

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga-SoCラボ5**

**SweRVolfでの  
Tensorflow Liteの実行**

**表1：RVfpga用語**

|  |  |
| --- | --- |
| **名前** | **説明** |
| **コース** | |
| **RVfpga** | プログラムを実行、および周辺機器を追加（RVfpgaラボ1～10）してシステムを拡張、およびシミュレーションを実行、性能を測定、命令を追加、メモリシステムを変更（RVfpgaラボ11～20）してコアとメモリシステムを調べるために、RVfpgaNexysとRVfpgaSim、RISC-V system-on-chips（SoCs）を使用する方法が示されているコース。このコース全体にわたって、RISC-Vツールチェイン（コンパイラおよびデバッガ）とシミュレータ、Verilator HDLシミュレータ、Western DigitalのWhisper命令セットシミュレータ（ISS）の使用方法も、示されています。 |
| **RVfpga-SoC** | SweRVコア、メモリ、周辺機器などの構成要素を使用して、ゼロからサブセットSweRVolfX SoCを構築する方法が示されているコース。このコースには、SweRVolfにZephyrリアルタイムオペレーティングシステム（RTOS）をロードする方法、およびオペレーティングシステムに加えてTensorflow Liteのhello world例が含まれているプログラムを実行する方法も、示されています。 |
| **コアおよびSoC** | |
| **SweRV EH1 コア** | Western Digital開発のオープンソース商用RISC-Vコア  （<https://github.com/chipsalliance/Cores-SweRV>）。 |
| **SweRV EH1 コア複合体** | 追加コアメモリ（ICCM、DCCM、命令キャッシュ）、プログラム可能割り込みコントローラ（PIC）、バスインターフェース、デバッグユニットが追加されれているSweRV EH1コア（<https://github.com/chipsalliance/Cores-SweRV>）。 |
| **SweRVolfX** | RVfpgaコースで使用するチップでのシステム。これはSweRVolfの拡張です。  **SweRVolf**（<https://github.com/chipsalliance/Cores-SweRVolf>）：SweRV EH1コア複合体周辺に構築されたオープンソースSoC。これにより、ブートROM、UARTインターフェース、システムコントローラ、インターコネクト（AXIインターコネクト、Wishboneインターコネクト、AXI-to-Wishboneブリッジ）、SPIコントローラが追加されます。  **SweRVolfX**：これにより次記の4つの新しい周辺機器がSweRVolfに追加されます：GPIO、PTC、追加のSPI、および8桁の7セグメント表示用コントローラ。 |
| **RVfpgaNexys** | Nexys A7ボードおよびその周辺機器を対象とするSweRVolfX SoC。これにより、DDR2インターフェース、CDC（クロックドメイン交差）ユニット、BSCANロジック（JTAGインターフェース用）、クロックジェネレータが追加されます。  RVfpgaNexysはSweRVolf Nexys（<https://github.com/chipsalliance/Cores-SweRVolf>）と同じですが、例外として後者はSweRVolfに基づいています。 |
| **RVfpgaSim** | シミュレーションを目的とした、テストベンチラッパおよびAXIメモリ付きSweRVolfX SoC。  RVfpgaSimはSweRVolf Sim（<https://github.com/chipsalliance/Cores-SweRVolf>）と同じですが、例外として後者はSweRVolfに基づいています。 |

# はじめに

このラボでは、Zephyr（リルタイムオペレーティングシステム）用Tensorflow Liteプロジェクトを構築する方法を示し、SweRVolfでZephyrプログラムを実行します。前のラボで見たのと同様に、Zephyr上で、基本的なC言語やアセンブリ言語のプログラムではなく、Tensorflowプログラムを実行します。

1. **TensorFlow Liteの概略の背景**

TensorFlowは、開発者がモバイル、デバイス、組み込みデバイス、IoTデバイスで自身のモデルを実行する役に立つことによって、オンデバイス機械学習が可能になるツールのセットです。これにより、TensorFlowモデルが小さなバイナリサイズの.tfliteモデルに、圧縮されます。これによってオンデバイス機械学習が可能になり、ハードウェアが加速されて性能が向上します。

