

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga-SoC Lab 3**

**Introduction to SweRVolf and FuseSoC**

# Introduction :

In this Lab, we will analyze SweRVolf in detail. We will also introduce FuseSoC and show how to use it to build SweRVolf.

In Lab 1, individual modules were written in Verilog/SystemVerilog. We then connected these modules to create RVfpga-SoC using Vivado’s Block Design Tool. In industry, designers use either visual methods of connecting modules or strictly Verilog/SystemVerilog code.

The following table provides a brief overview of the advantages and disadvantages of the Block Design approach.

|  |  |
| --- | --- |
| **Advantages of Block Design** | **Disadvantages of Block Design** |
| Easy to understand | Teamwork can be challenging |
| Learning curve | Ecosystem is required(Vivado Block Design) |
| Can combine complex modules in an intuitive way | Upgrading tools can break the system |
|  | May struggle to go beyond FPGAs into real SoC chips |
|  | Machine-generated code can be difficult to read and understand. |
|  | Fine-tuning the SoC for fabrication or to squeeze out more performance must be done at the Verilog/SystemVerilog level. |
|  | IP core providers provide Verilog source code but not necessarily in a form that will work with a specific Vivado Block Design tool. |

The following table provides a brief overview of the advantages and disadvantages of the Hand-tuned approach.

|  |  |
| --- | --- |
| **Advantages of Hand-tuned code** | **Disadvantages of Hand tuned code** |
| Can hand tune it to be very specific to an FPGA or an SoC fabrication process. | Steep learning curve |
| Can use version control systems | Requires knowledge of Verilog/SystemVerilog. |
| Can easily work in teams and collaborate (various people work on different modules) | It takes a while to familiarize oneself with how all the files/folders/modules are structured together. |

SweRVolf's modules (top module, swervolf\_core, SweRV) have been written by hand. The SweRV CPU core itself was created by Western Digital. The interconnect and other modules are IP cores from various vendors who have released them under open-source licenses. The top modules have been written by Olof Kindgren.

It is possible to connect all different modules and write Verilog by hand. But over time, this can become tedious. Especially as larger teams work together sharing some parts of the code.

During the SoC design process, several large teams are working together simultaneously. Different teams will be focused on different areas, but they also share various components and IP blocks. For example, the simulation team will be concerned with the same IP code (SoC+interconnect+CPU+Peripheral) but with wrappers for targeting it to the Instruction set simulator or Verilator. Meanwhile, the FPGA team would like to use the same IP code (SoC+Interconnect+CPU+Peripheral) but run their tests on various development boards with different peripherals.

A build system facilitates different teams working on the same IP code (SoC+Interconnect+CPU+Peripheral) but not step on each others' toes. The build system must be flexible and dynamic enough to suit various teams and their various constraints/requirements.

SweRVolf has been created using such a build system called FuseSoC (pronounced “fuse sock”). The FuseSoC build system pieces together a hardware design from individual building blocks.

In this Lab, we will be looking at building versions of SweRVolf within FuseSoC. We will show the step-by-step procedure of adding FuseSoC’s “SweRVolf” library and then targeting it for both simulation and board implementation. Then we will run example programs on SweRVolf.

# Requirements :

To complete this lab, you will need to install the following tools:

* Vivado 2019.2 Web Pack (Refer to Installation Guide (Page No.04))
* Verilator (V4.106) (Refer to Installation Guide (Page No.13))
* GTKWave (Refer to Installation Guide (Page No.13))
* FuseSoC (Refer to Installation Guide (Page No.14))
* OpenOCD (RISC-V-specific version) (Refer to Installation Guide (Page No.14))

**IMPORTANT:** Before starting RVfpga-SoC Labs, we highly recommend completing the RVfpga-SoC Installation Guide.

For example, if you have not already, install Verilator and Xilinx’s Vivado by following the instructions in the RVfpga-SoC Installation Guide. Make sure that you have copied the RVfpga-SoC folder that you downloaded from Imagination’s University Programme to your machine.

# What is FuseSoC :

FuseSoC is an award-winning package manager and a set of build tools for HDL (Hardware Description Language) code. Its primary purpose is to increase the reuse of IP (Intellectual Property) cores and create, build, and simulate SoC solutions.

A fundamental entity in FuseSoC is a core. Cores can be discovered by the FuseSoC package manager in local or remote locations and combined into a full hardware design by the build system.

A FuseSoC core is not necessarily a processor but instead is a reasonably self-contained, reusable piece of IP, such as a FIFO implementation. FuseSoc also refers to these as packages. In other systems, these reusable pieces of hardware are called modules.

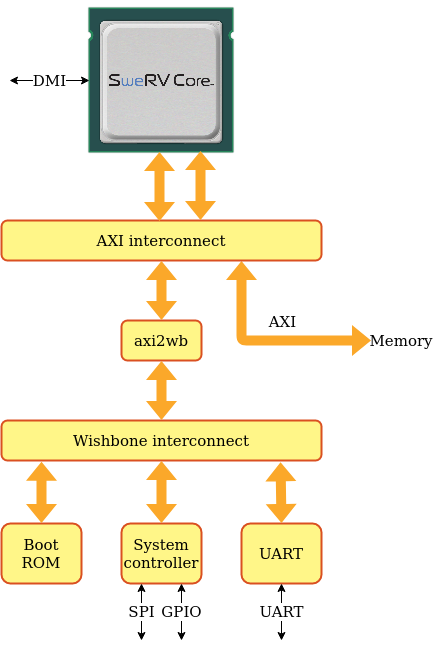
For more detailed information on FuseSoC, you can read its documentation at <https://fusesoc.readthedocs.io/en/latest/user/overview.html#understanding-fusesoc>

FuseSoC provides the “SweRVolf” (FuseSoC-based SoC for the SweRV RISC-V core) as a package in its library. We can add the “**swervolf**” library using FuseSoC and then build it for any specific target such as “sim” (Simulation) or “Nexys\_a7” (Board).

# SweRVolf : FuseSoC based SoC for SweRV Eh1

SweRVolf is a FuseSoC-based SoC for the SweRV RISC-V core. SweRVolf can run the RISC-V compliance tests, Zephyr OS, or other software in simulators or on FPGA boards. FuseSoC aims to support and increase portability, extendability, and ease of use; to allow SweRV users to get the software running quickly, modify the SoC to their needs or port it to new target devices.

Figure 1 shows a high-level block diagram of SweRVolf.



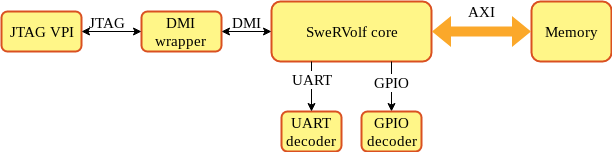
**Figure 1. SweRVolf Core**

The core of SweRVolf consists of a SweRV CPU with a boot ROM, AXI4 interconnect, UART, SPI, RISC-V timer, and GPIO. The core does not include any RAM but instead exposes a memory bus that the target-specific wrapper will connect to an appropriate memory controller. Other external connections are clock, reset, UART, GPIO, SPI, and DMI (Debug Module Interface).

