THE IMAGINATION UNIVERSITY PROGRAMME

# RVfpga-SoC Lab 3
# Introduction to SweRVolf and FuseSoC

**Table 1. RVfpga Terms**

| Name | Description |
|---|---|
| **Courses** | |
| **RVfpga** | A course that shows how to use RVfpgaNexys and RVfpgaSim, RISC-V system-on-chips (SoCs), to run programs and extend the system by adding peripherals (RVfpga Labs 1-10), and explore the core and memory system by running simulations, measuring performance, adding instructions, and modifying the memory system (RVfpga Labs 11-20). Throughout the course, users are also shown how to use the RISC-V toolchain (compilers and debuggers) and simulators, the Verilator HDL simulator, and Western Digital's Whisper instruction set simulator (ISS). |
| **RVfpga-SoC** | A course that shows how to build a subset SweRVolfX SoC from scratch using building blocks such as the SweRV core, memories, and peripherals. The course also shows how to load the Zephyr real-time operating system (RTOS) onto SweRVolf and run programs including Tensorflow Lite's hello world example on top of the operating system. |
| **Cores and SoCs** | |
| **SweRV EH1 Core** | Open-source commercial RISC-V core developed by Western Digital (https://github.com/chipsalliance/Cores-SweRV). |
| **SweRV EH1 Core Complex** | SweRV EH1 core with added memory (ICCM, DCCM, and instruction cache), programmable interrupt controller (PIC), bus interfaces, and debug unit (https://github.com/chipsalliance/Cores-SweRV). |
| **SweRVolfX** | The System on Chip that we use in the RVfpga course. It is an extension of SweRVolf.<br>**SweRVolf** (https://github.com/chipsalliance/Cores-SweRVolf): An open-source SoC built around the SweRV EH1 Core Complex. It adds a boot ROM, UART interface, system controller, interconnect (AXI Interconnect, Wishbone Interconnect, and AXI-to-Wishbone bridge), and an SPI controller.<br>**SweRVolfX**: It adds four new peripherals to SweRVolf: a GPIO, a PTC, an additional SPI, and a controller for the 8 Digit 7-Segment Displays. |
| **RVfpgaNexys** | The SweRVolfX SoC targeted to the Nexys A7 board and its peripherals. It adds a DDR2 interface, CDC (clock domain crossing) unit, BSCAN logic (for the JTAG interface), and clock generator.<br>RVfpgaNexys is the same as SweRVolf Nexys (https://github.com/chipsalliance/Cores-SweRVolf), except that the latter is based on SweRVolf. |
| **RVfpgaSim** | The SweRVolfX SoC with a testbench wrapper and AXI memory intended for simulation.<br>RVfpgaSim is the same as SweRVolf Sim, (https://github.com/chipsalliance/Cores-SweRVolf), except that the latter is based on SweRVolf. |

## 1. Introduction

In this Lab, we will analyze SweRVolf in detail. We will also introduce FuseSoC and show how to use it to build SweRVolf (https://github.com/chipsalliance/Cores-SweRVolf/).
In Lab 1, individual modules were written in Verilog/SystemVerilog. We then connected these modules to create a subset of SweRVolfX 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.

**Table 2. Pros and cons of Block Design**

| **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.

**Table 3. Pros and cons of Hand-tuned code**

| **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.

## 2. 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.08)) |
| ● GTKWave | (Refer to Installation Guide (Page No.08)) |
| ● FuseSoC | (Refer to Installation Guide (Page No.09)) |
| ● OpenOCD (RISC-V-specific version) | (Refer to Installation Guide (Page No.09)) |

**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.

## 3. 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).

## 4. 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.

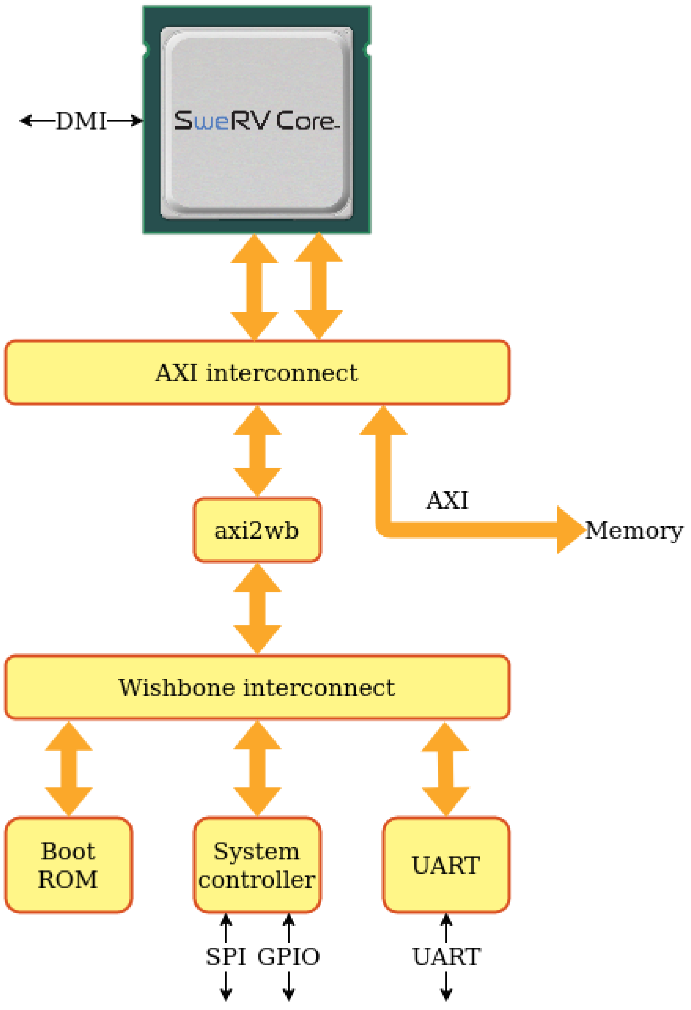


**Figure 1. SweRVolf Core**

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).

Table 4 gives the memory-mapped addresses of the peripherals that are connected to the SweRV EH1 core via the Wishbone interconnect & AXI interconnect.

**Table 4. MEMORY-MAPPED Addresses of SweRVolf**

| System | Address |
|---|---|
| RAM | 0x00000000-0x07FFFFFF |
| Boot ROM | 0x80000000-0x80000FFF |
| System Controller | 0x80001000-0x80001FFF |
| UART | 0x80002000-0x80002FFF |

| | |
|---|---|
| GPIO | 0x80001010-0x80001013 |

Similar to the SystemVerilog modules RVfpgaSim and RVfpgaNexys 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 (see Figure 2).

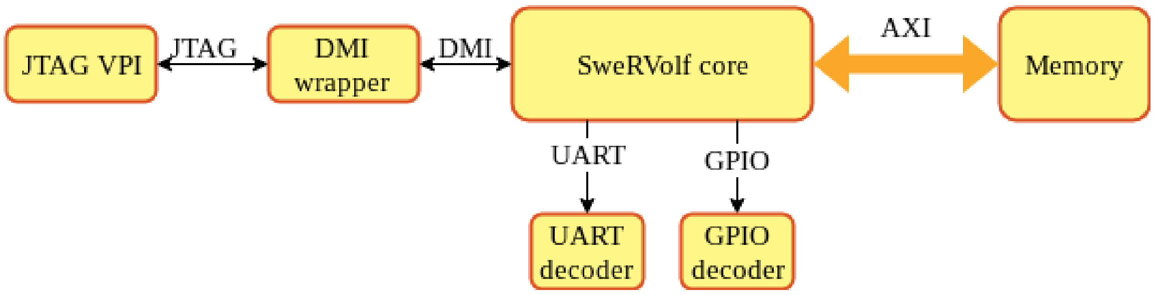


**Figure 2. SweRVolf Sim**

## 2. 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 (see Figure 3).

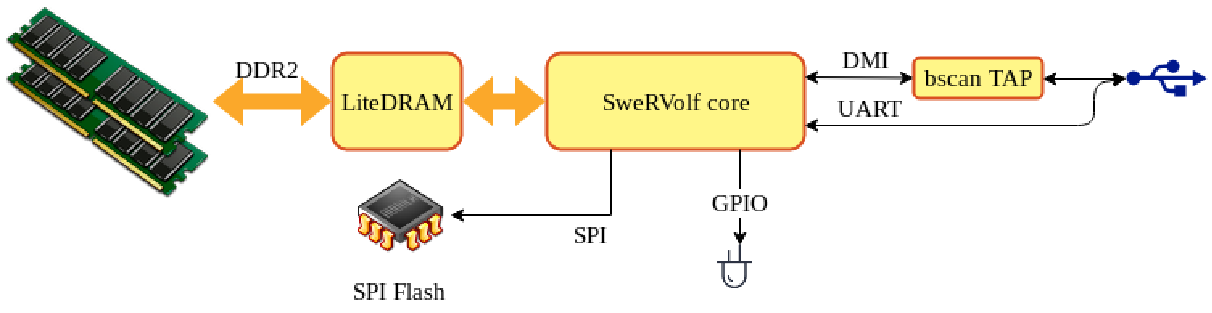


**Figure 3. SweRVolf Nexys**

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.

Like the **RVfpgaSim** in the previous lab, **SweRVolf Sim** is a simulation target that wraps the SweRVolf core in a testbench to be used by the verilator. Similarly, the **RVfpgaNexys** in the previous lab is similar to the **SweRVolf Nexys** that is a version of the SweRVolf SoC targeted to the Digilent Nexys A7 board.

## 5. Setting up the Environment

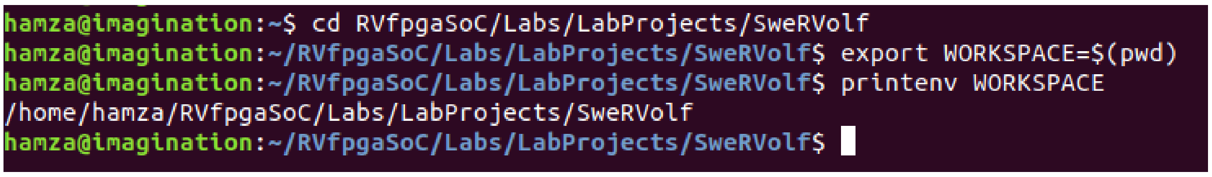
In this section, we show how to set up the environment for the Sim build and Nexys build.

