

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga-SoC Lab 5**

**Running Tensorflow Lite on SweRVolf**

**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.  **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.  **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.  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.  RVfpgaSim is the same as SweRVolf sim, (<https://github.com/chipsalliance/Cores-SweRVolf>), except that the latter is based on SweRVolf. |

# 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.

**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.

# 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.09))
* OpenOCD (RISC-V-specific version) (Refer to Installation Guide (Page No.09))
* Zephyr Prerequisites (Refer to Installation Guide (Page No.10))
* Zephyr SDK (v0.12.4) (Refer to Installation Guide (Page No.10))
* PuTTY (Refer to Installation Guide (Page No.11))

**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.

# 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](https://www.tensorflow.org/lite/microcontrollers/get_started_low_level).

# Setting up The Environment For Tensorflow

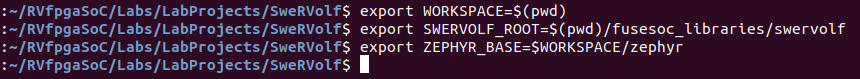
Open your Ubuntu terminal and complete the following steps :

**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

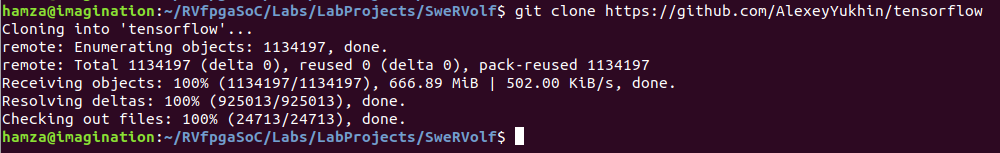
**$** export ZEPHYR\_BASE=$WORKSPACE/zephyr



**Figure 2. Set the shell variables**

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

**$** git clone https://github.com/AlexeyYukhin/tensorflow

****

**Figure 3. Tensorflow**

Now navigate to the “tensorflow” directory.

**$** cd tensorflow



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

Checkout the specific branch of the repository by the following command :

**$** git checkout -b adc570a50410be7aba1c33522854f45fa0f349be



**Figure 5. git checkout**

**Step 3.** Install the required packages.

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

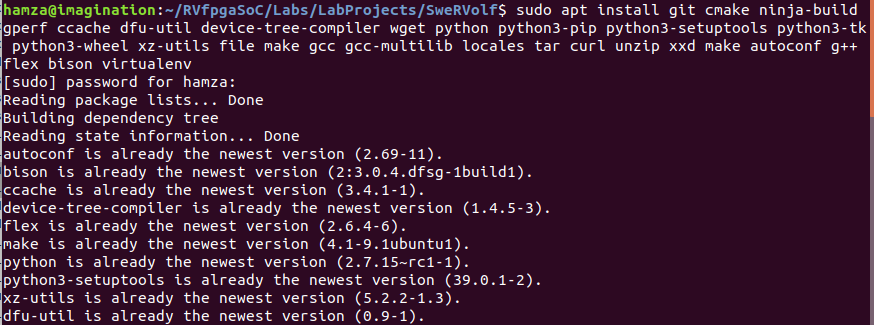
**$** cd ..



**Figure 6. 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



**Figure 7. Install packages**

**Step 4.** Create a virtual environment using the following command.

Navigate to the zephyr directory using the following command:

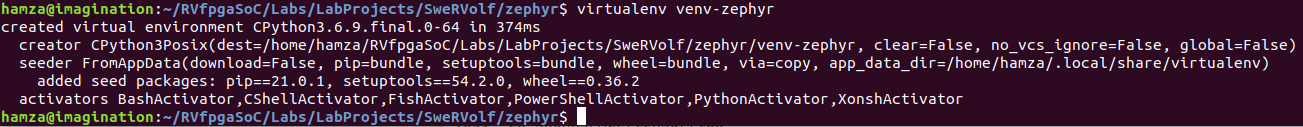
**$** cd zephyr

****

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

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

**$** virtualenv venv-zephyr



**Figure 9. Creating venv-zephyr**

**Step 5.** Enter the following command to activate the virtual environment created in the last step.

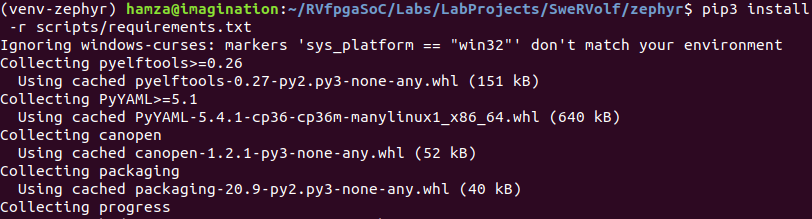
**$** source venv-zephyr/bin/activate



**Figure 10. Activating venv-zephyr**

**Step 6.** Install the required packages listed in the “scripts“ or requirements.txt” file using the following command.

**$** pip3 install -r scripts/requirements.txt



**Figure 11. 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.

# 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.

First, we will navigate to the tensorflow directory.

**$** cd ../tensorflow/



**Figure 12. Navigating to the “tensorflow” directory**

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

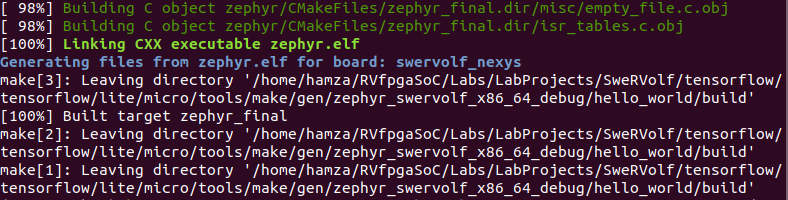


**Figure 13. Building hello\_world example**

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.



**Figure 14. 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*

# 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:

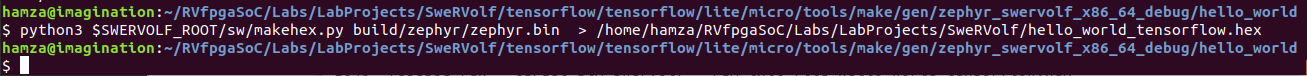
**$** cd tensorflow/lite/micro/tools/make/gen/zephyr\_swervolf\_x86\_64\_debug/hello\_world/



**Figure 15. “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/{YourUsername}/RVfpgaSoC/Labs/LabProjects/SweRVolf/hello\_world\_tensorflow.hex



**Figure 16. convert “.bin” to “.hex”**

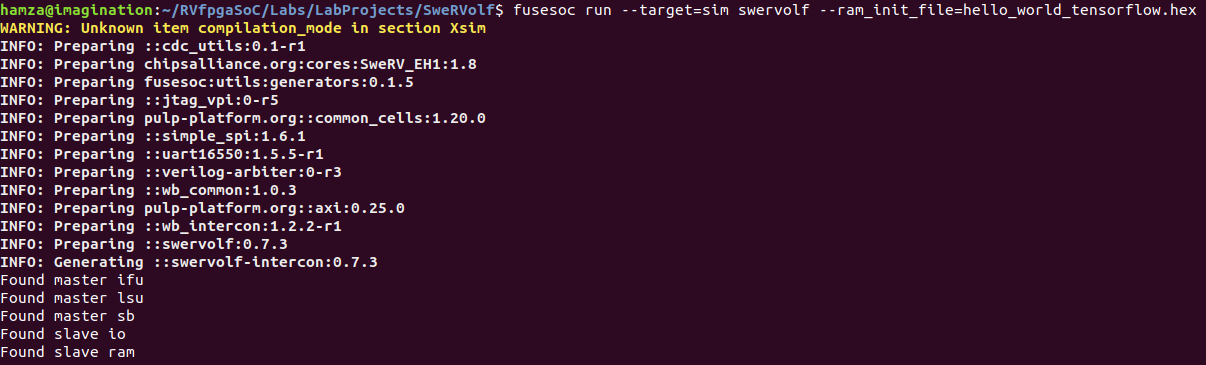
**Step 3.** Navigate back to the “WORKSPACE” directory



**Figure 17. 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



**Figure 18. Loading “.hex” file in the simulator**

We can see the output of the hello\_world example (See Figure 18).