Like the SystemVerilog modules RVfpga SIM and RVfpga Nexys in Lab 2, SweRVolf also provides the equivalent two versions of SweRVolf: SweRVolf SIM for simulation and SweRVolf Nexys for Digilent Nexys A7 board.

1. **SweRVolf SIM**

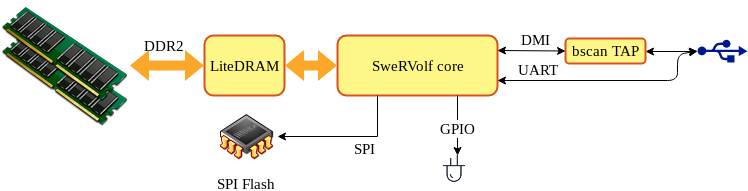
SweRVolf sim is a simulation target that wraps the SweRVolf core in a testbench to be used by verilator or other event-driven simulators such as QuestaSim. It can be used for full-system simulations that execute programs running on a SweRV processor. It also supports connecting a debugger through OpenOCD and JTAG VPI. The Debugging chapter contains more information on how to connect a debugger.



**Figure 7. SweRVolf SIM**

1. **SweRVolf Nexys**

SweRVolf Nexys is a version of the SweRVolf SoC targeted to the Digilent Nexys A7 board. It uses the on-board 128MB DDR2 for RAM, has GPIO connected to the LED, supports booting from SPI Flash, and uses the micro-USB port for UART and JTAG communication. The default bootloader for the SweRVolf Nexys target will attempt to load a program stored in SPI Flash by default.



**Figure 9. SweRVolf Nexys**

The SweRVolf SoC can be run in simulation or on hardware – that is, on the Digilent Nexys A7 board. In either case, FuseSoC can be used to either launch the simulation or build and run the FPGA build.

# Setting up the Environment

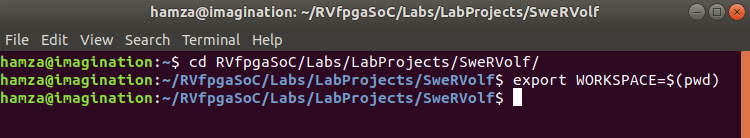
In this section, we show how to set up the environment for the SIM build and Nexys build.

**Step 1.** Navigate to the directory named “**SweRVolf**” to use as the root of the project.

This directory will from now on be called **$WORKSPACE*.*** All further commands will be run from $WORKSPACE unless otherwise stated. After entering the workspace directory, run.

**$** *export WORKSPACE=$(pwd)*

To set the $WORKSPACE shell variable (see Figure 2).



**Figure 2. Establish Workspace**

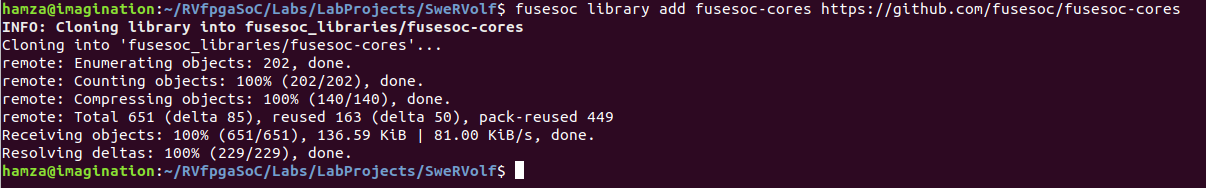
**Step 2.** Make sure you have FuseSoC installed already. If you do not have FuseSoC

installed on your machine, install it by running the following:

**$** sudo pip3 install fusesoc

**Step 3.** Add the FuseSoC base library to the workspace:

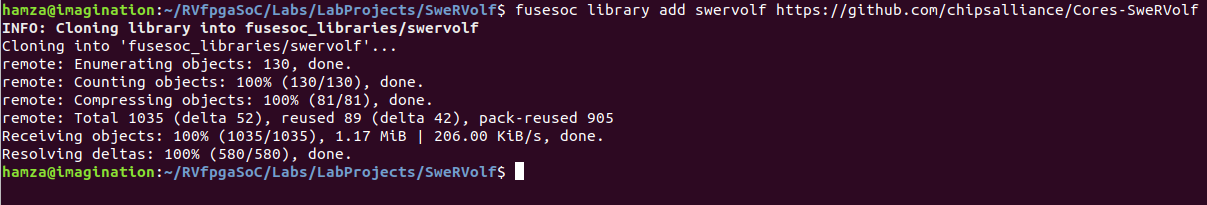
**$** fusesoc library add fusesoc-cores https://github.com/fusesoc/fusesoc-cores



**Figure 3. Add FuseSoC base library**

**Step 4.** Add the swervolf library:

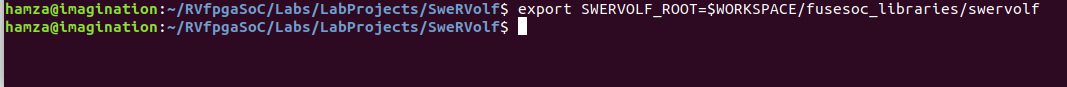
**$** fusesoc library add swervolf https://github.com/chipsalliance/Cores-SweRVolf

**

**Figure 4. Add Swervolf library**

**Step 5.** Set the swervolf directory as the “SWERVOLF\_ROOT” shell variable:

**$** export SWERVOLF\_ROOT=$WORKSPACE/fusesoc\_libraries/swervolf



**Figure 5. Set shell variable**

# Simulation Build

In this section, we show how to build SweRVolf SIM using FuseSoC using the following steps.

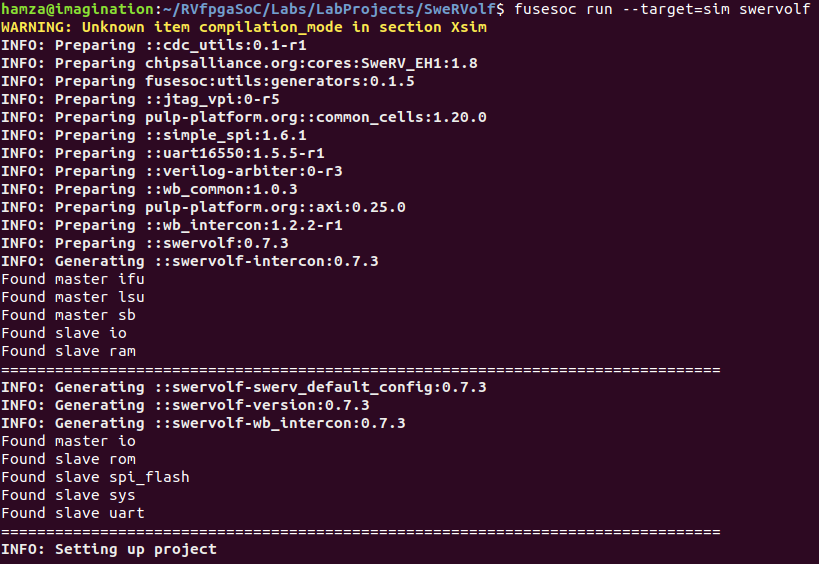
To select what to run, use the “**fusesoc** **run**” command with the --target parameter.