**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 4). You can use the following command to verify if the shell variable was successfully added.

➢ `printenv WORKSPACE`



```
hamza@imagination:~$ cd RVfpgaSoC/Labs/LabProjects/SweRVolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export WORKSPACE=$(pwd)
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ printenv WORKSPACE
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ █
```

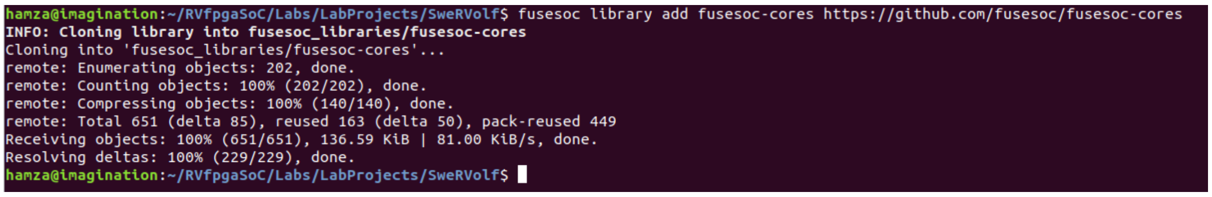
**Figure 4. Establish Workspace**

**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`

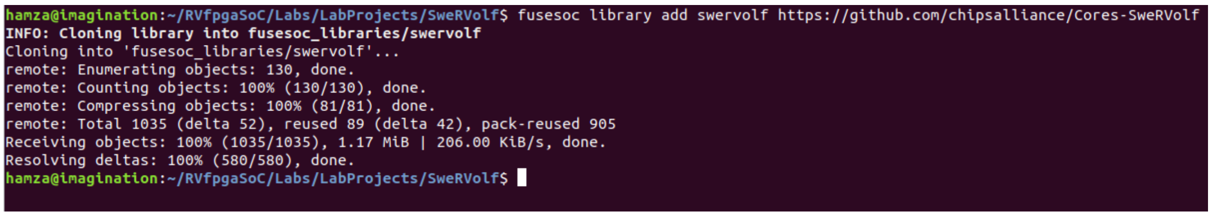


```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc library add fusesoc-cores https://github.com/fusesoc/fusesoc-cores
INFO: Cloning library into fusesoc_libraries/fusesoc-cores
Cloning into 'fusesoc_libraries/fusesoc-cores'...
remote: Enumerating objects: 202, done.
remote: Counting objects: 100% (202/202), done.
remote: Compressing objects: 100% (140/140), done.
remote: Total 651 (delta 85), reused 163 (delta 50), pack-reused 449
Receiving objects: 100% (651/651), 136.59 KiB | 81.00 KiB/s, done.
Resolving deltas: 100% (229/229), done.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ █
```

**Figure 5. Add FuseSoC base library**

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

➢ `fusesoc library add swervolf https://github.com/chipsalliance/Cores-SweRVolf`

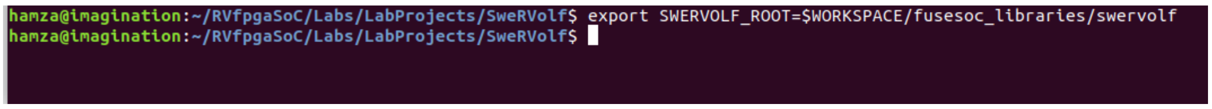


```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc library add swervolf https://github.com/chipsalliance/Cores-SweRVolf
INFO: Cloning library into fusesoc_libraries/swervolf
Cloning into 'fusesoc_libraries/swervolf'...
remote: Enumerating objects: 130, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (81/81), done.
remote: Total 1035 (delta 52), reused 89 (delta 42), pack-reused 905
Receiving objects: 100% (1035/1035), 1.17 MiB | 206.00 KiB/s, done.
Resolving deltas: 100% (580/580), done.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 6. Add Swervolf library**

**Step 5.** Set the swervolf directory as the "SWERVOLF_ROOT" shell variable:

➢ `export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf`



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 7. Set shell variable**

## 6. 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.

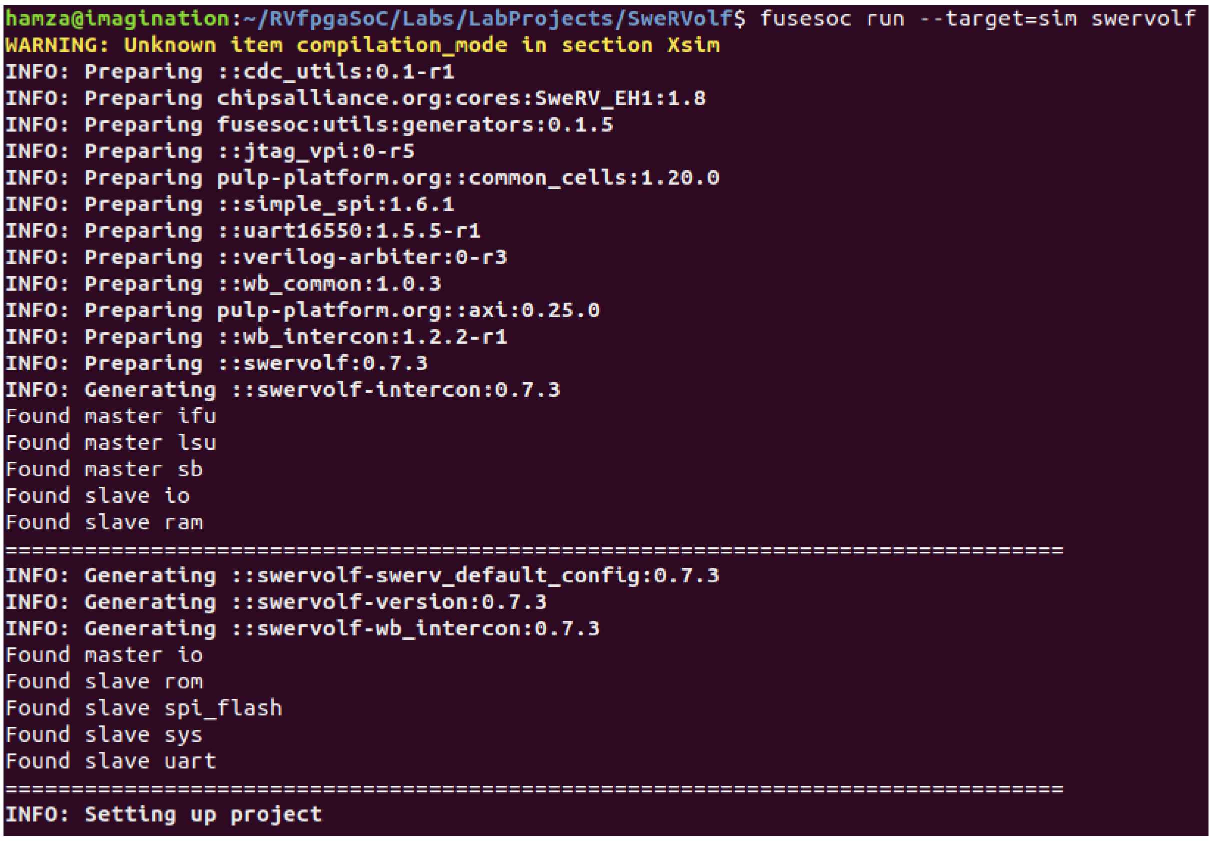
**Step 6.** To run in simulation use

➢ `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/*

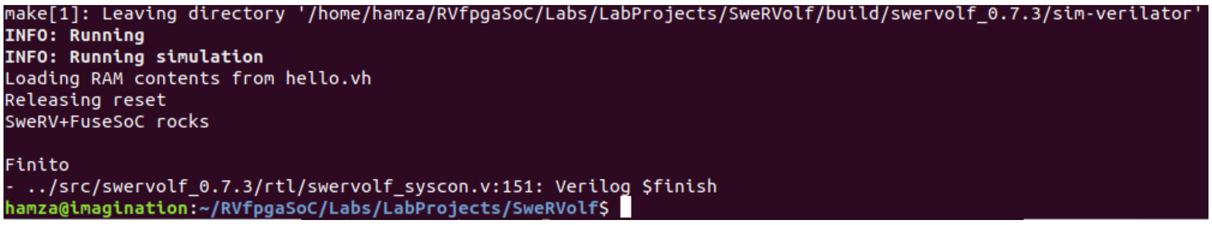
Note that this is similar to what we did in Lab 2 when we generated the "**Vrvfpgasim**" simulation binary with the make command in the following path: *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM*

However, the SoC used in Lab 2 was a subset of the SweRVolfX, whereas the SoC used here is the complete SweRVolf.



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
WARNING: Unknown item compilation_mode in section Xsim
INFO: Preparing ::cdc_utils:0.1-r1
INFO: Preparing chipsalliance.org:cores:SweRV_EH1:1.8
INFO: Preparing fusesoc:utils:generators:0.1.5
INFO: Preparing ::jtag_vpi:0-r5
INFO: Preparing pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::simple_spi:1.6.1
INFO: Preparing ::uart16550:1.5.5-r1
INFO: Preparing ::verilog-arbiter:0-r3
INFO: Preparing ::wb_common:1.0.3
INFO: Preparing pulp-platform.org::axi:0.25.0
INFO: Preparing ::wb_intercon:1.2.2-r1
INFO: Preparing ::swervolf:0.7.3
INFO: Generating ::swervolf-intercon:0.7.3
Found master ifu
Found master lsu
Found master sb
Found slave io
Found slave ram
================================================================================
INFO: Generating ::swervolf-swerv_default_config:0.7.3
INFO: Generating ::swervolf-version:0.7.3
INFO: Generating ::swervolf-wb_intercon:0.7.3
Found master io
Found slave rom
Found slave spi_flash
Found slave sys
Found slave uart
================================================================================
INFO: Setting up project
```