The program prints out the X and Y values of the “sine” function.



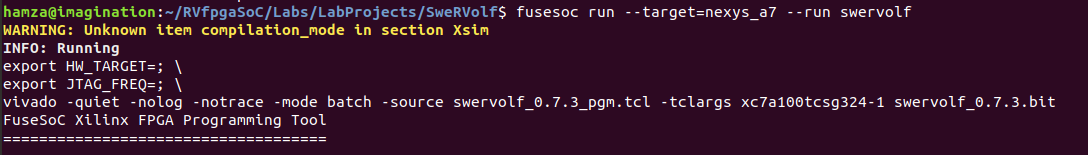
**Figure 19. “hello\_world” output**

# 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.

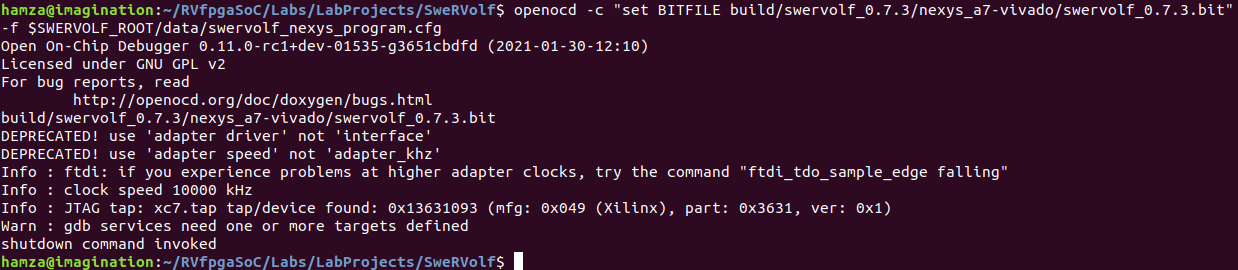
**$** fusesoc run --target=nexys\_a7 --run swervolf



**Figure 20. Run the FPGA build**

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

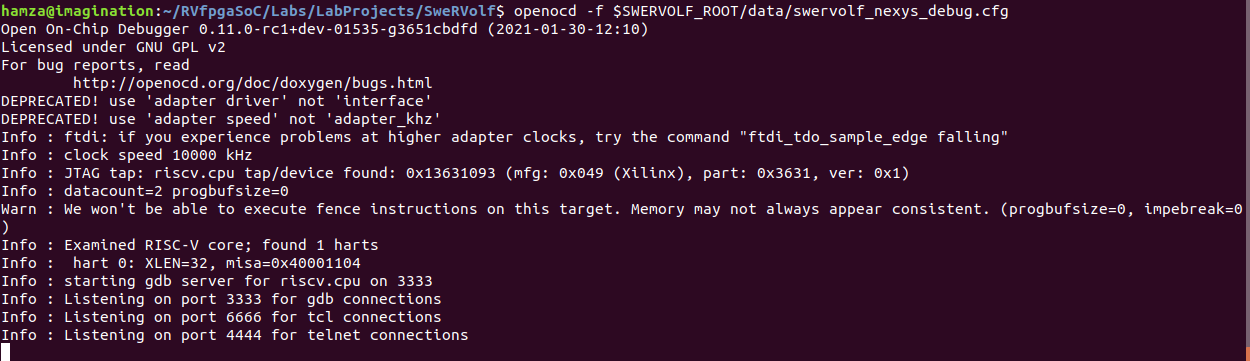
**$** openocd -c "set BITFILE build/swervolf\_0.7.3/nexys\_a7-vivado/swervolf\_0.7.3.bit" -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_program.cfg



**Figure 21. Run OpenOCD**

**Step 3.** Connect OpenOCD with SweRVolf.

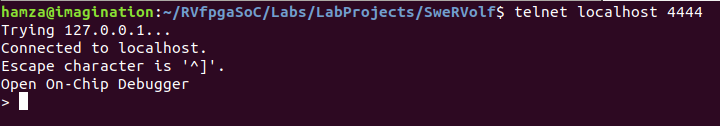
**$** openocd -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_debug.cfg



