



**THE IMAGINATION UNIVERSITY PROGRAMME**

# **RVfpga-SoC Lab 5**

## **Running Tensorflow Lite on SweRVolf**

**Table 1. RVfpga Terms**

Name		Description
<b>Courses</b>		
<b>RVfpga</b>		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).
<b>RVfpga-SoC</b>		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.
<b>Cores and SoCs</b>		
<b>SweRV EH1 Core</b>		Open-source commercial RISC-V core developed by Western Digital ( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).
<b>SweRV EH1 Core Complex</b>		SweRV EH1 core with added memory (ICCM, DCCM, and instruction cache), programmable interrupt controller (PIC), bus interfaces, and debug unit ( <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ).
<b>SweRVolfX</b>		The System on Chip that we use in the RVfpga course. It is an extension of SweRVolf. <b>SweRVolf</b> ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ): 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. <b>SweRVolfX</b> : It adds four new peripherals to SweRVolf: a GPIO, a PTC, an additional SPI, and a controller for the 8 Digit 7-Segment Displays.
<b>RVfpgaNexys</b>		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. RVfpgaNexys is the same as SweRVolf Nexys ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ), except that the latter is based on SweRVolf.
<b>RVfpgaSim</b>		The SweRVolfX SoC with a testbench wrapper and AXI memory intended for simulation. RVfpgaSim is the same as SweRVolf Sim, ( <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ), except that the latter is based on SweRVolf.

## 1. Introduction

In this Lab, we show how to build a Tensorflow Lite project for Zephyr (a real-time operating system) and then run that Zephyr program on SweRVolf. Similar to what we have seen in the previous lab, we will be running a Tensorflow program on top of Zephyr instead of a basic C or Assembly language program.

### 1. Brief background of TensorFlow Lite

TensorFlow Lite is a set of tools that enables on-device machine learning by helping developers run their models on mobile, embedded, and IoT devices. It compresses a TensorFlow model to a .tflite model that has a small binary size. This enables on-device machine learning and uses hardware acceleration to improve performance.

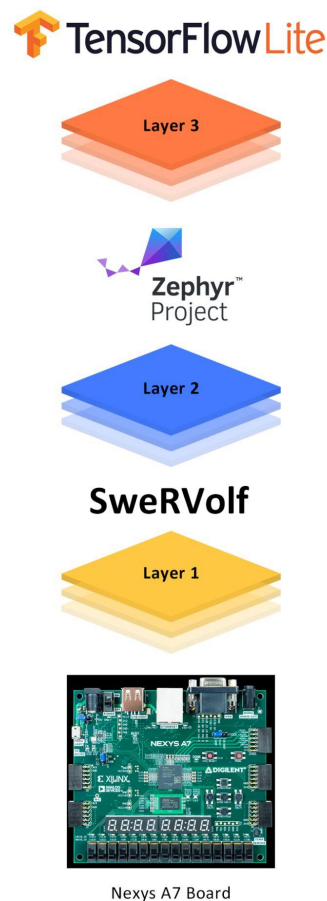
Its key features are:

- Optimized for on-device machine learning by addressing five key constraints: latency (there's no round-trip to a server), privacy (no personal data leaves the device), connectivity (internet connectivity is not required), size (reduced model and binary size), and power consumption (efficient inference and a lack of network connections).
- Multiple platform support, covering Android and iOS devices, embedded Linux, and microcontrollers.
- Diverse language support includes Java, Swift, Objective-C, C++, and Python.
- High performance, with hardware acceleration and model optimization.
- End-to-end examples for common machine learning tasks such as image classification, object detection, pose estimation, question answering, text classification, etc., on multiple platforms.

For more information, visit <https://www.tensorflow.org/lite/microcontrollers>

## SweRVolf and Tensorflow Lite

Figure 1 illustrates the hierarchical layers on top of the Nexys A7 board that we will implement in this Lab.