.........



```
make[1]: Leaving directory '/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Loading RAM contents from hello.vh
Releasing reset
SweRV+FuseSoC rocks

Finito
- ../src/swervolf_0.7.3/rtl/swervolf_syscon.v:151: Verilog $finish
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 8. 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`

## 7. Nexys A7 Build

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

**Step 1.** To build (and optionally program) an image for a Nexys A7 board, run

➢ `fusesoc run --target=nexys_a7 swervolf`

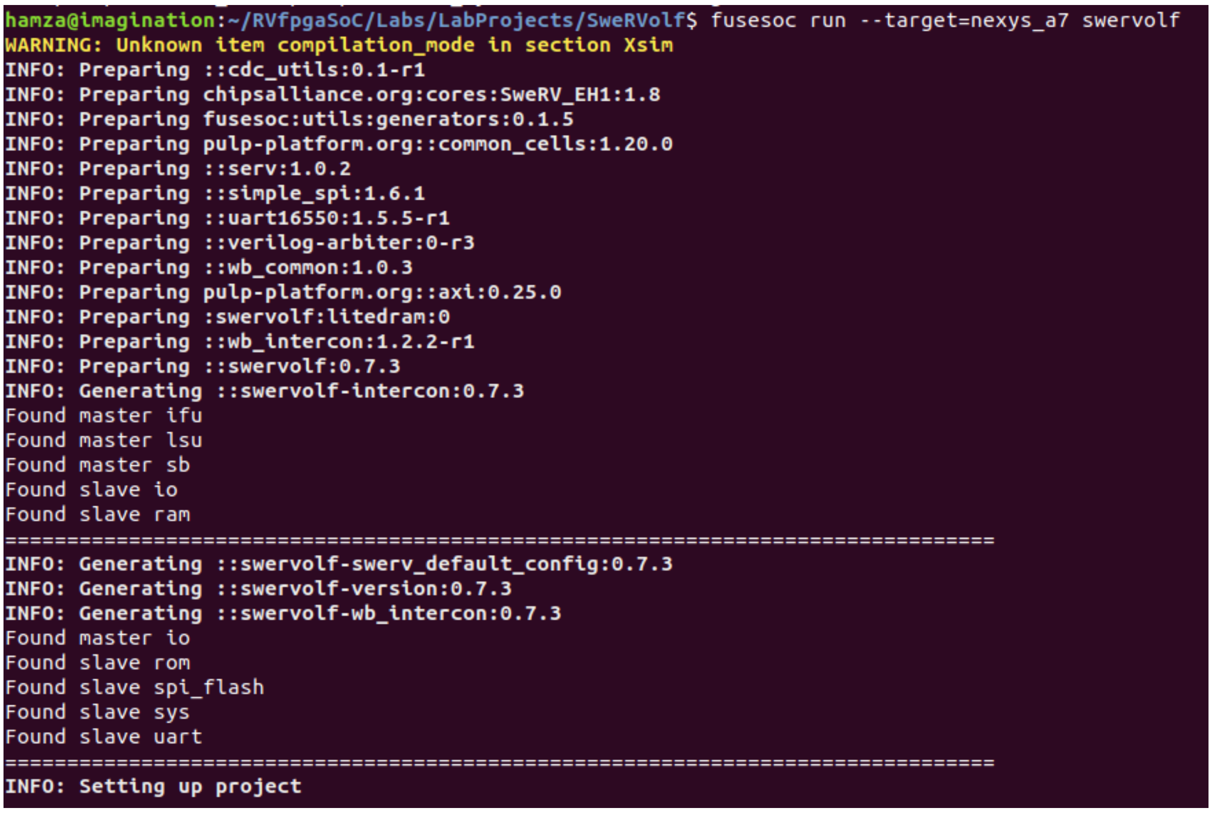
**Note**: Be sure to connect the Nexys A7 board to your machine before running this command and make sure the board is turned ON.

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.

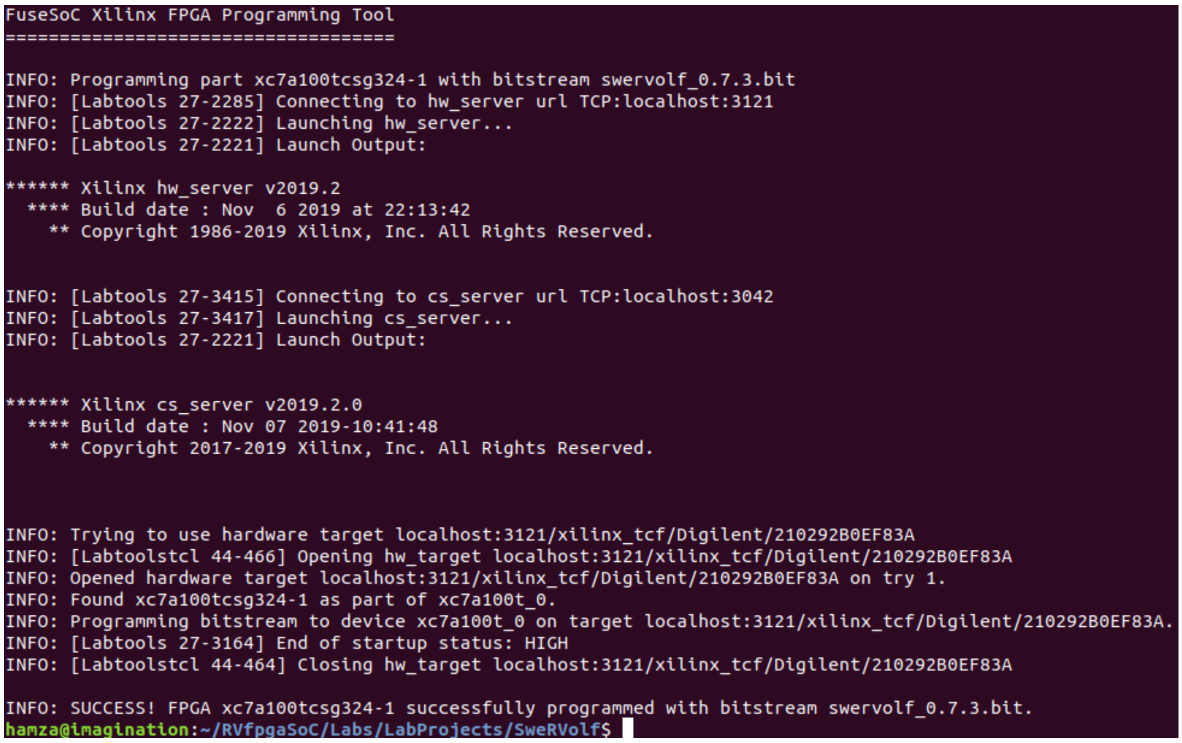
This is similar to what we did in lab 1 when we generated the "**rvfpga.bit**" file in Vivado after creating the block design using the "Generate Bitstream" option. Later in Lab 2, we uploaded the "**rvfpga.bit**" file to our Nexys A7 board using PlatformIO.

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



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=nexys_a7 swervolf
WARNING: Unknown item compilation_mode in section Xsim
INFO: Preparing ::cdc_utils:0.1-r1
INFO: Preparing chipsalliance.org:cores:SweRV_EH1:1.8
INFO: Preparing fusesoc:utils:generators:0.1.5
INFO: Preparing pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::serv:1.0.2
INFO: Preparing ::simple_spi:1.6.1
INFO: Preparing ::uart16550:1.5.5-r1
INFO: Preparing ::verilog-arbiter:0-r3
INFO: Preparing ::wb_common:1.0.3
INFO: Preparing pulp-platform.org::axi:0.25.0
INFO: Preparing :swervolf:litedram:0
INFO: Preparing ::wb_intercon:1.2.2-r1
INFO: Preparing ::swervolf:0.7.3
INFO: Generating ::swervolf-intercon:0.7.3
Found master ifu
Found master lsu
Found master sb
Found slave io
Found slave ram
================================================================================
INFO: Generating ::swervolf-swerv_default_config:0.7.3
INFO: Generating ::swervolf-version:0.7.3
INFO: Generating ::swervolf-wb_intercon:0.7.3
Found master io
Found slave rom
Found slave spi_flash
Found slave sys
Found slave uart
================================================================================
INFO: Setting up project
```

……….



```
FuseSoC Xilinx FPGA Programming Tool
====================================

INFO: Programming part xc7a100tcsg324-1 with bitstream swervolf_0.7.3.bit
INFO: [Labtools 27-2285] Connecting to hw_server url TCP:localhost:3121
INFO: [Labtools 27-2222] Launching hw_server...
INFO: [Labtools 27-2221] Launch Output:

****** Xilinx hw_server v2019.2
  **** Build date : Nov  6 2019 at 22:13:42
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.


INFO: [Labtools 27-3415] Connecting to cs_server url TCP:localhost:3042
INFO: [Labtools 27-3417] Launching cs_server...
INFO: [Labtools 27-2221] Launch Output:


****** Xilinx cs_server v2019.2.0
  **** Build date : Nov 07 2019-10:41:48
    ** Copyright 2017-2019 Xilinx, Inc. All Rights Reserved.



INFO: Trying to use hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: Opened hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A on try 1.
INFO: Found xc7a100tcsg324-1 as part of xc7a100t_0.
INFO: Programming bitstream to device xc7a100t_0 on target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A.
INFO: [Labtools 27-3164] End of startup status: HIGH
INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A

INFO: SUCCESS! FPGA xc7a100tcsg324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 9. SweRVolf Nexys build**

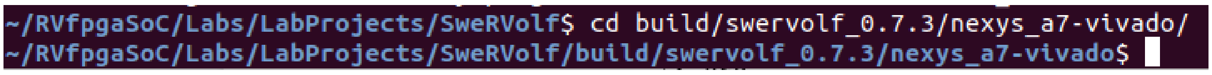
After the completion of the commands as mentioned earlier, our Workspace directory will look like as follows:

- `$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.

We can open the Vivado project to visualize the hierarchy as we saw in Lab 1.
**Step 2.** Enter the "**nexys_a7-vivado**" directory by the following command:

➢ `cd build/swervolf_0.7.3/nexys_a7-vivado/`

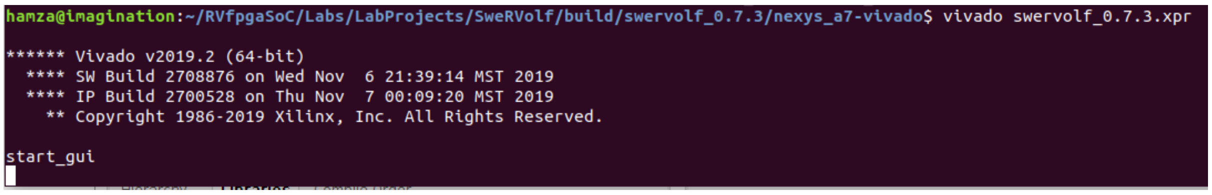