**Step 6.** To run in simulation use

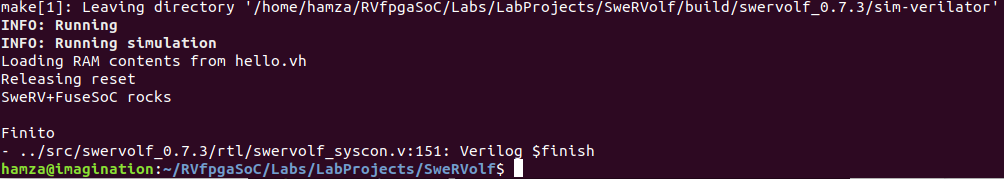
**$** fusesoc run --target=sim swervolf

This command generates the simulation binary called “**Vswervolf\_core\_tb**” inside the following path: *SweRVolf/build/swervolf\_0.7.3/sim-verilator/*

Note that this is similar to what we did in Lab 2 when we generated the “**Vrvfpgasim**” simulation binary with the make command in the following path: *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM*



**………**



**Figure 6. SweRVolf Sim Build**

This command will create the following hierarchy in our Workspace directory :

* $WORKSPACE
  + fusesoc\_libraries
  + Build
    - swervolf\_0.7.3
      * sim-verilator
        + **Vswervolf\_core\_tb**
      * src

# Nexys A7 Build

In this section, we show how to build SweRVolf Nexys using FuseSoC.

**Step 7.** To build (and optionally program) an image for a Nexys A7 board, run

**$** fusesoc run --target=nexys\_a7 swervolf

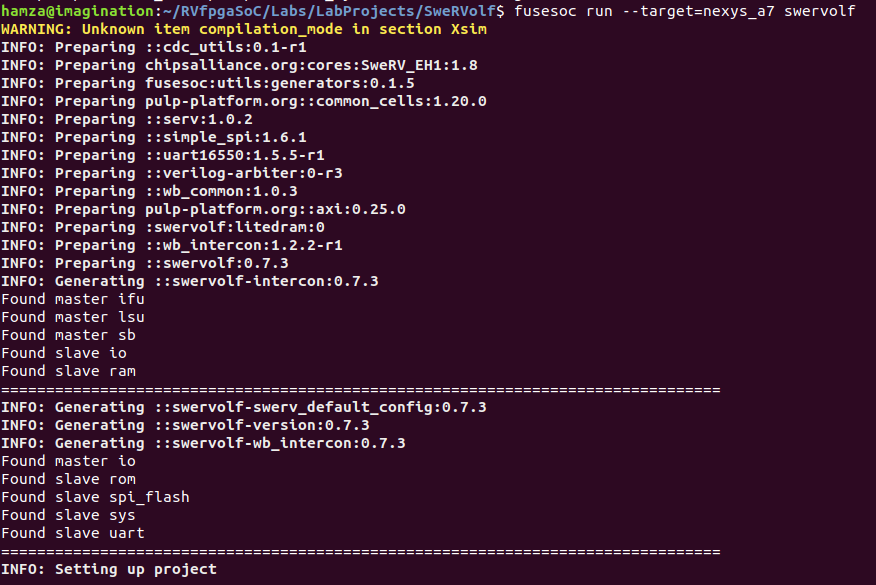
**Note**: Be sure to connect the Nexys A7 board to your machine before running this command and turn the board ON.

This command generates the bitstream file used to program the FPGA (ending in “.bit”) called “**swervolf\_0.7.3.bit**” inside the following path: *SweRVolf/build/swervolf\_0.7.3/nexys\_a7-vivado/*

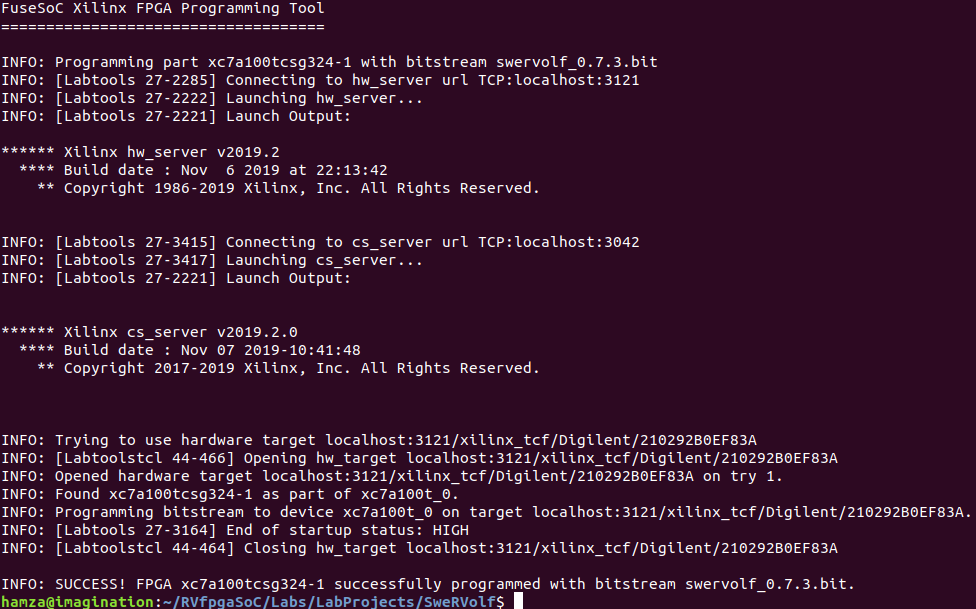
It also uploads the bitstream to the Nexys A7 Board connected to our machine.

This is similar to what we did in Lab 1, when we generated the “**rvfpga.bit**” file in Vivado after creating the block design using the “Generate Bitstream” option. Later in Lab 2, we uploaded the “**rvfpga.bit**” file to our Nexys A7 board using PlatformIO.

**FuseSoC does all of this tedious work that we manually did in Lab 1 and Lab 2 with the command mentioned above.**



……….



**Figure 8. SweRVolf Nexys build**

After the completion of the commands as mentioned earlier, our Workspace directory will look like as follows:

* $WORKSPACE
  + fusesoc\_libraries
  + Build
    - swervolf\_0.7.3
      * sim-verilator
      * nexys\_a7-vivado
        + **swervolf\_0.7.3.bit**
      * src

FuseSoC uses Vivado and creates the project, sets global variables, adds constraint files, and generates the bitstream automatically (See Figure 12).

We can open the Vivado project to visualize the hierarchy as we saw in Lab 1.