**Figure 1. Layers on top of the FPGA board**

The steps for running a TensorFlow Lite program on the Nexys A7 board are subtly different from the ones in Lab 4.

### Step 1. Download SweRVolf onto the FPGA board

First, we download the SweRVolf, the RISC-V system targeted to an FPGA, to the Nexys A7 FPGA board. We download the SweRVolf onto the board by either uploading the bitstream to the board using PlatformIO or by using the FuseSoC run command, which uploads the generated bitstream to the board if it's connected.

### Step 2. Build Tensorflow programs

In this step, we build a Tensorflow Lite application for Zephyr. The Zephyr RTOS is built as part of this build. The output is an elf file.

### Step 3. Load programs on SweRVolf.

In this step, we load the elf file generated during Step 2 onto SweRVolf.

## 2. Requirements

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

- Vivado 2019.2 Web Pack (Refer to Installation Guide (Page No.04))
- Verilator (v4.106) (Refer to Installation Guide (Page No.08))
- FuseSoC (Refer to Installation Guide (Page No.08))
- OpenOCD (RISC-V-specific version) (Refer to Installation Guide (Page No.09))
- Zephyr Prerequisites (Refer to Installation Guide (Page No.09))
- Zephyr SDK (v0.12.4) (Refer to Installation Guide (Page No.10))
- PuTTY (Refer to Installation Guide (Page No.10))

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

For example, if you have not already, install Xilinx's Vivado and Verilator 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. Tensorflow's Hello World Example

In this Lab, we will only set up the Tensorflow environment and run a simple Hello-World tensor operation.

The Hello World example is designed to demonstrate the absolute basics of using TensorFlow Lite for Microcontrollers. This program trains and runs a model that replicates a sine function, i.e., it takes a single number as its input and outputs the number's sine value.

For more information, visit TensorFlow's official documentation at this [link](#).

## 4. Setting up The Environment For Tensorflow

Open your Ubuntu terminal and complete the following steps:

**Step 1.** Navigate to the directory "**SweRVolf**". We have to set the following shell variables. To do that, we run the following:

- `export WORKSPACE=$(pwd)`
- `export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf`
- `export ZEPHYR_BASE=$WORKSPACE/zephyr`

You can also enter "`printenv <variable-name>`" command in the terminal window to verify if the shell variables have been successfully set or not.

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export WORKSPACE=$(pwd)
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export SWERVOLF_ROOT=$(pwd)/fusesoc_libraries/swervolf
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export ZEPHYR_BASE=$WORKSPACE/zephyr
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 2. Set the shell variables**

**Step 2.** Clone the Tensorflow GitHub repository.

➤ `git clone https://github.com/tensorflow/tensorflow`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ git clone https://github.com/tensorflow/tensorflow
Cloning into 'tensorflow'...
remote: Enumerating objects: 1155704, done.
remote: Counting objects: 100% (193/193), done.
remote: Compressing objects: 100% (141/141), done.
remote: Total 1155704 (delta 62), reused 118 (delta 51), pack-reused 1155511
Receiving objects: 100% (1155704/1155704), 683.05 MiB | 157.00 KiB/s, done.
Resolving deltas: 100% (942622/942622), done.
Checking out files: 100% (25049/25049), done.
```

**Figure 3. Tensorflow**

Now navigate to the “tensorflow” directory.

➤ `cd tensorflow`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd tensorflow/
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow$
```

**Figure 4. Navigate to the “tensorflow” directory**

Check out the “v2.5.0” branch of the repository by the following command :