```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd build/swervolf_0.7.3/nexys_a7-vivado/
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/nexys_a7-vivado$
```

**Figure 10. Navigate to "nexys_a7-vivado" Directory**

**Step 3.** Enter the following command to open the Vivado Project:

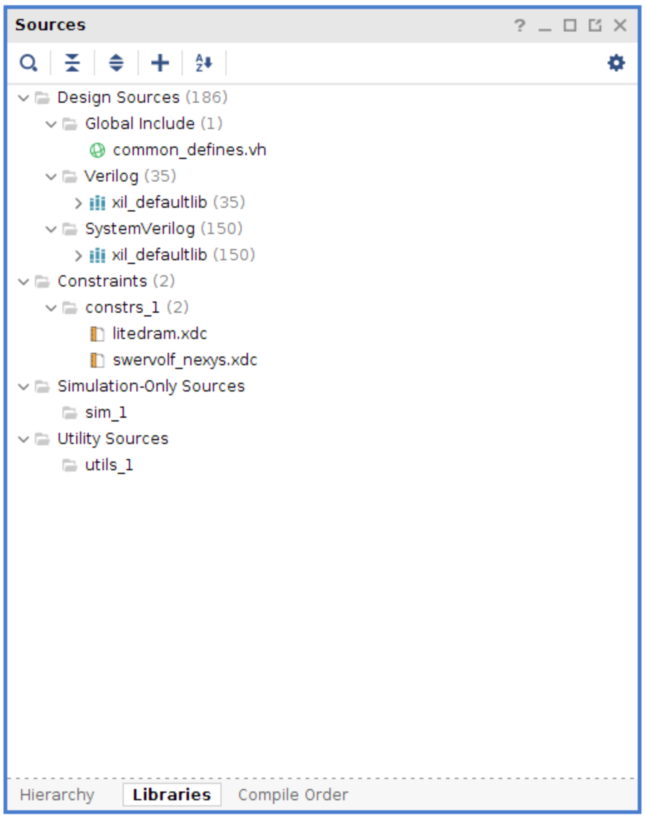
➢ `vivado swervolf_0.7.3.xpr`



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/nexys_a7-vivado$ vivado swervolf_0.7.3.xpr

****** Vivado v2019.2 (64-bit)
  **** SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
  **** IP Build 2700528 on Thu Nov  7 00:09:20 MST 2019
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

start_gui
```

**Figure 11. Open Vivado Project**

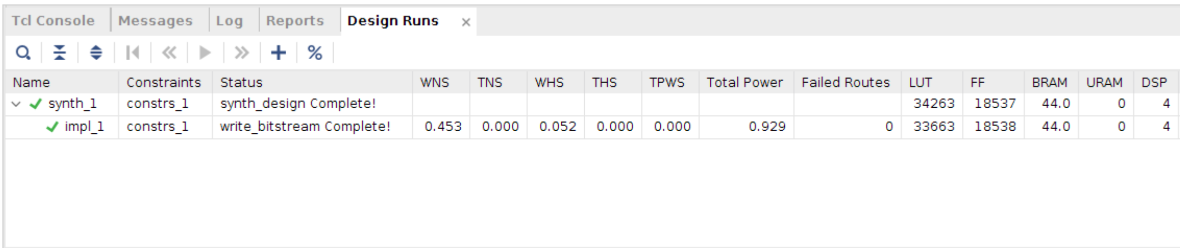
We can visualize the global variables and constraints files set in the sources panel.



```
Sources
Design Sources (186)
  Global Include (1)
    common_defines.vh
  Verilog (35)
    xil_defaultlib (35)
  SystemVerilog (150)
    xil_defaultlib (150)
Constraints (2)
  constrs_1 (2)
    litedram.xdc
    swervolf_nexys.xdc
Simulation-Only Sources
  sim_1
Utility Sources
  utils_1
Hierarchy   Libraries   Compile Order
```

**Figure 12. Sources panel**

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



Tcl Console | Messages | Log | Reports | Design Runs

| Name | Constraints | Status | WNS | TNS | WHS | THS | TPWS | Total Power | Failed Routes | LUT | FF | BRAM | URAM | DSP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ✓ synth_1 | constrs_1 | synth_design Complete! | | | | | | | | 34263 | 18537 | 44.0 | 0 | 4 |
| ✓ impl_1 | constrs_1 | write_bitstream Complete! | 0.453 | 0.000 | 0.052 | 0.000 | 0.000 | 0.929 | 0 | 33663 | 18538 | 44.0 | 0 | 4 |

**Figure 13. Design runs**

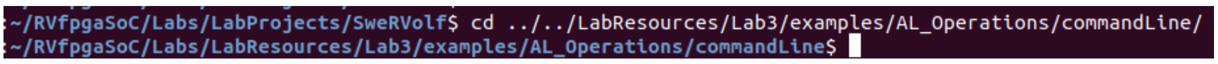
## 8. Running AL_Operations Example on SweRVolf Sim

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/`

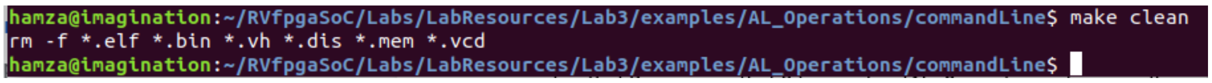


```
:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd ../../LabResources/Lab3/examples/AL_Operations/commandLine/
:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

**Figure 14. AL_Operations directory**

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

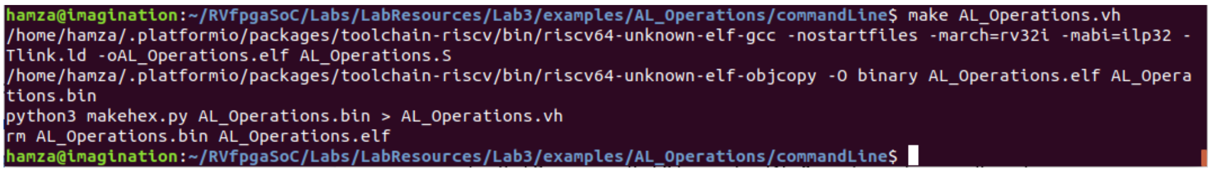
➢ `make clean`



```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ make clean
rm -f *.elf *.bin *.vh *.dis *.mem *.vcd
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

**Figure 15. make clean**

➢ `make AL_Operations.vh`



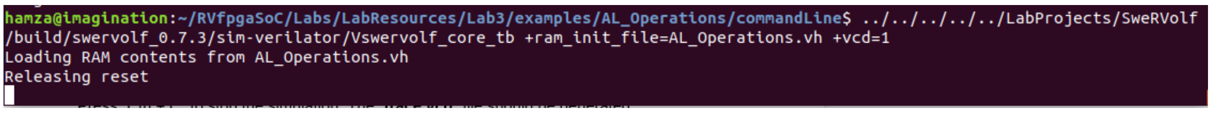
```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ make AL_Operations.vh
/home/hamza/.platformio/packages/toolchain-riscv/bin/riscv64-unknown-elf-gcc -nostartfiles -march=rv32i -mabi=ilp32 -
Tlink.ld -oAL_Operations.elf AL_Operations.S
/home/hamza/.platformio/packages/toolchain-riscv/bin/riscv64-unknown-elf-objcopy -O binary AL_Operations.elf AL_Opera
tions.bin
python3 makehex.py AL_Operations.bin > AL_Operations.vh
rm AL_Operations.bin AL_Operations.elf
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

**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.



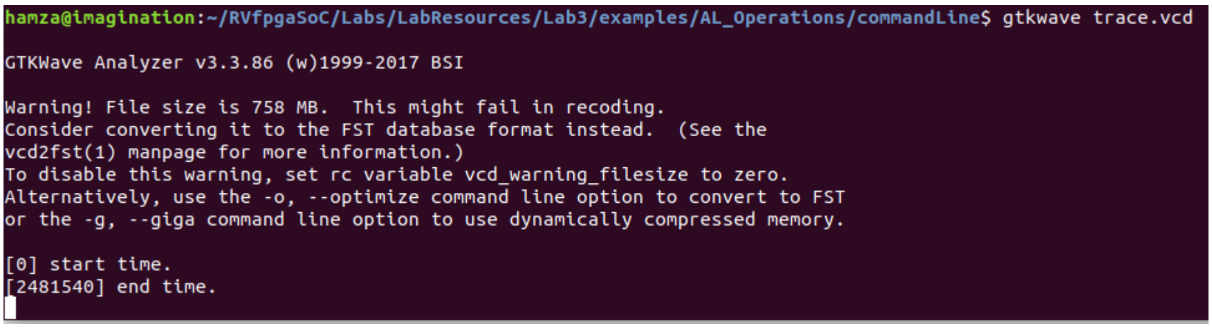
```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ ../../../../../LabProjects/SweRVolf
/build/swervolf_0.7.3/sim-verilator/Vswervolf_core_tb +ram_init_file=AL_Operations.vh +vcd=1
Loading RAM contents from AL_Operations.vh
Releasing reset
```

**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
```



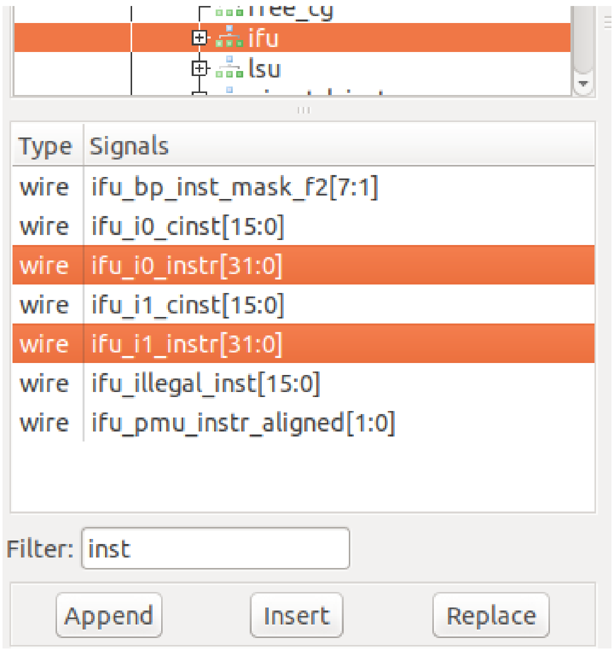
```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ gtkwave trace.vcd

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

Warning! File size is 758 MB.  This might fail in recoding.
Consider converting it to the FST database format instead.  (See the
vcd2fst(1) manpage for more information.)
To disable this warning, set rc variable vcd_warning_filesize to zero.
Alternatively, use the -o, --optimize command line option to convert to FST
or the -g, --giga command line option to use dynamically compressed memory.

[0] start time.
[2481540] end time.
```

**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.

**Step 5.** On the top left pane of *GTKWave*, expand the SoC hierarchy so that you can add signals to the graph. Expand the hierarchy into **TOP** → **swervolf_core_tb**→ **swervolf** → **rvtop** → **swerv,** and click on module **ifu**, select signal *clk* (which is the clock used for the core), and drag it into the white Signals pane or the black Waves pane on the right.