Enter the “**nexys\_a7-vivado**” directory by the following command:

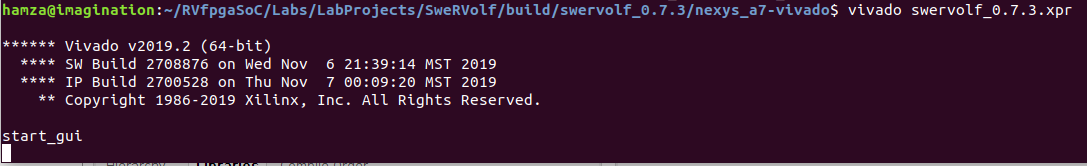
**$** cd build/swervolf\_0.7.3/nexys\_a7-vivado/

****

**Figure 10. Navigate to “nexys\_17-vivado” Directory**

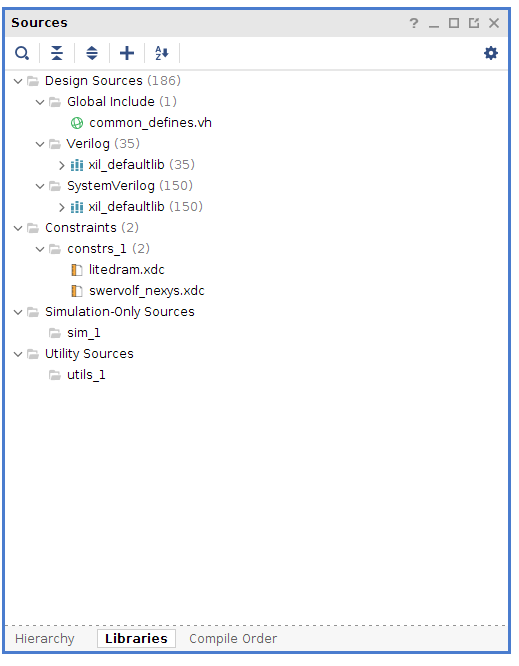
Enter the following command to open the Vivado Project:

**$** vivado swervolf.0.7.3.xpr

****

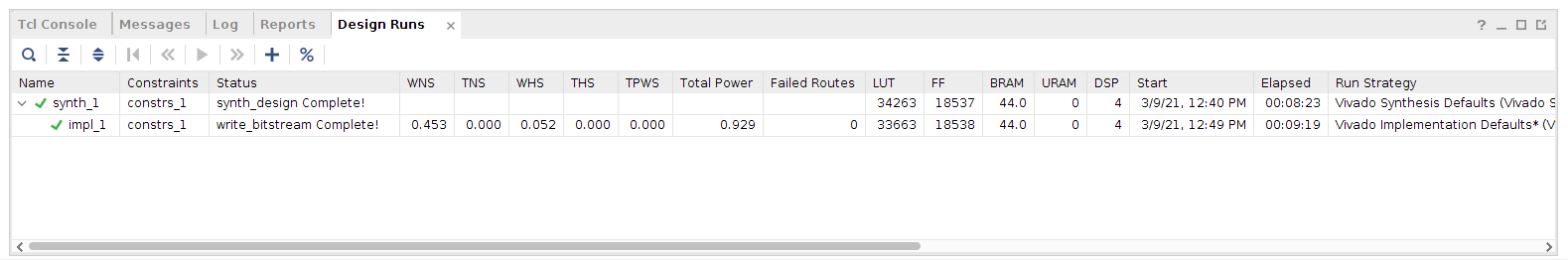
**Figure 11. Open Vivado Project**

We can visualize the global variables and constraints files set in the sources panel.

****

**Figure 12. Sources panel**

In the “Design Runs” tab, we can see that the synthesis and implementation have already been completed successfully.

****

**Figure 13. Design runs**

# Running AL\_Operations Example on SweRVolf SIM:

In this section, we show how to run the *AL\_Operations* program on SweRVolf-SIM using Verilator.

**Step 1**. Enter the examples directory whose path is:

*[RvfpgaSoCPath]/RvfpgaSoC/Labs/LabResources/Lab3/examples/*

*AL\_Operations/CommandLine/*

**$** cd ../../LabResources/Lab3/examples/AL\_Operations/commandLine/



**Figure 14. AL\_Operations directory**

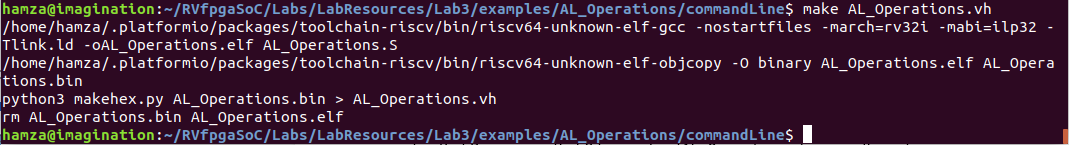
**Step 2**. Create the hexadecimal program for simulation:

**$** make clean

****

**Figure 15. make clean**

**$** make AL\_Operations.vh

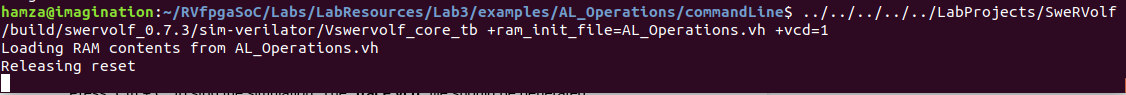
****

**Figure 16. create AL\_Operations.vh file**

**Step 3**. Execute the simulator:

**$** ../../../../../LabProjects/SweRVolf/build/swervolf\_0.7.3/sim-verilator/Vswervolf\_core\_tb +ram\_init\_file=AL\_Operations.vh +vcd=1

The “***ram\_init\_file***” parameter loads a Verilog hex file to use as initial on-chip RAM contents. So we load the Verilog hex file “**AL\_Operations.vh**” that we have just created using that option (+ram\_init\_file) and execute the simulation binary “**Vswervolf\_core\_tb**”. We also set the vcd (value change dump) as 1 to dump all variables within the module.

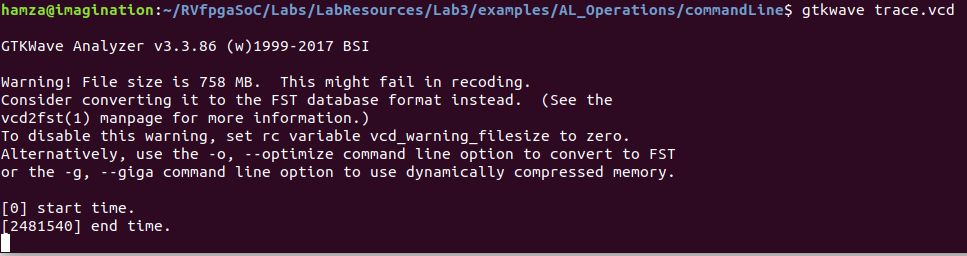
****

**Figure 17. Execute the simulator**

Press “Ctrl + c” to stop the simulation. The “**trace.vcd**” file should be generated.