➤ `git checkout -b v2.5.0`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow$ git checkout -b v2.5.0
Switched to a new branch 'v2.5.0'
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow$
```

**Figure 5. git checkout**

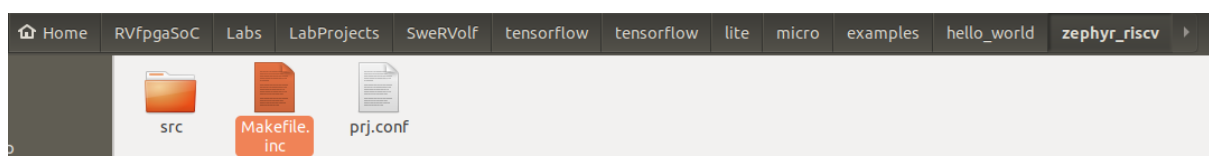
**Step 3.** In order to add support for Zephyr SweRVolf in TensorFlow, we must copy a couple of files in this TensorFlow repository.

The first file is the “**Makefile.inc**” file for the hello\_world example. Navigate to the following path to copy “**Makefile.inc**”:

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab5/Makefile.inc

Now paste the “**Makefile.inc**” file to the following location (see Figure 6)

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/lite/micro/examples/hello\_world/zephyr\_riscv/



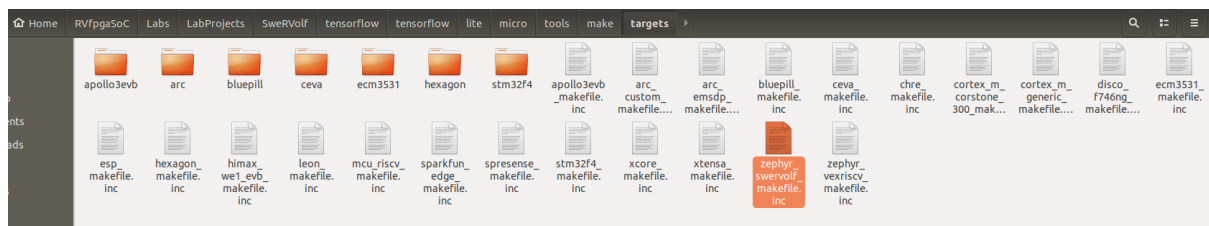
**Figure 6. Makefile.inc**

The second file is the “**zephyr\_swervolf\_makefile.inc**” file. Navigate to the following path to copy “**zephyr\_swervolf\_makefile.inc**”:

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab5/  
zephyr\_swervolf\_makefile.inc

Now paste the “**zephyr\_swervolf\_makefile.inc**” file to the following location (see Figure 7)

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/  
w/tensorflow/lite/micro/tools/make/targets/

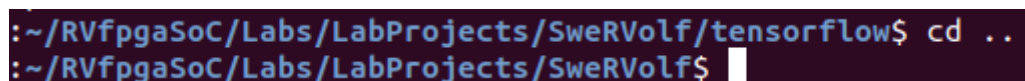


**Figure 7. zephyr\_swervolf\_makefile.inc**

**Step 4.** Install the required packages.

Navigate to the “**WORKSPACE**” directory using the following command:

➤ `cd ..`



```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow$ cd ..
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 8. Navigate to the WORKSPACE directory**

Install the required packages by the following command:

➤ `sudo apt install git cmake ninja-build gperf ccache dfu-util  
device-tree-compiler wget python python3-pip  
python3-setuptools python3-tk python3-wheel xz-utils file  
make gcc gcc-multilib locales tar curl unzip xxd make  
autoconf g++ flex bison virtualenv`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ sudo apt install git cmake ninja-build
gperf ccache dfu-util device-tree-compiler wget python python3-pip python3-setuptools python3-tk
python3-wheel xz-utils file make gcc gcc-multilib locales tar curl unzip xxd make autoconf g++
flex bison virtualenv
[sudo] password for hamza:
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version (2.69-11).
bison is already the newest version (2:3.0.4.dfsg-1build1).
ccache is already the newest version (3.4.1-1).
device-tree-compiler is already the newest version (1.4.5-3).
flex is already the newest version (2.6.4-6).
make is already the newest version (4.1-9.1ubuntu1).
python is already the newest version (2.7.15~rc1-1).
python3-setuptools is already the newest version (39.0.1-2).
xz-utils is already the newest version (5.2.2-1.3).
dfu-util is already the newest version (0.9-1).
```

**Figure 9. Install packages**

**Step 5.** Create a virtual environment.

First Navigate to the zephyr directory using the following command:

➤ `cd zephyr`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd zephyr/
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$
```