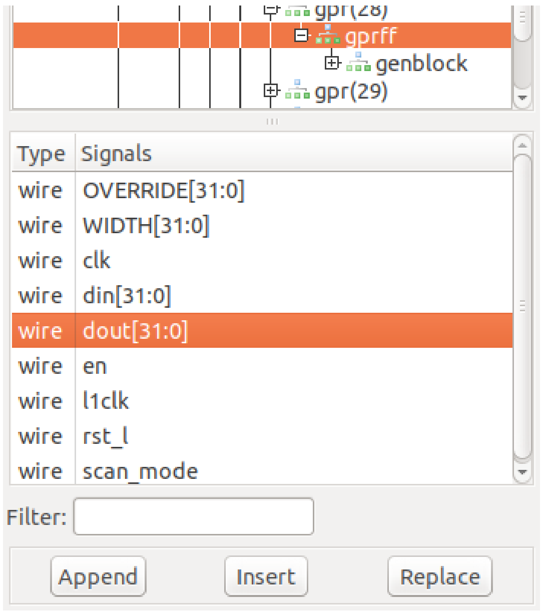
**Step 6.**Make sure the ifu module is highlighted (see Figure 19), then type “inst” in the filter search and then insert the “ifu_i0_instr[31:0]” and “ifu_i1_instr[31:0]” signals.



| Type | Signals |
|---|---|
| wire | ifu_bp_inst_mask_f2[7:1] |
| wire | ifu_i0_cinst[15:0] |
| wire | ifu_i0_instr[31:0] |
| wire | ifu_i1_cinst[15:0] |
| wire | ifu_i1_instr[31:0] |
| wire | ifu_illegal_inst[15:0] |
| wire | ifu_pmu_instr_aligned[1:0] |

**Figure 19. “instr” signals**

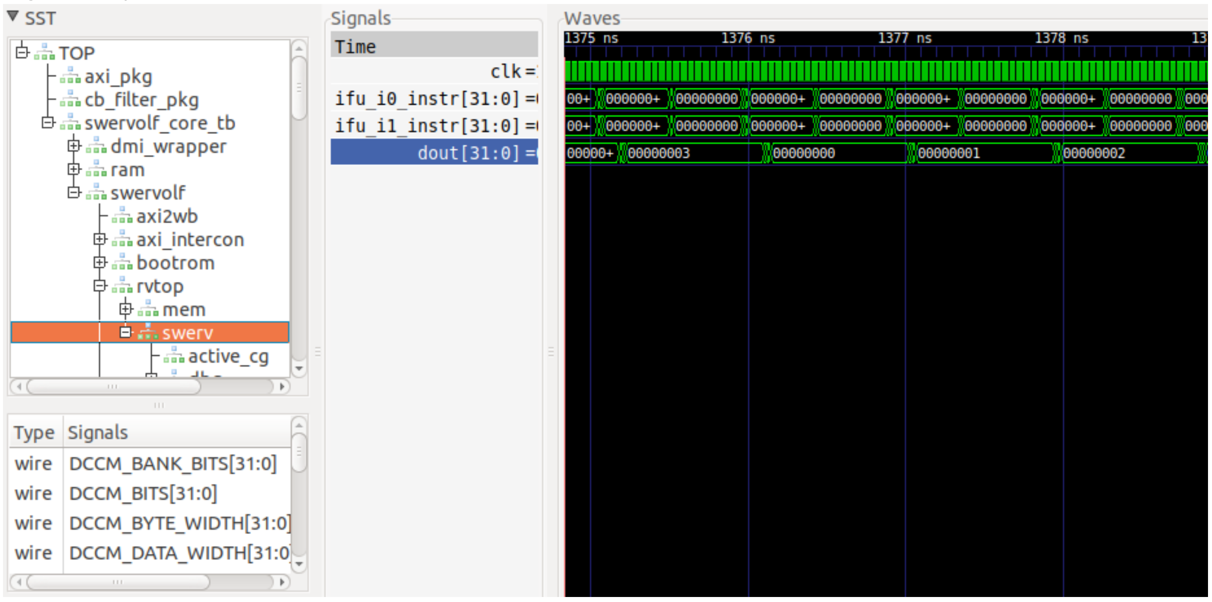
**Step 7.** Now navigate to: **TOP** → **swervolf_core_tb**→ **swervolf** → **rvtop** → **swerv** → **dec** → **arf** → **gpr_banks(0)** → **gpr(28)** → **gprff,** and click on “dout[31:0]” signal and insert it in (see Figure 20).



| Type | Signals |
|---|---|
| wire | OVERRIDE[31:0] |
| wire | WIDTH[31:0] |
| wire | clk |
| wire | din[31:0] |
| wire | dout[31:0] |
| wire | en |
| wire | l1clk |
| wire | rst_l |
| wire | scan_mode |

**Figure 20. "dout" signal**

Now you can zoom in on the signals using the "+" button and analyze the waveforms (see Figure 21).



| Type | Signals |
|---|---|
| wire | DCCM_BANK_BITS[31:0] |
| wire | DCCM_BITS[31:0] |
| wire | DCCM_BYTE_WIDTH[31:0] |
| wire | DCCM_DATA_WIDTH[31:0 |

**Figure 21. analyze the signals**

## 9. 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 SweRVolfX 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.

**Table 5. MEMORY-MAPPED GPIO Addresses of SweRVolf & SweRVolfX**

| SoC | System | Address |
|---|---|---|
| SweRVolfX (Lab 1) | GPIO | 0x80001400 - 0x8000143F |
| SweRVolf (Lab 3) | GPIO | 0x80001010 - 0x80001013 |

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

**Step 1.** Connect the Nexys A7 board to your computer.

**Step 2.** Turn on the Nexys A7 board using the switch at the top left. (see Figure 22)

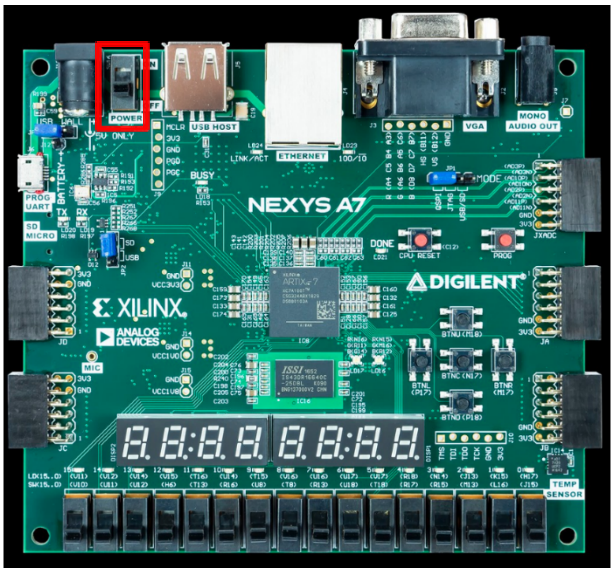


**Figure 22. Nexys A7 Board ON/OFF Button**

**Step 3.** Open VSCode and PlatformIO if it is not already open.

**Step 4.** On the top menu bar, click on *File → Open Folder* (see Figure 23) and browse into directory *[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab3/examples/*.

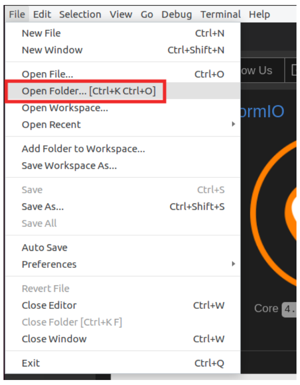


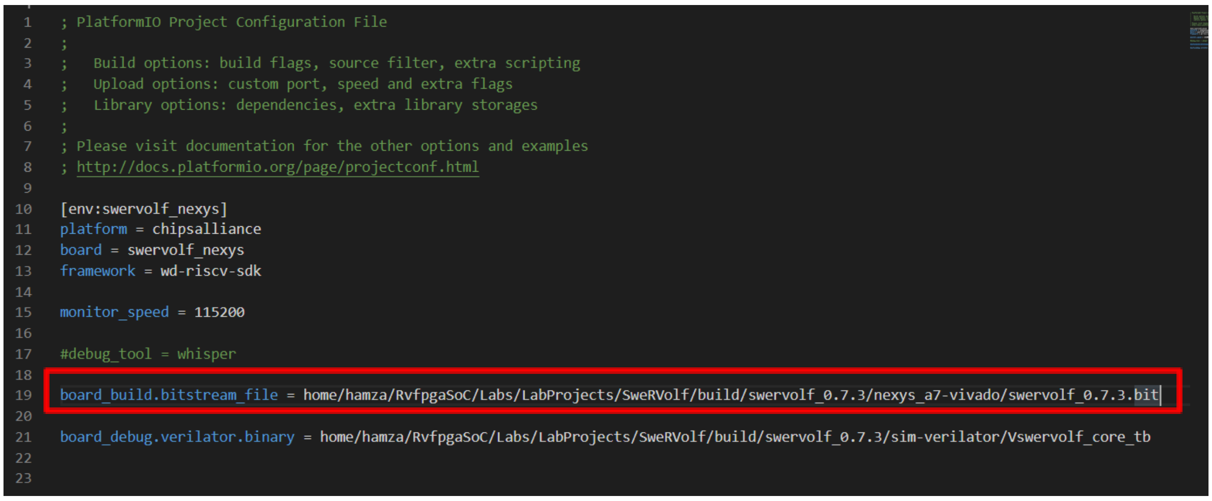
**Figure 23. Open Folder**

**Step 5.** 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.

**Step 6.** Open file *platformio.ini* by clicking on *platformio.ini* in the left sidebar (see Figure 24). Establish the path to the RVfpga bitstream in your system by editing the following line (see Figure 24).