**Figure 22. OpenOCD connected**

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

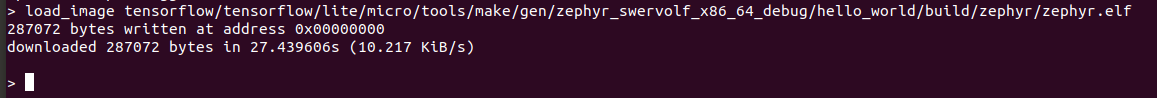
**$** telnet localhost 4444



**Figure 23. 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



**Figure 24. loading the “.elf” file**

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

**>** reg pc 0



**Figure 25. Set program counter to zero**

Now start the program using this command:

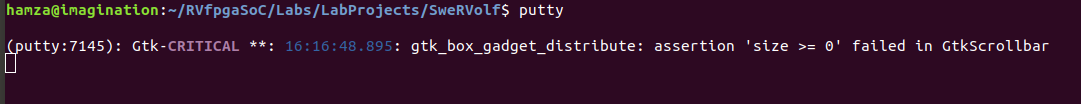
**>** resume



**Figure 26. Start the program**

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

**$** putty



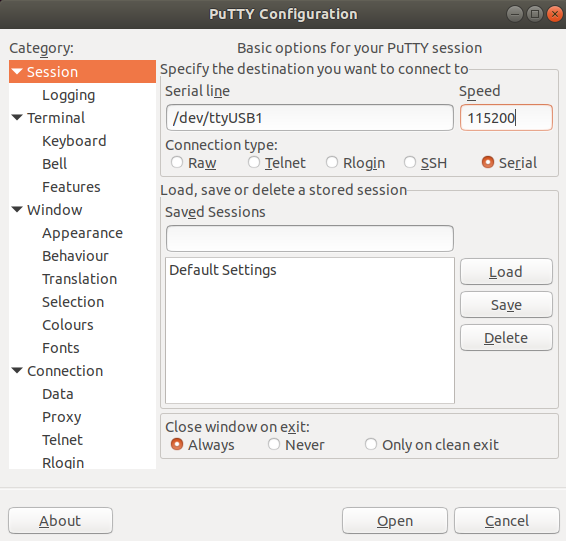
**Figure 27. Open PuTTY**

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

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

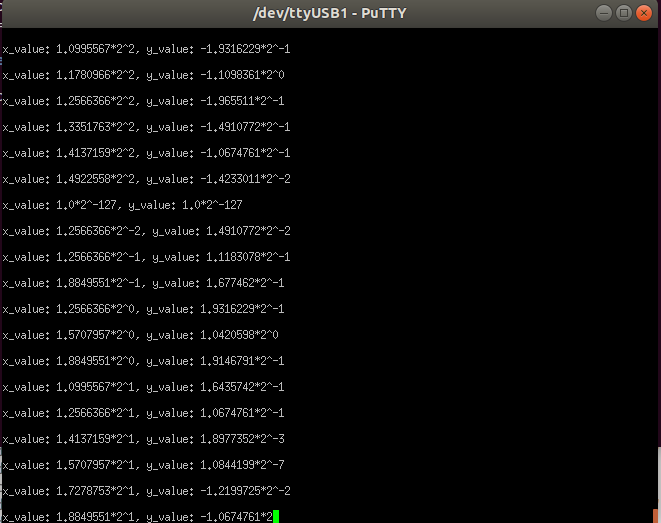
Select the connection type as “**Serial**”, then enter “**/dev/ttyUSB1**” as the serial line, and set the speed equal to “**115200**”.

Now click “Open” to start the serial console.



**Figure 28. PuTTY configuration**

In the serial console, we can see the output of the hello\_world example (See Figure 29).



**Figure 29. 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 running putty as an Administrator using “sudo” or try “/dev/ttyUSB0” as the serial line. |