**Figure 10. Navigate to the zephyr directory**

Create a virtual environment inside the zephyr directory using the following command:

➤ `virtualenv venv-zephyr`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$ virtualenv venv-zephyr
created virtual environment CPython3.6.9.final.0-64 in 374ms
creator CPython3Posix(dest=/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/venv-zephyr, clear=False, no_vcs_ignore=False, global=False)
seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=/home/hamza/.local/share/virtualenv)
added seed packages: pip==21.0.1, setuptools==54.2.0, wheel==0.36.2
activators BashActivator,CShellActivator,FishActivator,PowerShellActivator,PythonActivator,XonshActivator
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$
```

**Figure 11. Creating venv-zephyr**

**Step 6.** Enter the following command to activate the virtual environment created in the last step.

➤ `source venv-zephyr/bin/activate`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$ source venv-zephyr/bin/activate
(venv-zephyr) hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$
```

**Figure 12. Activating venv-zephyr**

**Step 7.** Install the required packages listed in the “requirements.txt” file using the following command.

➤ `pip3 install -r scripts/requirements.txt`



```
(venv-zephyr) hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$ pip3 install
-r scripts/requirements.txt
Ignoring windows-curses: markers 'sys_platform == "win32"' don't match your environment
Collecting pyelftools>=0.26
  Using cached pyelftools-0.27-py2.py3-none-any.whl (151 kB)
Collecting PyYAML>=5.1
  Using cached PyYAML-5.4.1-cp36-cp36m-manylinux1_x86_64.whl (640 kB)
Collecting canopen
  Using cached canopen-1.2.1-py3-none-any.whl (52 kB)
Collecting packaging
  Using cached packaging-20.9-py2.py3-none-any.whl (40 kB)
Collecting progress
```

**Figure 13. Installing required packages**

Now we can close this terminal tab and return to our main terminal tab, where we will be building the “hello\_world” example.

## 5. Building Hello World Example for Swervolf

In this section, we will be building the “hello\_world” example for SweRVolf. We will be generating the “zephyr.bin” and “zephyr.elf” files for the “hello\_world” example.

**Step 1.** First, we will navigate to the tensorflow directory.

```
> cd ../tensorflow/
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr$ cd ../tensorflow/
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow$
```

**Figure 14. Navigating to the “tensorflow” directory**

**Step 2.** For this Lab, we are going to build Hello world for SweRVolf. This is done with the following command:

```
> make -f tensorflow/lite/micro/tools/make/Makefile
TARGET=zephyr_swervolf BUILD_TYPE=debug hello_world_bin
```

```
(venv-zephyr) hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow$ make -f tensorflow/lite/micro/tools/make/Makefile
TARGET=zephyr_swervolf BUILD_TYPE=debug hello_world_bin
--2021-05-20 19:44:08-- http://mirror.tensorflow.org/github.com/google/flatbuffers/archive/dca12522a9f9e37f126ab925fd385c807ab4f84e.zip
Resolving mirror.tensorflow.org (mirror.tensorflow.org)... 216.58.210.80, 2a00:1450:4018:803::2010
Connecting to mirror.tensorflow.org (mirror.tensorflow.org)|216.58.210.80|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1760478 (1.7M) [application/zip]
Saving to: '/tmp/tmp.kvIHESzFqo/dca12522a9f9e37f126ab925fd385c807ab4f84e.zip'

/tmp/tmp.kvIHESzFqo/dca12522a9f9 100%[=====] 1.68M 1.83MB/s in 0.9s

2021-05-20 19:44:09 (1.83 MB/s) - '/tmp/tmp.kvIHESzFqo/dca12522a9f9e37f126ab925fd385c807ab4f84e.zip' saved [1760478/1760478]

Cloning into 'tensorflow/lite/micro/tools/make/downloads/pigweed'...
remote: Sending approximately 15.02 MiB ...
remote: Counting objects: 33, done
remote: Finding sources: 100% (33/33)
remote: Total 22687 (delta 9753), reused 22681 (delta 9753)
Receiving objects: 100% (22687/22687), 15.01 MiB | 1.78 MiB/s, done.
Resolving deltas: 100% (9753/9753), done.
Note: checking out '47268dff45019863e20438ca3746c6c2df6ef09'.
```