**Step 7.** The "**swervolf_0.7.3.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
```



```
; PlatformIO Project Configuration File
;
;   Build options: build flags, source filter, extra scripting
;   Upload options: custom port, speed and extra flags
;   Library options: dependencies, extra library storages
;
; Please visit documentation for the other options and examples
; http://docs.platformio.org/page/projectconf.html

[env:swervolf_nexys]
platform = chipsalliance
board = swervolf_nexys
framework = wd-riscv-sdk

monitor_speed = 115200

#debug_tool = whisper

board_build.bitstream_file = home/hamza/RvfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit

board_debug.verilator.binary = home/hamza/RvfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/sim-verilator/Vswervolf_core_tb
```

**Figure 24. Platformio initialization file: platformio.ini**

**Step 8.** Click on the PlatformIO icon in the left menu ribbon (see Figure 25).

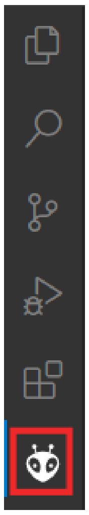
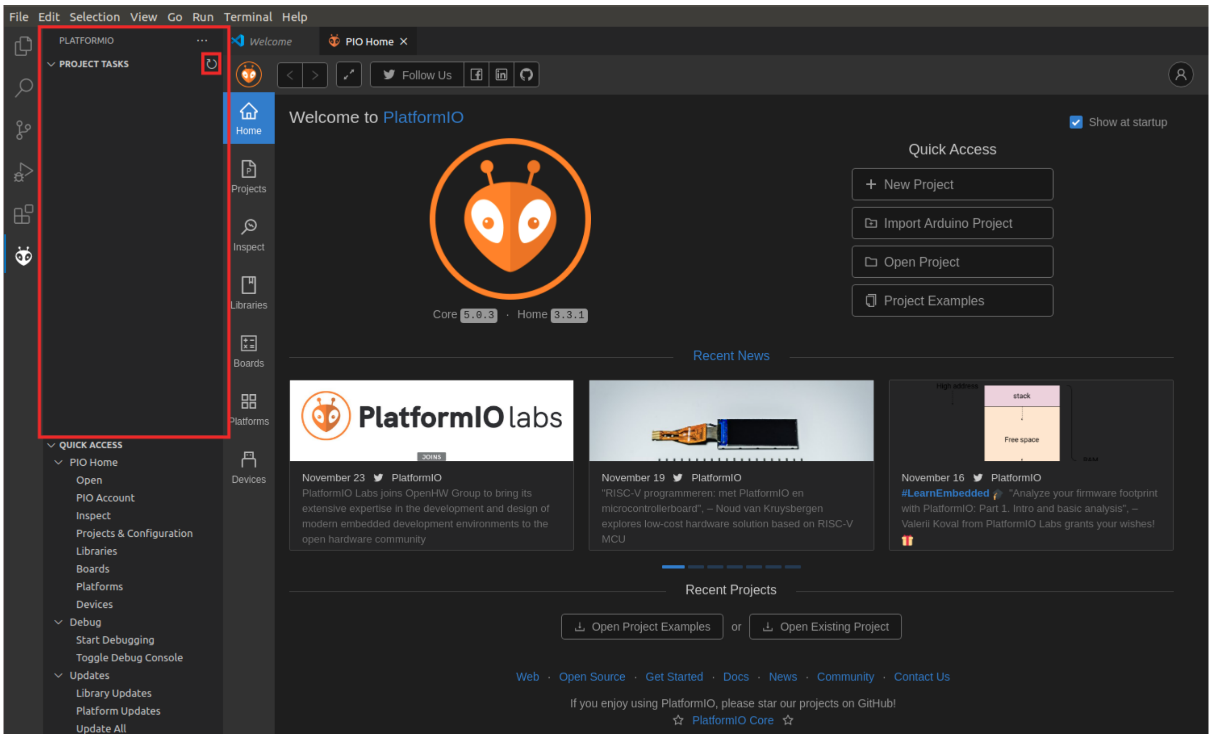


**Figure 25. PlatformIO icon**

In case the Project Tasks window is empty (Figure 26), you must refresh the Project Tasks first by clicking on . This can take several minutes.



November 23 PlatformIO
PlatformIO Labs joins OpenHW Group to bring its extensive expertise in the development and design of modern embedded development environments to the open hardware community

November 19 PlatformIO
"RISC-V programmeren: met PlatformIO en microcontrollerboard", – Noud van Kruysbergen explores low-cost hardware solution based on RISC-V MCU

November 16 PlatformIO
#LearnEmbedded "Analyze your firmware footprint with PlatformIO: Part 1. Intro and basic analysis", – Valerii Koval from PlatformIO Labs grants your wishes!

If you enjoy using PlatformIO, please star our projects on GitHub!

**Figure 26. PROJECT TASKS window empty – Refresh**

**Step 9.** Expand Project Tasks → env:swervolf_nexys → Platform and click on Upload Bitstream, as shown in Figure 27.

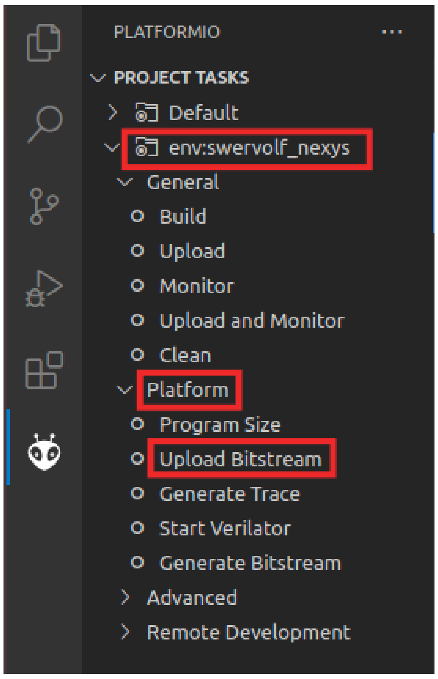
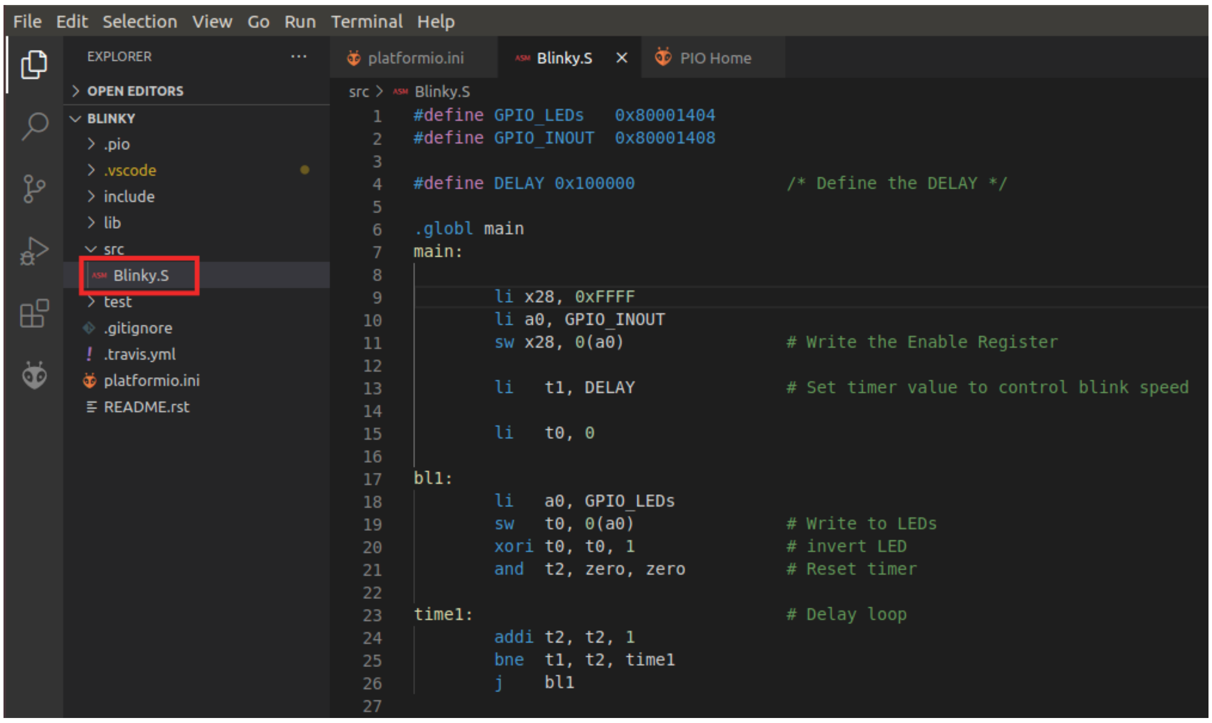


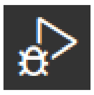
**Figure 27. Upload Bitstream**

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



