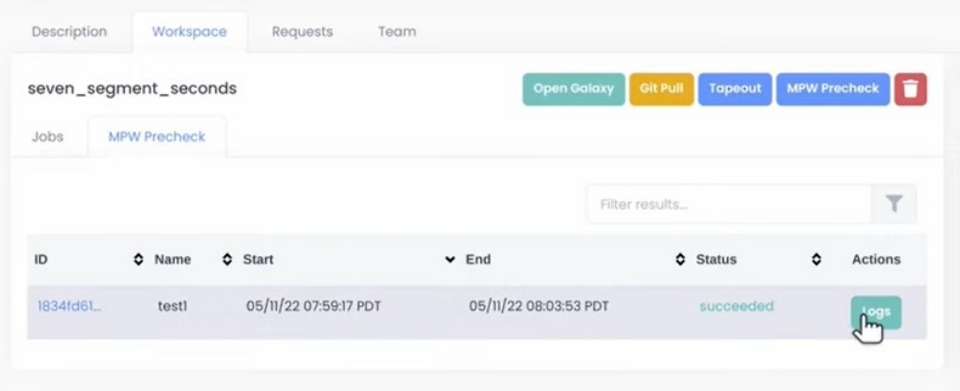


Figure 25. Image of Successful MPW Precheck

After you have successfully passed the 'MPW Precheck' the last thing that you will have to do is pass the Tapeout. Click on the 'Tapeout' button and the following pop-up window will appear as displayed in figure 26 below.

Figure 26. Image of Tapeout pop up window

You will then need to select the ‘MPW Precheck’ job name that you created and enter in a new Job name for the Tapeout then click ‘Submit Job’. Now this test will take somewhere around an hour to complete. But once the Tapeout has finished you have succesfully submitted your design to Efabless for the MPW submission.

10. Caravel PCB daughter & carrier boards you receive from Efabless

10.1. Introduction

The efabless caravel_board is an open-source development platform for designing and testing custom System on Chips (SoCs). It is designed to be flexible, customizable, and accessible to a wide range of users, from hobbyists to professional designers. The caravel_board is built on top of the efabless OpenLANE tool flow, which provides a complete open-source RTL to GDSII flow for chip design.

10.2.0. Overview of caravel_board

The caravel_board is a small form factor board that measures 85mm x 54mm. It consists of several components that are organized into logical sections on the board. These sections include the power management section, the FPGA section, the GPIO section, and the JTAG section.

10.2.1. Power management section

The power management section is responsible for providing power to the various components on the board. It consists of a 3.3V regulator, a 1.8V regulator, and a power-on reset circuit. The regulators ensure that the various components on the board receive the correct voltage levels, while the power-on reset circuit ensures that the board is properly initialized when power is applied.

10.2.2. FPGA section

The FPGA section is the heart of the caravel_board. It consists of an FPGA chip and several components that support its operation. The FPGA chip is a Lattice iCE40HX8K FPGA, which provides 7680 logic cells, 128Kbits of embedded RAM, and 4 PLLs. The FPGA is connected to the rest of the board through several interfaces, including SPI, I2C, and GPIO.

10.2.3. GPIO section

The GPIO section provides a set of general-purpose input/output (GPIO) pins that can be used to interface with external devices. The GPIO pins are connected to the FPGA and can be programmed to perform various functions, such as controlling LEDs, reading buttons, or communicating with other devices.

10.2.4. JTAG section

The JTAG section provides a JTAG interface for programming the FPGA and debugging the board. The JTAG interface is connected to the FPGA and can be used to program the FPGA with new firmware, or to test and debug custom logic on the board.

10.3.0. Overview of caravel_Nucleo

The caravel_Nucleo is an accessory board that is used in conjunction with the caravel_board. It provides additional functionality that is necessary for programming and testing the caravel_board. The caravel_Nucleo board consists of several components, including an ST-Link debugger, a USB interface, and several buttons and LEDs.

10.3.1. ST-Link debugger

The ST-Link debugger is used to program the caravel_board with new firmware, and to debug custom logic on the board. It is connected to the JTAG interface on the caravel_board and communicates with the board through the USB interface.

10.3.2. USB interface

The USB interface provides a way to communicate with the caravel_board and to transfer data to and from the board. It is connected to the ST-Link debugger and can be used to program the board with new firmware, or to read and write data to the board.

10.3.3. Buttons and LEDs

The buttons and LEDs on the caravel_Nucleo board are used for testing and debugging the caravel_board. The buttons can be used to trigger various events on the board, while the LEDs can be used to indicate the status of various components on the board.

10.4.0 How the parts on the board connect to each other?

The various parts on the caravel_board and caravel_Nucleo are connected to each other through a series of connectors and traces on the board.

10.4.1. FPGA connections

The FPGA on the caravel_board is connected to several components on the board through a series of connectors and traces. The FPGA is connected to the power management section, which provides the necessary power to the FPGA. It is also connected to the GPIO section, which provides a set of GPIO pins that can be used to interface with external devices. The FPGA is also connected to the JTAG section, which provides a JTAG interface for programming the FPGA and debugging the board.

10.4.2. Power management connections

The power management section on the caravel_board is connected to the various components on the board through a series of connectors and traces. The 3.3V regulator is connected to the FPGA and the GPIO section, while the 1.8V regulator is connected to the FPGA. The power-on reset circuit is connected to the FPGA and the JTAG section.

10.4.3. GPIO connections

The GPIO section on the caravel_board is connected to the FPGA through a series of connectors and traces. The GPIO pins are used to interface with external devices and can be programmed to perform various functions, such as controlling LEDs, reading buttons, or communicating with other devices.