**Figure 15. Building hello\_world example**

This will take a few minutes since it has to download some toolchains for the dependencies. Once it has finished, you should see some folders created inside a path like

- tensorflow/lite/micro/tools/make/gen/zephyr\_swervolf\_x86\_64\_debug/hello\_world/

These folders contain the generated project and source files.

```
[ 98%] Building C object zephyr/CMakeFiles/zephyr_final.dir/misc/empty_file.c.obj
[ 98%] Building C object zephyr/CMakeFiles/zephyr_final.dir/isr_tables.c.obj
[100%] Linking CXX executable zephyr.elf
Generating files from zephyr.elf for board: swervolf_nexys
make[3]: Leaving directory '/home/hanza/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/
tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build'
[100%] Built target zephyr_final
make[2]: Leaving directory '/home/hanza/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/
tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build'
make[1]: Leaving directory '/home/hanza/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/
tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build'
```

**Figure 16. Building hello\_world example completed**

The resulting binaries (zephyr.bin and zephyr.elf) will be generated in the following path:

- tensorflow/lite/micro/tools/make/gen/zephyr\_swervolf\_x86\_64\_debug/hello\_world/build/zephyr

**Step 3.** Now you can exit out of the virtual environment by entering the following command:

➤ deactivate

## 6. Running Hello World Example on Verilator

In this section, we will be converting the “zephyr.bin” file into a “.hex” file and then load it in as the initial ram file while running the simulator for SweRVolf.

**Step 1.** Navigate to the “hello\_world” project directory. Enter the following command to enter that directory:

➤ cd  
tensorflow/lite/micro/tools/make/gen/zephyr\_swervolf\_x86\_64\_debug/hello\_world/

```
/tensorflow$ cd tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/
/tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world$
```

**Figure 17. “hello\_world” project path**

**Step 2.** Convert the “.bin” file to the “.hex” file. To create the “.hex” file, run the following command from the hello\_world directory :

➤ python3 \$SWERVOLF\_ROOT/sw/makehex.py build/zephyr/zephyr.bin  
>  
/home/<Username>/RVfpgaSoC/Labs/LabProjects/SweRVolf/hello\_world\_tensorflow.hex

```
hanza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world
$ python3 $SWERVOLF_ROOT/sw/makehex.py build/zephyr/zephyr.bin > /home/hanza/RVfpgaSoC/Labs/LabProjects/SweRVolf/hello_world_tensorflow.hex
hanza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world
$
```

**Figure 18. convert “.bin” to “.hex”**

**Step 3.** Navigate back to the “WORKSPACE” directory

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/lite/micro/tools/nake/gen/zephyr_swervolf_x86_64_debug/hello_world$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

**Figure 19. Installing packages in venv-zephyr**

**Step 4.** Load the “.hex” file in the simulator:

```
> fusesoc run --target=sim swervolf
--ram_init_file=hello_world_tensorflow.hex
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf --ram_init_file=hello_world_tensorflow.hex
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
```

**Figure 20. Loading “.hex” file in the simulator**

We can see the output of the hello\_world example (see Figure 21). The program prints out the X and Y values of the “sine” function.

```
g++ jtagServer.o jtag_common.o tb.o verilated.o verilated dpi.o verilated vcd.c.o Vswervolf_core_tb_ALL.a -o Vswervolf_core_tb
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 /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/hello_world_tensorflow.hex
Releasing reset
*** Booting Zephyr OS build zephyr-v2.4.0 ***
x_value: 1.0*2^-127, y_value: 1.0*2^-127

x_value: 1.2566366*2^-2, y_value: 1.4910772*2^-2
x_value: 1.2566366*2^-1, y_value: 1.1183078*2^-1
x_value: 1.8849551*2^-1, y_value: 1.677462*2^-1
x_value: 1.2566366*2^0, y_value: 1.9316229*2^-1
```

**Figure 21. “hello\_world” output**

Press “ctrl + c” to exit out of the program.