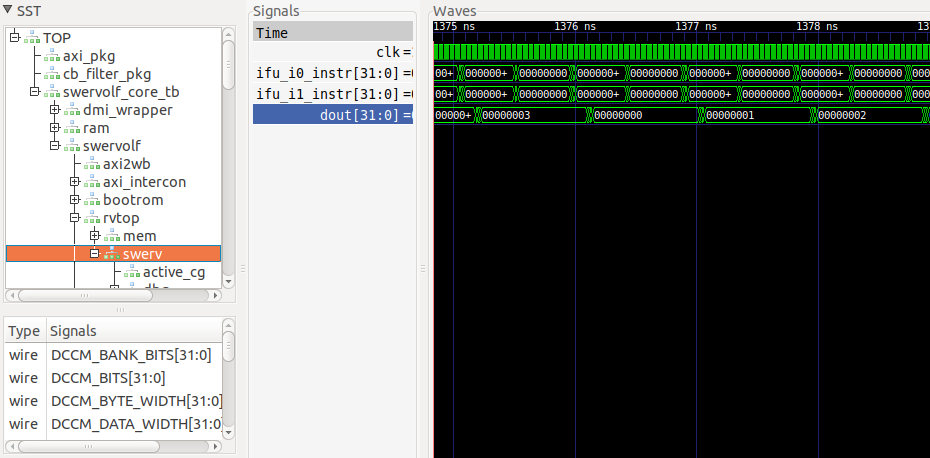
**Step 4**. Open the trace file :

**$** gtkwave trace.vcd

****

**Figure 18. open trace.vcd file in gtkwave**

Now, **gtkwave** will be launched. You will now need to repeat the process of Lab 2 to add the signals to the graph and analyze them.



**Figure 19. analyze the signals**

# Running Blinky Example on SweRVolf Nexys

In this section, we show how to run the Blinky program on the SweRVolf Nexys. We will be using the “**.bit**” file generated by FuseSoC earlier in this Lab.

|  |
| --- |
| **Note:** The address memory maps for LEDs and I/O for the RVfpga-SoC modules used in Lab 1 and Lab 2 are different from SweRVolf (FuseSoC-based SoC for SweRV EH1). Hence there are separate example directories of the same Example Program (Blinky) in Lab 2 and Lab 3 with changed memory addresses. |

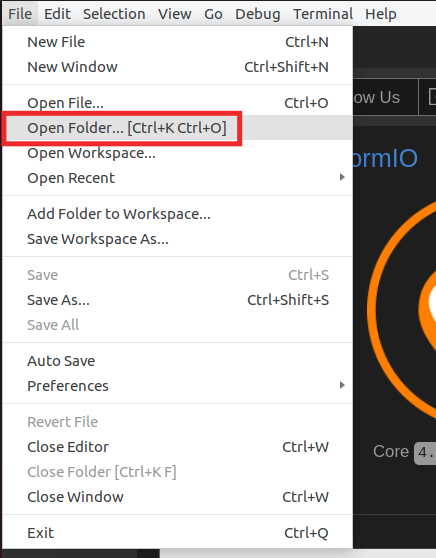
Complete the following steps to program your Nexys A7 board with SweRVolf Nexys and then run the Blinky Program:

1. Connect the Nexys A7 board to your computer.
2. Turn on the Nexys A7 board using the switch at the top left. (see Figure 20)



**Figure 20. Nexys A7 Board ON/OFF Button**

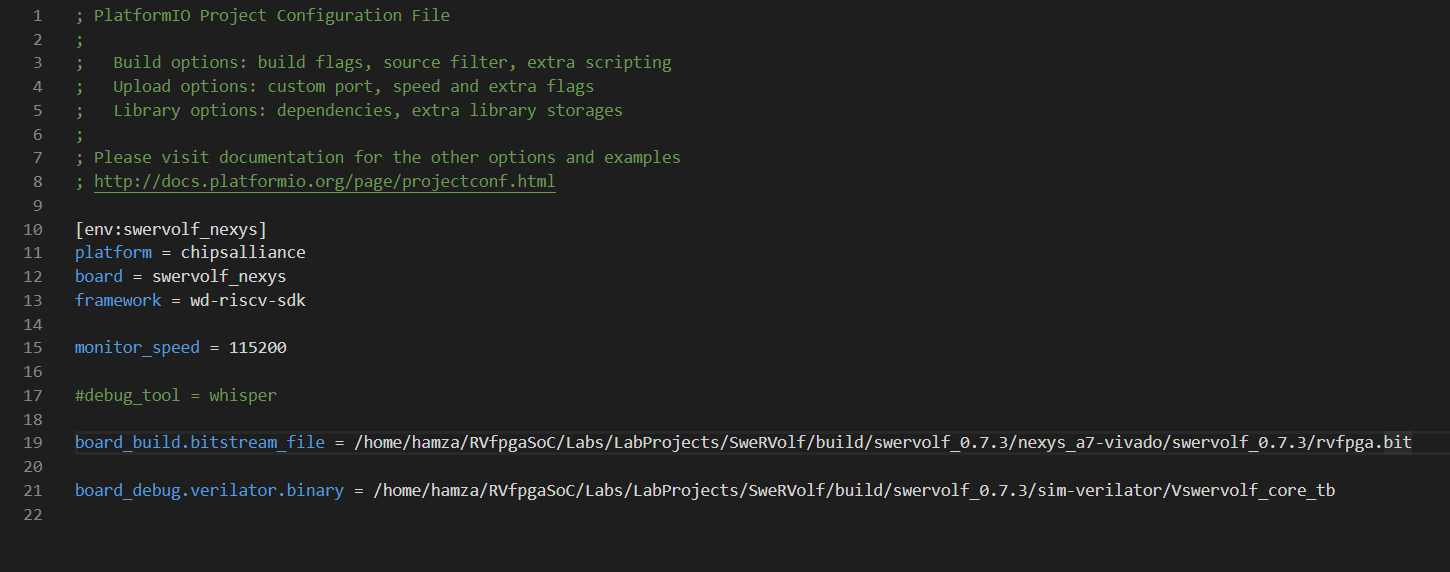
1. Open VSCode and PlatformIO if it is not already open.
2. On the top menu bar, click on *File* → *Open Folder* (see Figure 21) and browse into directory *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab3/examples/*.