その主要な機能は以下のとおりです。

* 次記の5つの主要な制約に対処することによるオンデバイス機械学習に対する最適化：待ち時間（サーバーとの往復なし）、プライバシー（個人データのデバイスからの漏出なし）、接続性（インターネットとの接続性不要）、サイズ（モデルサイズおよびバイナリサイズの低減）、電力消費（効率的な推論およびネットワーク接続なし）。
* 複数プラットフォームに対応、AndroidおよびiOSデバイスに対応、Linux組み込み、マイクロコントローラ。
* Java、Swift、Objective-C、C++、Pythonなどの多様な言語に対応。
* ハードウェアの加速およびモデルの最適化による高性能。
* 複数のプラットフォームでの、画像分類、オブジェクト検出、姿勢評価、質疑応答、テキスト分類、その他などの一般的機械学習タスク用のエンドツーエンドの例。

詳細については、<https://www.tensorflow.org/lite/microcontrollers>にアクセスしてください。

**SweRVolfおよびTensorflow Lite**

このラボで導入するNexys A7ボードの上の階層構造層が、図1に示されています。



**図1：FPGAボード上の層**

TensorFlow LiteプログラムをNexys A7ボードで実行するステップは、ラボ4のものと微妙に相違しています。

**ステップ1**：**SweRVolfをFPGAボードにダウンロードする**

まず、FPGAを対象とするRISC-VシステムであるSweRVolfを、Nexys A7 FPGAボードにダウンロードします。PlatformIOを使用してビットストリームをボードにアップロードするか、生成されたビットストリームをボード（接続されている場合）にアップロードするFuseSoC実行コマンドを使用して、SweRVolfをボードにダウンロードします。

**ステップ2：Tensorflowプログラムを構築する**

このステップで、Zephyr用アプリケーションTensorflow Liteを構築します。Zephyr RTOSは、この構築の一部として構築されます。出力はelfファイルです。

**ステップ3：SweRVolfにプログラムをロードする。**

このステップでは、ステップ2で生成されたelfファイルをSweRVolfにロードします。

# 要件

このラボを完了するには、以下をインストールする必要があります。

* Vivado 2019.2 Web Pack （インストールガイド（04ページ）を参照）
* Verilator（v4.106） （インストールガイド（08ページ）を参照）
* FuseSoC （インストールガイド（08ページ）を参照）
* OpenOCD（RISC-V固有のバージョン） （インストールガイド（09ページ）を参照）
* Zephyr必要条件 （インストールガイド（09ページ）を参照）
* Zephyr SDK（v0.12.4） （インストールガイド（10ページ）を参照）
* PuTTY （インストールガイド（10ページ）を参照）

**重要：**RVfpga-SoCラボを開始する前に、RVfpga-SoCインストールガイドを完了することを、強くお勧めします。

例えば、まだ完了していない場合は、XilinxのVivadoおよびVerilatorをRVfpga-SoCインストールガイドの手順に従ってインストールします。ImaginationのUniversity ProgrammeからダウンロードしたRVfpga-SoCフォルダをお使いのマシンにコピーしたことを、確認します。

# TensorflowのHello World例

このラボでは、Tensorflow環境のみをセットアップし、簡単なHello-Worldテンソル演算を実行します。

Hello World例は、マイクロコントーラ用TensorFlow Liteの初歩中の初歩の使用を実際に示すために、設計されています。このプログラムにより、正弦関数を反復して行うモデル（つまり、単一の数を入力として、その数値の正弦値をを出力する）をトレーニングおよび実行します。

詳細については、次記の[link](https://www.tensorflow.org/lite/microcontrollers/get_started_low_level)のTensorFlowの公式ドキュメントを参照してください。

# Tensorflowの環境のセットアップ

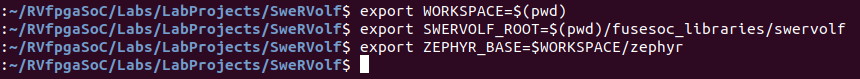
Ubuntuターミナルを開き、以下のコステップを実行します。

**ステップ1：**ディレクトリ「**SweRVolf**」に移動します。以下のシェル変数を設定する必要があります。そのためには、以下を実行します。

* export WORKSPACE=$(pwd)