## 7. Running Hello World Example on the Nexys A7 Board

In this section, we will be running the “hello\_world” project on the board using OpenOCD.

**Step 1.** Connect the Nexys A7 board to your computer and then run the FPGA build command in the Workspace directory.

```
> fusesoc run --target=nexys_a7 --run swervolf
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=nexys_a7 --run swervolf
WARNING: Unknown item compilation_mode in section Xsim
INFO: Running
export HW_TARGET=; \
export JTAG_FREQ=; \
vivado -quiet -nolog -notrace -mode batch -source swervolf_0.7.3_pgm.tcl -tclargs xc7a100tcs9324-1 swervolf_0.7.3.bit
Fusesoc Xilinx FPGA Programming Tool
=====
```

## Figure 22. Run the FPGA build

**Step 2.** Connect OpenOCD with 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 23. OpenOCD connected

**Step 3.** Open a new terminal using “Ctrl + Shift + t” & connect to the debug session through OpenOCD using the following command:

➤ `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 24. telnet localhost 4444

OpenOCD supports loading ELF program files by running `load_image /path/to/file.elf`. Remember that the path is relative to the directory from where OpenOCD was launched.

➤ `load_image tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build/zephyr/zephyr.elf`

```
> load_image tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build/zephyr/zephyr.elf
287072 bytes written at address 0x00000000
downloaded 287072 bytes in 27.439606s (10.217 KiB/s)
> █
```

## Figure 25. loading the “.elf” file

After the program has been loaded, set the program counter to address zero using the following command:

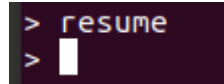
➤ `reg pc 0`

```
> reg pc 0
pc (/32): 0x00000000
```

## Figure 26. Set program counter to zero

Now start the program using this command:

➤ `resume`

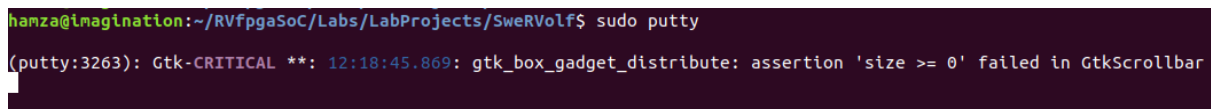


```
> resume
> 
```

**Figure 27. Start the program**

**Step 4.** Open a new terminal using “Ctrl + Shift + t”. Open “PuTTY” using the command

➤ `sudo putty`



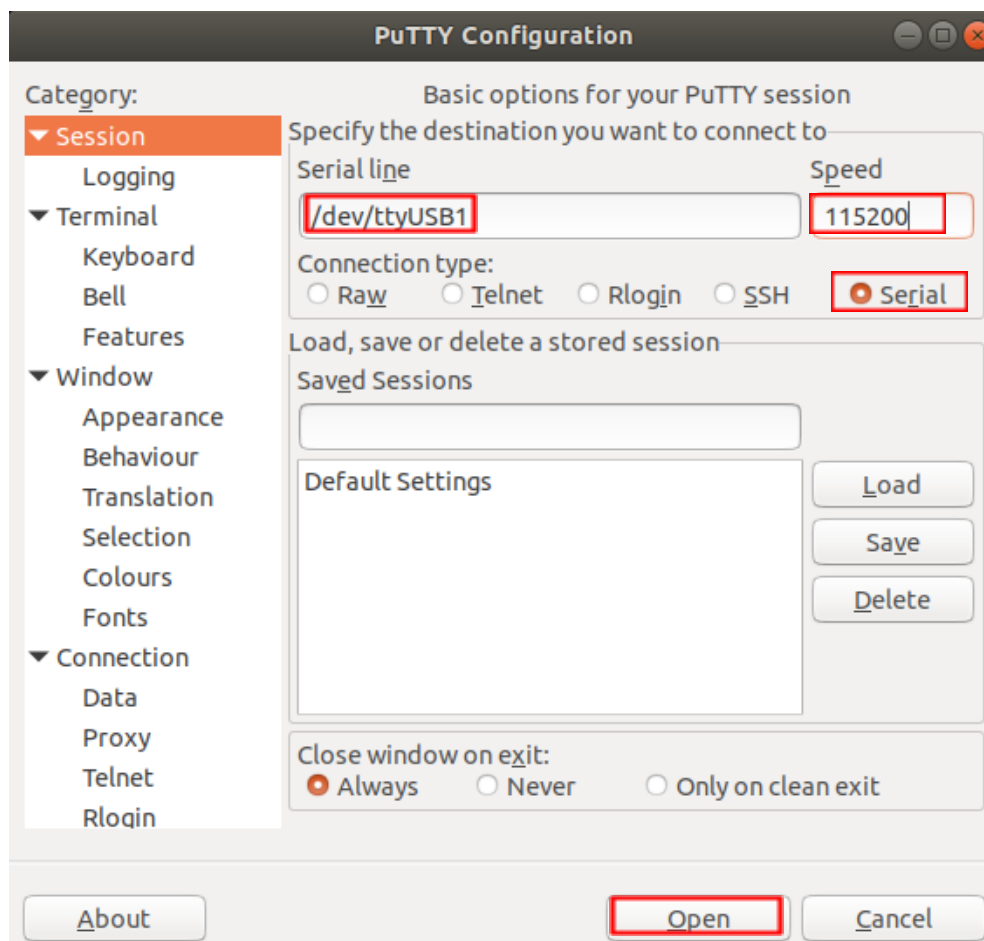
```
hanza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ sudo putty
(puTTY:3263): Gtk-CRITICAL **: 12:18:45.869: gtk_box_gadget_distribute: assertion 'size >= 0' failed in GtkScrollbar
```

**Figure 28. Open PuTTY**

We will be using PuTTY here as a serial console for our Nexys A7 board.