**Figure 21. Open Folder**

1. Select the directory *Blinky\_FuseSoC* (do not open it, but just select it)and click OK at the top of the window. PlatformIO will now open the example.
2. Open file *platformio.ini* by clicking on *platformio.ini* in the left sidebar (see Figure 22). Establish the path to the RVfpga bitstream in your system by editing the following line (see Figure 22).
3. The bit file created using FuseSoC is in the following path :

board\_build.bitstream\_file = /home/{Username}/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf\_0.7.3/nexys\_a7-vivado/swervolf\_0.7.3.bit

****

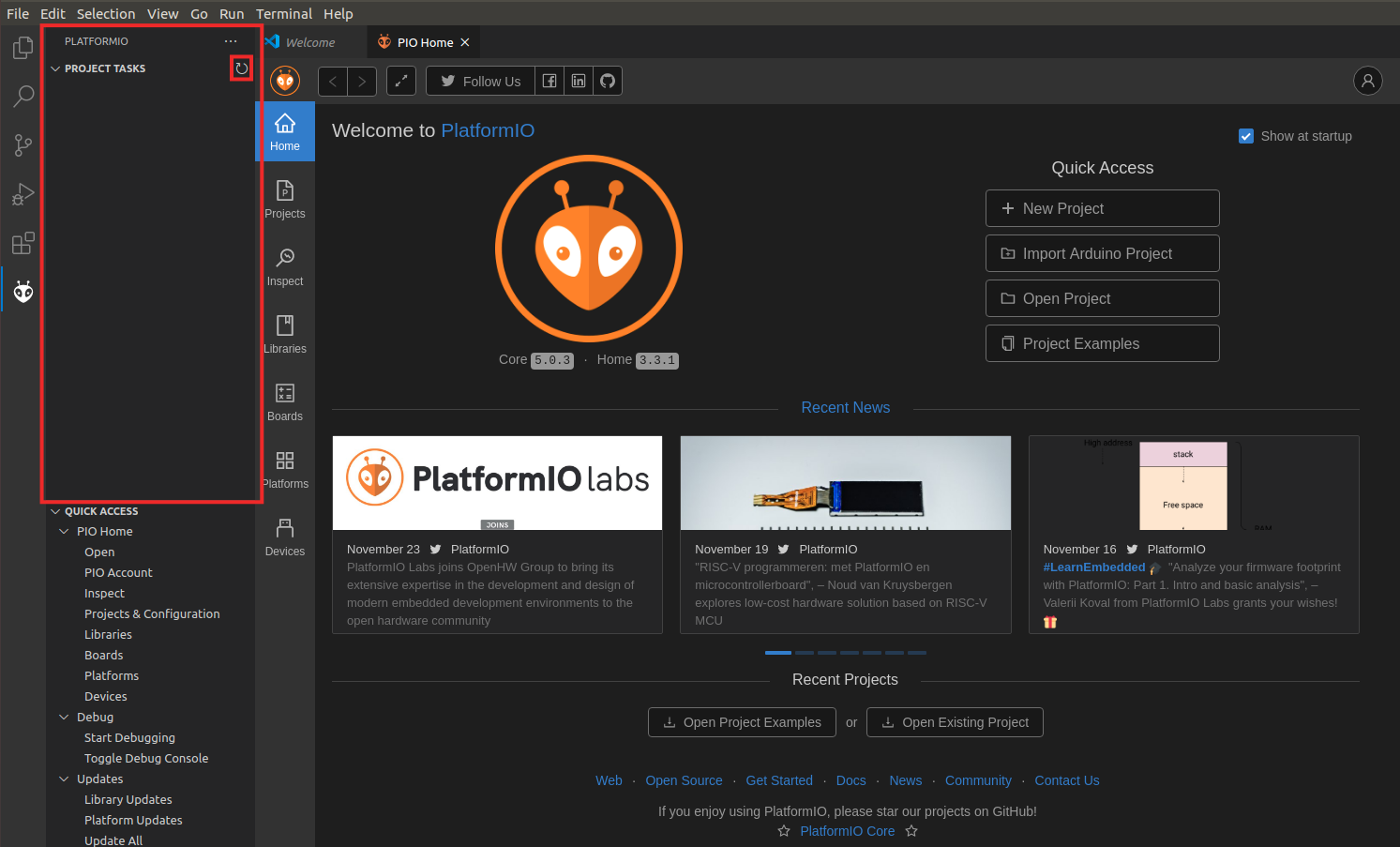
**Figure 22. Platformio initialization file: platformio.ini**

1. Click on the PlatformIO icon  in the left menu ribbon (see Figure 23).



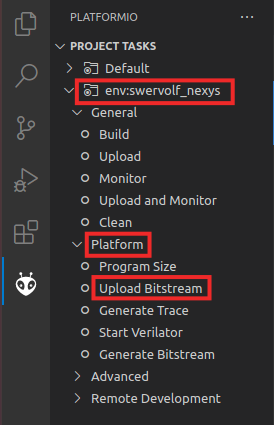
**Figure 23. PlatformIO icon**

In case the Project Tasks window is empty (Figure 24), you must refresh the Project Tasks first by clicking on . This can take several minutes.



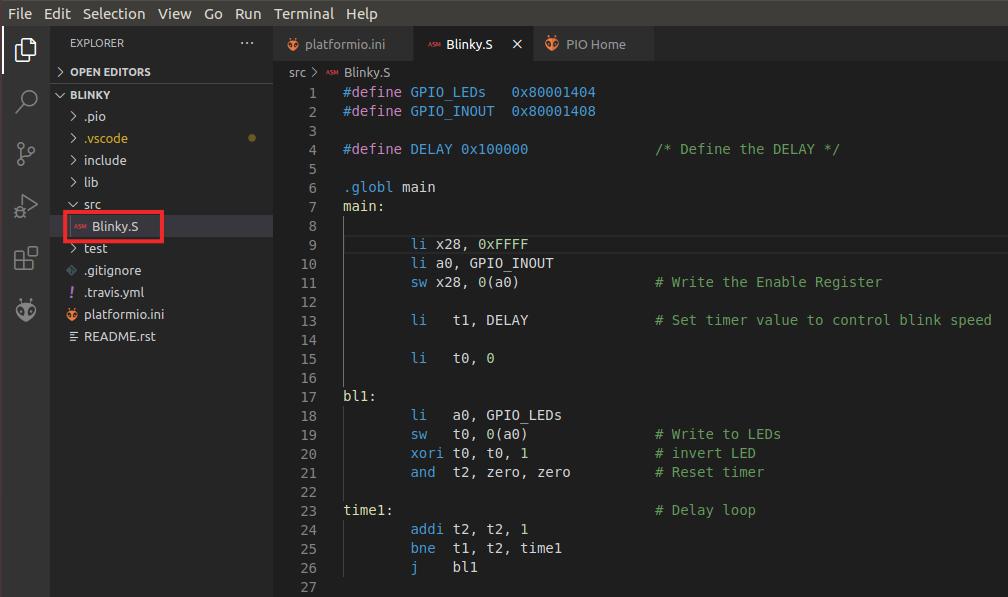
**Figure 24. PROJECT TASKS window empty – Refresh**

Expand Project Tasks → env:swervolf\_nexys → Platform and click on Upload Bitstream, as shown in Figure 25.



**Figure 25. Upload Bitstream**

Now that the bitstream has been uploaded, we will start the debugging process.