```
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

#define DELAY 0x100000                  /* Define the DELAY */

.globl main
main:

        li x28, 0xFFFF
        li a0, GPIO_INOUT
        sw x28, 0(a0)                   # Write the Enable Register

        li   t1, DELAY                  # Set timer value to control blink speed

        li   t0, 0

bl1:
        li   a0, GPIO_LEDs
        sw   t0, 0(a0)                  # Write to LEDs
        xori t0, t0, 1                  # invert LED
        and  t2, zero, zero             # Reset timer

time1:                                  # Delay loop
        addi t2, t2, 1
        bne  t1, t2, time1
        j    bl1
```

**Figure 28. blinky.S in PlatformIO**

**Step 10.** Click on  to run and debug the program; then start debugging by clicking on the play button RUN ▷ PIO Debug. PlatformIO sets a temporary breakpoint at the beginning of the main function. So, click on the Continue button to run the program.

**Step 11.** On the board, you will see the right-most LED start to blink.

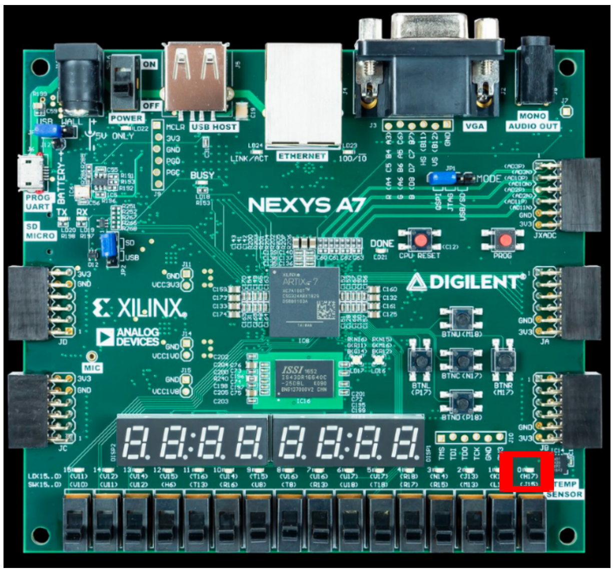
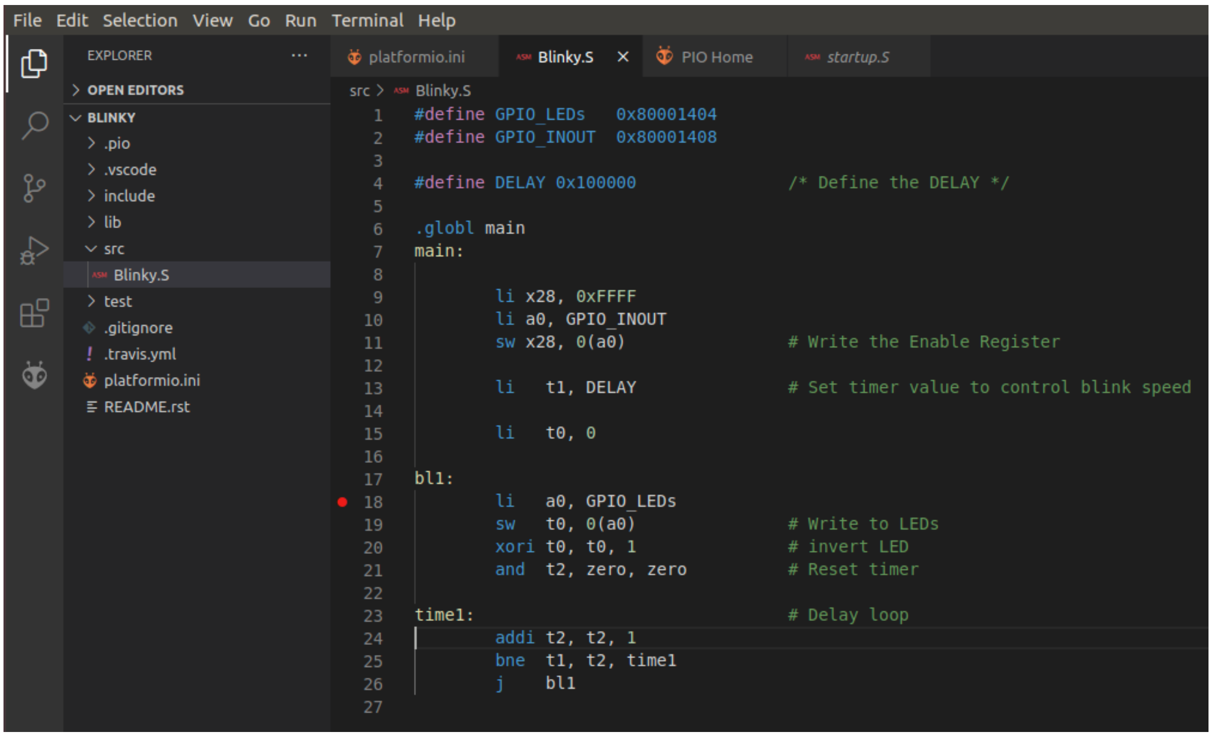


**Figure 29. rightmost LED Blinking**

**Step 12.** Pause the execution by clicking on the pause button . The execution will stop somewhere inside the infinite loop (probably, inside the `time1` delay loop).

**Step 13.** 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 30).



```
#define GPIO_LEDs   0x80001404
#define GPIO_INOUT  0x80001408

#define DELAY 0x100000              /* Define the DELAY */

.globl main
main:

        li x28, 0xFFFF
        li a0, GPIO_INOUT
        sw x28, 0(a0)               # Write the Enable Register

        li   t1, DELAY              # Set timer value to control blink speed

        li   t0, 0

bl1:
        li   a0, GPIO_LEDs
        sw   t0, 0(a0)              # Write to LEDs
        xori t0, t0, 1              # invert LED
        and  t2, zero, zero         # Reset timer

time1:                              # Delay loop
        addi t2, t2, 1
        bne  t1, t2, time1
        j    bl1
```

**Figure 30. Setting a breakpoint in blinky.S**

**Step 14.** Then, continue execution by clicking on the Continue button . Execution will continue, and it will stop after the store word (`sw`) instruction, which writes 1 (or 0) to the right-most LED.

**Step 15.** Continue execution several times; you will see that the value-driven to the right-most LED changes each time.

**Step 16.** Stop debugging and go back to the Explorer window by clicking on . Close the program by selecting *File → Close Folder.*

## 10. Debugging SweRVolf

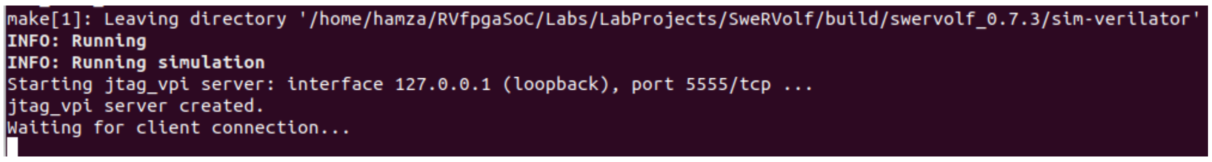
SweRVolf supports debugging both on hardware and in simulation. There are different procedures on how to connect the debugger, but once connected, the same commands can be used (although programs run much slower in simulation).

### 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.



```
make[1]: Leaving directory '/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Starting jtag_vpi server: interface 127.0.0.1 (loopback), port 5555/tcp ...
jtag_vpi server created.
Waiting for client connection...
```

**Figure 31. Run the simulation command**

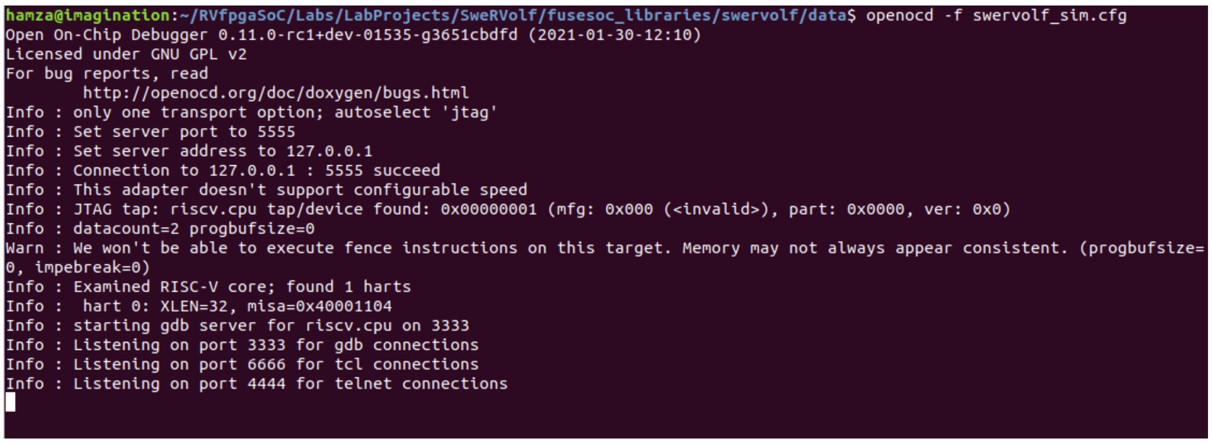
**Step 2**. Open a new terminal using “Ctrl + Shift + t”, then from the Workspace directory, navigate to the data directory by running:

➢ `cd fusesoc_libraries/swervolf/data/`

**Step 3.** 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 32).

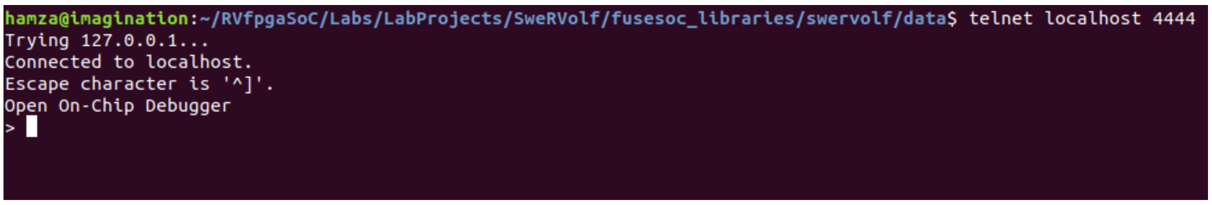