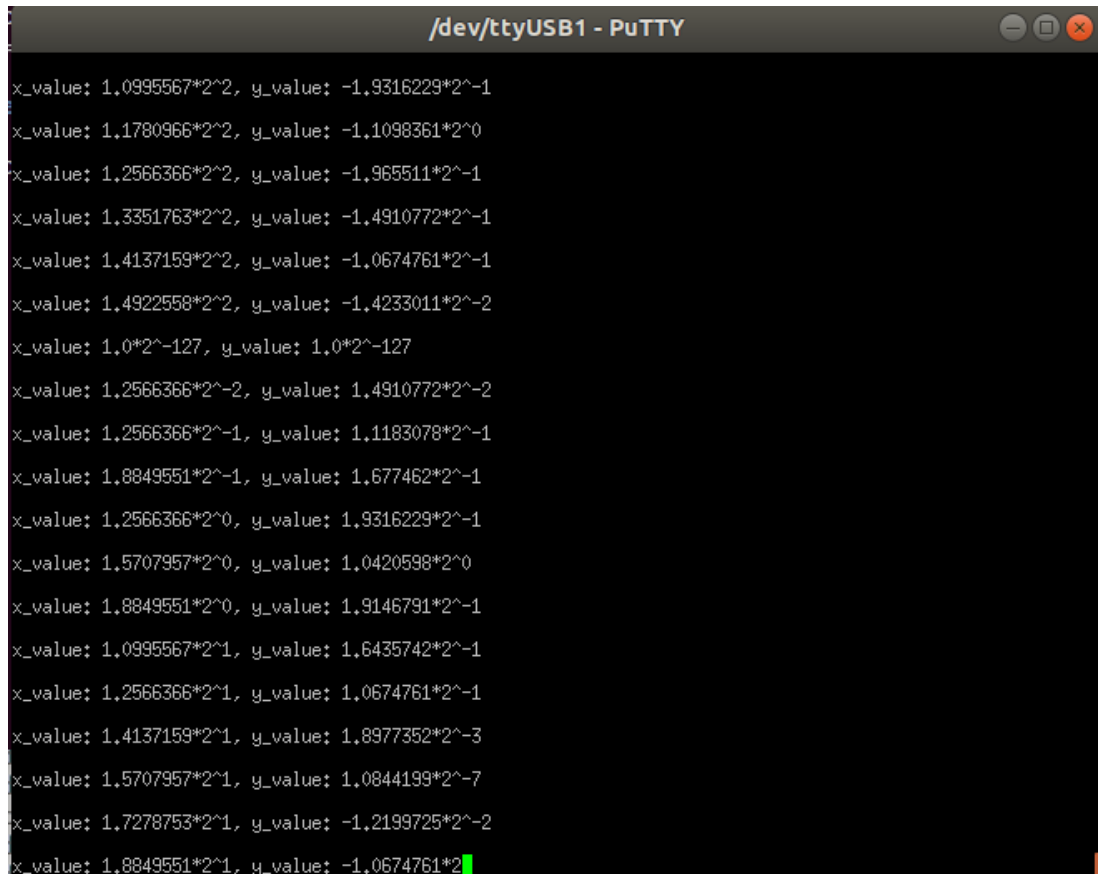
**Step 5.** Set the following configuration:

Select the connection type as “**Serial**”, then enter “**/dev/ttyUSB1**” as the serial line, and set the speed equal to “**115200**”. Now click “Open” to start the serial console.



**Figure 29. PuTTY configuration**

In the serial console, we can see the output of the hello\_world example (see Figure 30).



```

/dev/ttyUSB1 - PuTTY
x_value: 1.0995567*2^2, y_value: -1.9316229*2^-1
x_value: 1.1780966*2^2, y_value: -1.1098361*2^0
x_value: 1.2566366*2^2, y_value: -1.965511*2^-1
x_value: 1.3351763*2^2, y_value: -1.4910772*2^-1
x_value: 1.4137159*2^2, y_value: -1.0674761*2^-1
x_value: 1.4922558*2^2, y_value: -1.4233011*2^-2
x_value: 1.0*2^-127, y_value: 1.0*2^-127
x_value: 1.2566366*2^-2, y_value: 1.4910772*2^-2
x_value: 1.2566366*2^-1, y_value: 1.1183078*2^-1
x_value: 1.8849551*2^-1, y_value: 1.677462*2^-1
x_value: 1.2566366*2^0, y_value: 1.9316229*2^-1
x_value: 1.5707957*2^0, y_value: 1.0420598*2^0
x_value: 1.8849551*2^0, y_value: 1.9146791*2^-1
x_value: 1.0995567*2^1, y_value: 1.6435742*2^-1
x_value: 1.2566366*2^1, y_value: 1.0674761*2^-1
x_value: 1.4137159*2^1, y_value: 1.8977352*2^-3
x_value: 1.5707957*2^1, y_value: 1.0844199*2^-7
x_value: 1.7278753*2^1, y_value: -1.2199725*2^-2
x_value: 1.8849551*2^1, y_value: -1.0674761*2

```

**Figure 30. serial console**

Again as we saw in the simulation section, the program prints the “X” and “Y” coordinates of the sine function that the TensorFlow model is plotting.

**Note:** If you are unable to open a serial console, try “/dev/ttyUSB0” as the serial line.

So in this lab, we have successfully build the “hello\_world” example of TensorFlow as a Zephyr application and then run that example on SweRVolf.