****

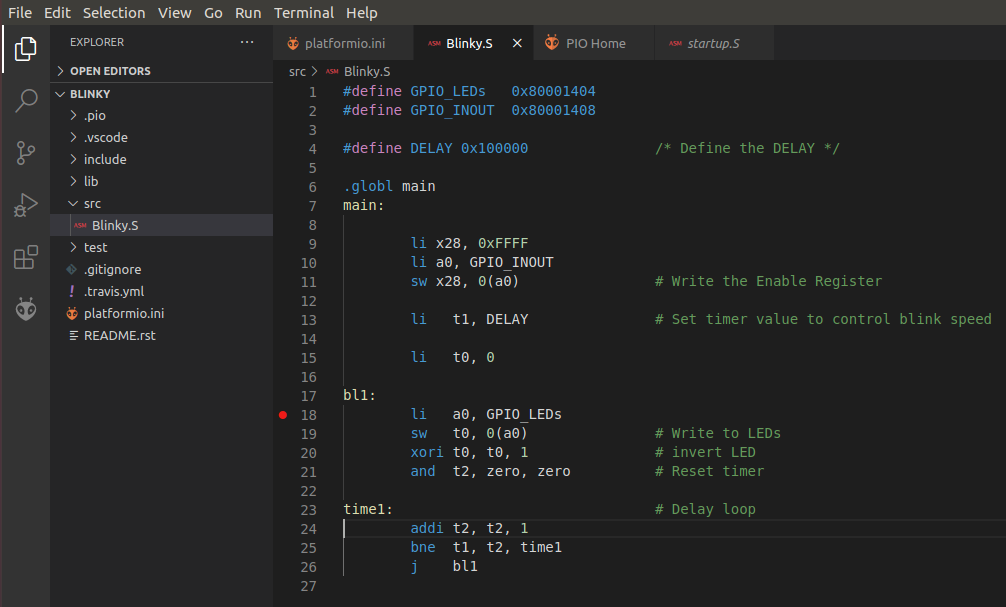
**Figure 26. blinky.S in PlatformIO**

1. Click on  to run and debug the program; then start debugging by clicking on the play button . PlatformIO sets a temporary breakpoint at the beginning of the main function. So, click on the Continue button  to run the program.
2. On the board, you will see the right-most LED start to blink.



**Figure 27. rightmost LED Blinking**

1. Pause the execution by clicking on the pause button . The execution will stop somewhere inside the infinite loop (probably, inside the time1 delay loop).
2. Establish a breakpoint by clicking to the left of line number 18. A red dot will appear, and the breakpoint will be added to the BREAKPOINTS tab (see Figure 28).

****

**Figure 28. Setting a breakpoint in blinky.S**

1. Then, continue execution by clicking on the Continue button . Execution will continue, and it will stop after the store byte (sb) instruction, which writes 1 (or 0) to the right-most LED.
2. Continue execution several times; you will see that the value-driven to the right-most LED changes each time.
3. Stop debugging  and go back to the Explorer window by clicking on . Close the program by selecting *File* → *Close Folder.*

# Debugging SweRVolf :

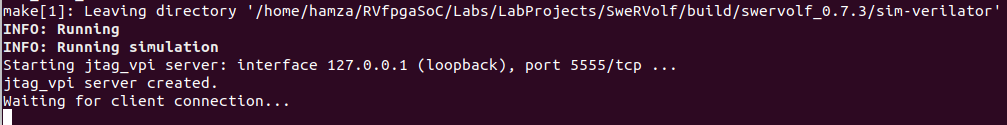
SweRVolf supports debugging both on hardware and in simulation. There are different procedures on how to connect the debugger, but once connected, the same commands can be used (although programs run much slower in simulation).

**SIMULATION :**

**Step 1**. Enter the WORKSPACE directory “SweRVolf” and then run the simulation command:

**$** fusesoc run --target=sim swervolf --jtag\_vpi\_enable

The “***--jtag\_vpi\_enable***” parameter enables the JTAG server to which OpenOCD can connect to. When a SweRVolf simulation is launched with the “***--jtag\_vpi\_enable***”, it will start a JTAG server waiting for a client to connect and send JTAG commands.



**Figure 29. Run the simulation command**

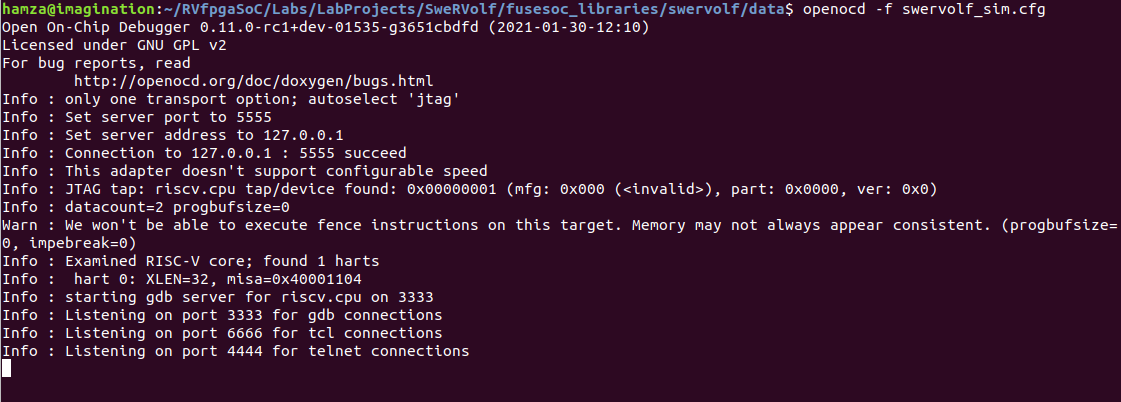
**Step 2**.Open a new terminal using “Ctrl + Shift + t”, then from the Workspace directory, navigate to the data directory by running :

**$** cd fusesoc\_libraries/swervolf/data/

Now, run the following command to connect OpenOCD to the simulation instance.

**$** openocd -f swervolf\_sim.cfg

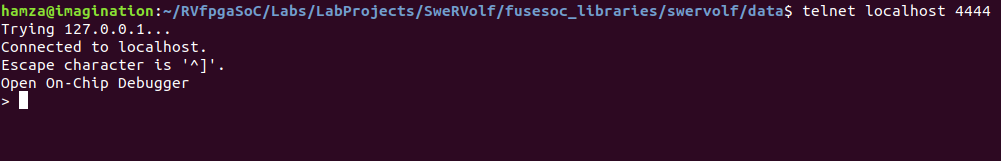
If successful, OpenOCD should output the following (see Figure 30).



**Figure 30. Run OpenOCD**

**Step 3**. Open a third terminal using “Ctrl + Shift + t” and connect to the debug session through OpenOCD.

**$** telnet localhost 4444



**Figure 31. telnet**

**Step 4**. Now enter commands to give instructions :

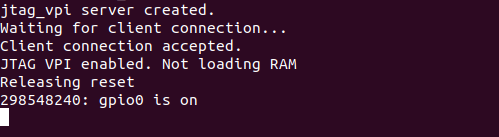
We can now provide live instructions to SweRVolf through the third terminal.