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

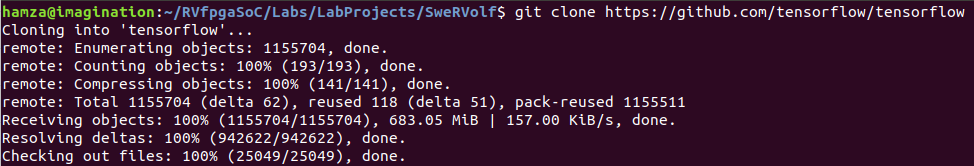
「printenv <variable-name>」コマンドをターミナルウィンドウに入力して、シェル変数が正常に設定されているかを、確認することもできます。



**図2：シェル変数を設定**

**ステップ2：**Tensorflow GitHubリポジトリをクローンします。

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

****

**図3：Tensorflow**

「tensorflow」ディレクトリに移動します。

* cd tensorflow



**図4：「tensorflow」ディレクトリに移動する**

次記のコマンドでリポジトリの「v2.5.0」分岐をチェックアウトします。

* git checkout -b v2.5.0



**図5：gitのチェックアウト**

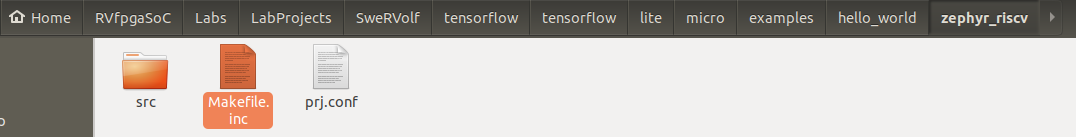
**ステップ3：**TensorFlowでZephyr SweRVolfのサポートを追加するため、このTensorFlowリポジトリの2つのファイルをコピーする必要があります。

1番目のファイルはhello\_world例用の「**Makefile.inc**」ファイルです。次記のパスに移動して、「**Makefile.inc**」をコピーします。

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

「**Makefile.inc**」ファイルを以下の場所に貼り付けます（図6を参照）。

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



**図6：Makefile.inc**

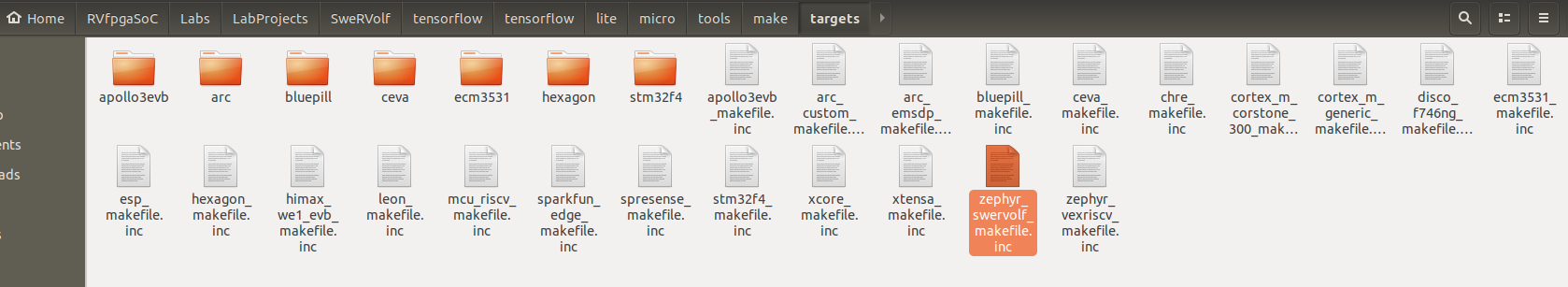
2番目のファイルは「**zephyr\_swervolf\_makefile.inc**」ファイルです。次記のパスに移動して、「**zephyr\_swervolf\_makefile.inc**」をコピーします。

* [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab5/

zephyr\_swervolf\_makefile.inc

「**zephyr\_swervolf\_makefile.inc**」ファイルを以下の場所に貼り付けます（図7を参照）。

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



**図7：zephyr\_swervolf\_makefile.inc**

**ステップ4：**必要なパッケージをインストールします。

以下のコマンドを使用して、「WORKSPACE」ディレクトリに移動します。

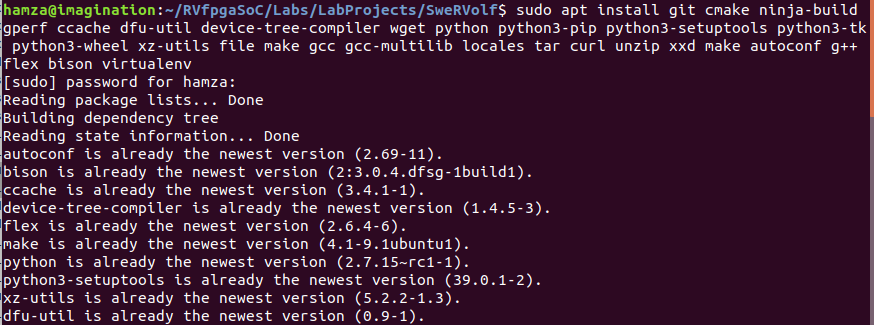
* cd ..