```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/fusesoc_libraries/swervolf/data$ openocd -f swervolf_sim.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Info : Set server port to 5555
Info : Set server address to 127.0.0.1
Info : Connection to 127.0.0.1 : 5555 succeed
Info : This adapter doesn't support configurable speed
Info : JTAG tap: riscv.cpu tap/device found: 0x00000001 (mfg: 0x000 (<invalid>), part: 0x0000, ver: 0x0)
Info : datacount=2 progbufsize=0
Warn : We won't be able to execute fence instructions on this target. Memory may not always appear consistent. (progbufsize=
0, impebreak=0)
Info : Examined RISC-V core; found 1 harts
Info :  hart 0: XLEN=32, misa=0x40001104
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

**Figure 32. Run OpenOCD**

**Step 4**. Open a third terminal using “Ctrl + Shift + t” and connect to the debug session through OpenOCD.

➢ `telnet localhost 4444`



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/fusesoc_libraries/swervolf/data$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
>
```

**Figure 33. telnet**

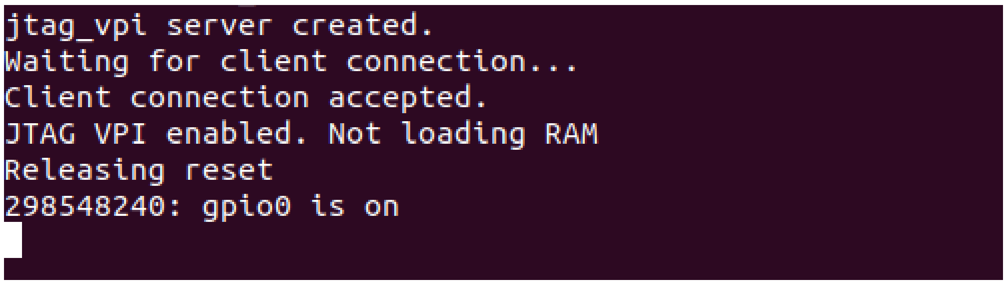
**Step 5**. Now enter commands to give instructions. We can now provide live instructions to SweRVolf through the third terminal.

In this FuseSoC based SoC, 16 LEDs are controlled by memory-mapped GPIO at address **`0x80001010-0x80001011.`** These addresses are different from the RVfpga system 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 34).

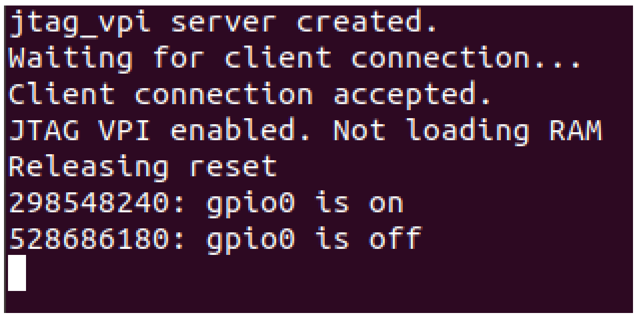


```
jtag_vpi server created.
Waiting for client connection...
Client connection accepted.
JTAG VPI enabled. Not loading RAM
Releasing reset
298548240: gpio0 is on
```

**Figure 34. 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
```



```
jtag_vpi server created.
Waiting for client connection...
Client connection accepted.
JTAG VPI enabled. Not loading RAM
Releasing reset
298548240: gpio0 is on
528686180: gpio0 is off
```

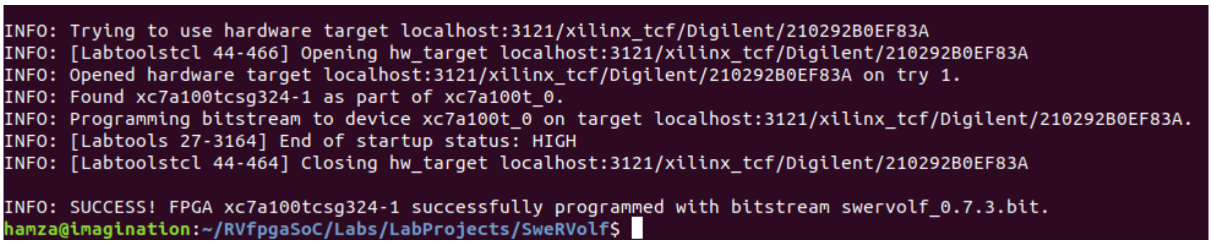
**Figure 35. gpio0 is OFF.**

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
```



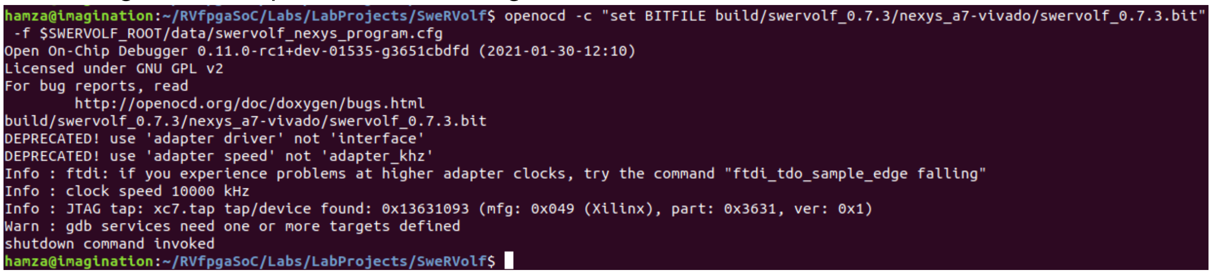
```
INFO: Trying to use hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: [Labtoolstcl 44-466] Opening hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A
INFO: Opened hardware target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A on try 1.
INFO: Found xc7a100tcsg324-1 as part of xc7a100t_0.
INFO: Programming bitstream to device xc7a100t_0 on target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A.
INFO: [Labtools 27-3164] End of startup status: HIGH
INFO: [Labtoolstcl 44-464] Closing hw_target localhost:3121/xilinx_tcf/Digilent/210292B0EF83A

INFO: SUCCESS! FPGA xc7a100tcsg324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

Figure 36. 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 37.

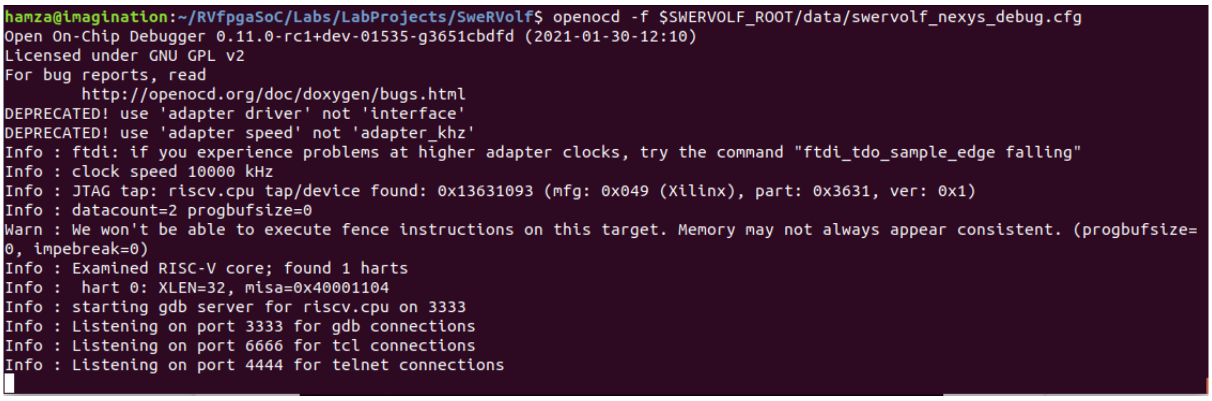


```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit"
 -f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
shutdown command invoked
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

Figure 37. Run OpenOCD

**Step 3.** Connect OpenOCD to SweRVolf

➢ `openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg`

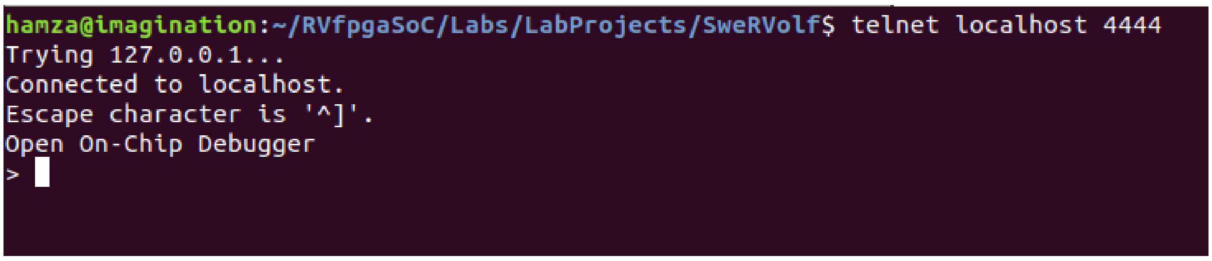


```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -f $SWERVOLF_ROOT/data/swervolf_nexys_debug.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651cbdfd (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: riscv.cpu tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Info : datacount=2 progbufsize=0
Warn : We won't be able to execute fence instructions on this target. Memory may not always appear consistent. (progbufsize=
0, impebreak=0)
Info : Examined RISC-V core; found 1 harts
Info :  hart 0: XLEN=32, misa=0x40001104
Info : starting gdb server for riscv.cpu on 3333
Info : Listening on port 3333 for gdb connections
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
```

Figure 38. OpenOCD Connected

**Step 4**. Open a third terminal using "Ctrl + Shift + t" and connect to the debug session through OpenOCD:

➢ `telnet localhost 4444`



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Open On-Chip Debugger
>
```

Figure 39. telnet

**Step 5**. 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.

To quickly recap this Lab, we first compared the Block Design code with the Hand-tuned code. Then we were introduced to FuseSoC, a package manager containing the SweRVolf (FuseSoC based SoC for SweRV Eh1) package.

We then explored its two versions, "SweRVolf Sim" and "SweRVolf Nexys".

Like the **RVfpgaSim** in the previous lab, **SweRVolf Sim** is a simulation target that wraps the SweRVolf core in a testbench to be used by the verilator. Similarly, the **RVfpgaNexys** in the previous lab is similar to the **SweRVolf Nexys** that is a version of the SweRVolf SoC targeted to the Digilent Nexys A7 board.

We then built 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 on RVfpgaSim and RVfpgaNexys.