In this FuseSoC based SoC, 16 LEDs are controlled by memory-mapped GPIO at address **0x80001010-0x80001011.** These addresses are different from the RVfpga-SoC modules that we used in Labs 1 and 2.

So we can write a value of “1” at that address in the memory.

**>** mwb 0x80001010 1

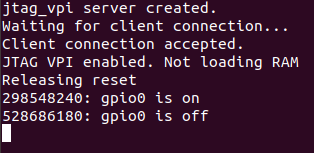
Now open up the first terminal, and you will see that gpio0 is now ON (see Figure 32).



**Figure 32. gpio0 is ON.**

Again, go back to the third terminal and write a value of “0” at the same memory address to turn OFF gpio0.

**>** mwb 0x80001010 0



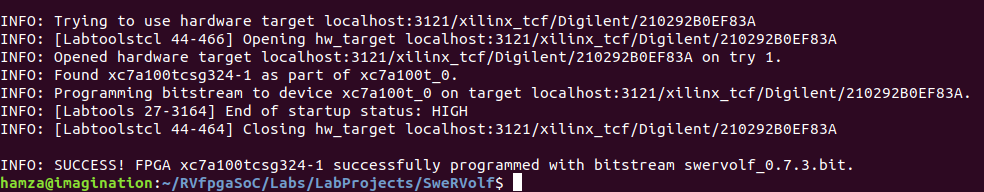
**Figure 33. gpio0 is OFF.**

Now we will complete this same process on the Nexys A7 board.

**HARDWARE :**

**Step 1**. Connect the Nexys A7 board to your computer and then run the FPGA build command in the Workspace directory. To upload the bitstream without rebuilding for the Nexys A7 board again, you can add the “**--run**” parameter.

**$** fusesoc run --target=nexys\_a7 --run swervolf

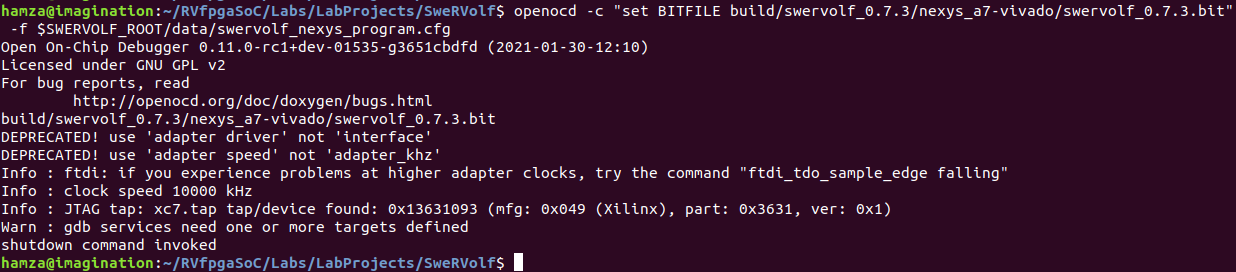


**Figure 34. Run FPGA Build**

**Step 2.** Program the board using OpenOCD

**$** openocd -c "set BITFILE build/swervolf\_0.7.3/nexys\_a7-vivado/swervolf\_0.7.3.bit" -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_program.cfg

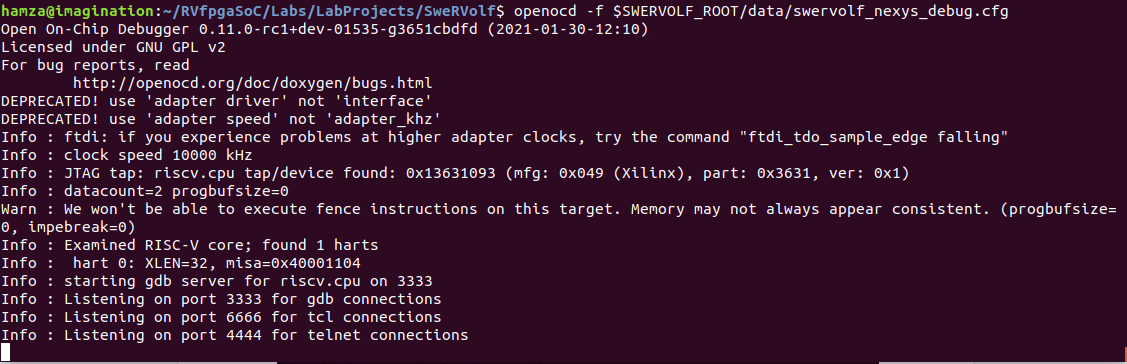
It should give the output as shown in Figure 35.



**Figure 35. Run OpenOCD**

**Step 3.** Connect OpenOCD to SweRVolf

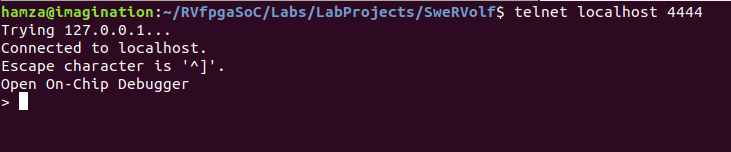
**$** openocd -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_debug.cfg



**Figure 36. OpenOCD Connected**

**Step 3**. Open a third terminal using “Ctrl + Shift + t” and connect to the debug session through OpenOCD:

**$** telnet localhost 4444



**Figure 37. telnet**

**Step 4**. Now enter commands to give instructions to the board directly :

We can now give live instructions to SweRVolf through the third terminal.

The 16 LEDs are controlled by memory-mapped GPIO at addresses **0x80001010 - 0x80001011.**

We write a value of “1” to 0x80001010 in memory to turn ON the right-most LED:

**>** mwb 0x80001010 1

Now run the following command to turn the right-most LED OFF:

**>** mwb 0x80001010 0

This will turn OFF the right-most LED on the board.

In the end, to summarize all this lab we first compared the Block Design code with the Hand-tuned code. We concluded that the hand-tuned code is more preferable and then we were introduced to FuseSoC, which is a package manager that contains **SweRVolf** (FuseSoC based SoC for SweRV Eh1) package. We then explored its two versions “SweRVolf SIM” and “SweRVolf Nexys”. Similar to the **RVfpgaSIM** in the previous lab, **SweRVolf SIM** is a simulation target that wraps the SweRVolf core in a testbench to be used by verilator and similar to the **RVfpga**, **SweRVolf Nexys** is a version of the SweRVolf SoC targeted to the Digilent Nexys A7 board.

We then build the SweRVolf SIM for simulation and SweRVolf Nexys for the Nexys A7 board using FuseSoC run. Later we run the same examples on these builds that we ran in the previous lab using the provided simulation binary and the block design generated bitstream. At the end we learn the process of debugging SweRVolf using OpenOCD.