10.4.4. JTAG connections

The JTAG section on the caravel_board is connected to the FPGA through a series of connectors and traces. The JTAG interface is used to program the FPGA with new firmware, or to test and debug custom logic on the board.

10.4.5. ST-Link connections

The ST-Link debugger on the caravel_Nucleo board is connected to the JTAG interface on the caravel_board through a series of connectors and traces. The ST-Link communicates with the caravel_board through the USB interface, which is also connected to the caravel_Nucleo board.

10.4.6. Button and LED connections

The buttons and LEDs on the caravel_Nucleo board are connected to the caravel_board through a series of connectors and traces. The buttons can be used to trigger various

events on the board, while the LEDs can be used to indicate the status of various components on the board.

10.5. How the caravel_board connects to the user area and how to test it?

The caravel_board can be used to design and test custom SoCs. The user area is where the custom logic is implemented and tested. The user area is a part of the FPGA chip and is programmed with custom firmware using the JTAG interface and the ST-Link debugger.

To connect to the user-area, the user can use the available pads on the caravel_board. The user can also add additional pads if necessary. The user can then use the open-source tools provided by efabless to design and test custom logic for the user-area.

To test the user area,the user can write custom Verilog code to implement the desired functionality and then use the open-source tools provided by efabless to synthesize, place, and route the design. Once the design is complete, it can be programmed onto the caravel_board using the ST-Link debugger and the JTAG interface.

Furthermore, to test the user area, the user can then interact with the custom logic using the GPIO pins on the caravel_board. The GPIO pins can be programmed to perform various functions, such as controlling LEDs, reading buttons, or communicating with other devices. The user can also use the JTAG interface and the ST-Link debugger to debug the custom logic and to monitor its behavior.

In short, to test the user-area, the user can use the following steps:

1. Design custom logic using the open-source tools provided by efabless, such as OpenLANE RTL to GDSII flow.
2. Synthesize the design using the open-source Yosys synthesis tool.
3. Place and route the design using the OpenLANE tool.
4. Generate the GDSII layout file for the design.
5. Create a new firmware for the caravel_board that includes the custom logic.
6. Program the caravel_board with the new firmware using the JTAG interface on the caravel_Nucleo board.
7. Test the custom logic on the caravel_board using the appropriate testbench or software

The open-source tools and documentation provided by efabless make it easy for users to customize and extend the caravel_board for their specific needs. The caravel_Nucleo

board provides the necessary interfaces for programming and testing the caravel_board, and the power management circuitry ensures that the board is powered properly during testing.

10.6. Conclusion

In conclusion, the efabless caravel_board is an open-source platform that provides an excellent opportunity for learning and experimenting with ASIC/SOC development. It includes all the essential components required for a complete system-on-a-chip design, such as the FPGA, RAM, flash memory, voltage regulators, and various input/output interfaces. The board's open-source nature enables the community to contribute and improve the design, making it accessible and affordable for hobbyists and professionals alike.

The caravel_board design flow uses open-source tools such as the Yosys synthesis tool and the OpenLANE flow for design automation. It simplifies the design process, enabling users to customize the platform to their needs and interests. With caravel_board, users can create custom designs and test them in a real-world environment without requiring significant financial in gnificant financial investment.

At last, the efabless caravel_board is a valuable platform for anyone interested in learning or experimenting with ASIC/SOC development. Its open-source nature, affordability, and community support make it an ideal choice for hobbyists and professionals alike. It provides an excellent opportunity for users to experiment with custom designs and develop their skills in FPGA development.

10.7. Full form of all abbreviations

ASIC - Application-Specific Integrated Circuit
SOC - System-on-a-Chip
FPGA - Field-Programmable Gate Array
PDK - Process Design Kit
RAM - Random Access Memory
HDL - Hardware Description Language
I/O - Input/Output
PCB - Printed Circuit Board
USB - Universal Serial Bus
JTAG -Joint Test Action Group

GPIO - General Purpose Input/Output
GDSII -short for Graphic Data System II
LED -Light Emitting Diode
OpenLANE - Open-Source Digital ASIC Implementation Flow
RTL -Register Transfer Level
RISC-V - Reduced Instruction Set Computing - Five

11. References

1 Caravel Board on GitHub. https://github.com/efabless/caravel_board
2 Caravel Documentation. https://caravel-design.readthedocs.io/en/latest/
3 Introduction to the efabless Caravel Platform. https://www.youtube.com/watch?v=xj9KYWz1H7k
4 efabless Caravel Board: The Best Place to Start Learning About Chip Design. https://www.electronicsforu.com/electronics-projects/hardware-diy/efabless-caravel-board-best-place-start-learning-chip-design
5 The Caravel Open Source ASIC Design Flow. https://www.allaboutcircuits.com/technical-articles/the-caravel-open-source-asic-design-flow/
6 An Introduction to ASIC Development with Caravel. https://www.hackster.io/news/an-introduction-to-asic-development-with-caravel-396a20c8b33a
7 Caravel: An Open Source PDK and ASIC/SOC Development Platform. https://efabless.com/news/2019/8/7/caravel-an-open-source-pdk-and-asicsoc-development-platform
8 Design Your Own Chip: The efabless Open-Source Caravel Platform. https://www.iotforall.com/design-your-own-chip-the-efabless-open-source-caravel-platform/
9 ASICs and FPGAs and SoCs, Oh My! A Guide to Silicon on Chip. https://www.eetimes.com/asics-and-fpgas-and-socs-oh-my-a-guide-to-silicon-on-chip/
10 Open Source FPGA Toolchain. https://en.wikipedia.org/wiki/Open_Source_FPGA_Toolchain