**図8：WORKSPACEディレクトリに移動する**

以下のコマンドで、必要なパッケージをインストールします。

* 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



**図9：パッケージをインストールする**

**ステップ5：**仮想環境を作成します。

まず、次記のコマンドを使用して、zephyrディレクトリに移動します。

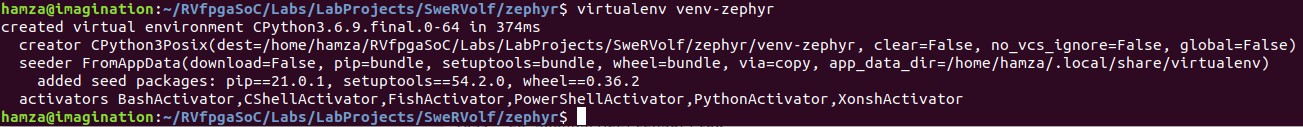
* cd zephyr

****

**図10：zephyrディレクトリに移動する**

次記のコマンドを使用して、zephyrディレクトリ内に仮想環境を作成します。

* virtualenv venv-zephyr



**図11：venv-zephyrの作成**

**ステップ6：**次記のコマンドを入力して、最後のステップで作成した仮想環境をアクティブにします。

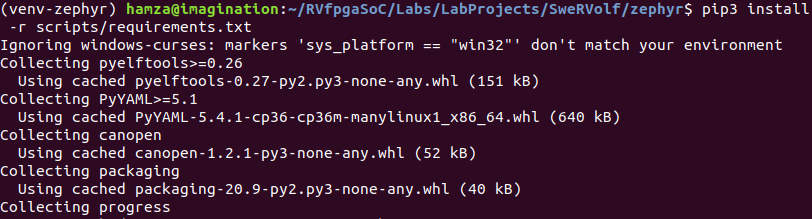
* source venv-zephyr/bin/activate



**図12：venv-zephyrのアクティブ化**

**ステップ7：**次記コマンドを使用して、「requirements.txt」ファイルにリストされている必要なパッケージをインストールします。

* pip3 install -r scripts/requirements.txt



**図13：必要なパッケージのインストール**

ここで、ターミナルタブを閉じて、メインターミナルタブに戻ることができます。そこで「hello\_world」例を構築します。

# Swervolf用のHello World例の構築

このセクションでは、SweRVolf用に「hello\_world」例を構築します。「hello\_world」例用に、「**zephyr.bin**」ファイルおよび「**zephyr.elf**」ファイルを生成します。

**ステップ1：**まず、tensorflowディレクトリに移動します。

* cd ../tensorflow/



**図14：「tensorflow」ディレクトリに移動する**

**ステップ2：**このラボでは、SweRVolf用にHello worldを構築します。これは、次記のコマンドで実行されます。

* make -f tensorflow/lite/micro/tools/make/Makefile TARGET=zephyr\_swervolf BUILD\_TYPE=debug hello\_world\_bin

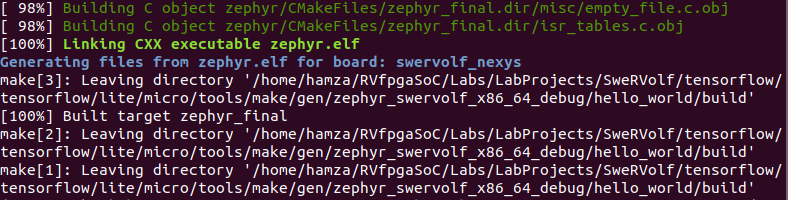


**図15：hello\_world例の構築**

これには数分かかります。その理由は、依存性のために一部のツールチェインをダウンロードする必要があるからです。これが完了すると、次記のようなパス内に作成された一部の  
フォルダが、表示されるはずです。

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

これらのフォルダには、生成されたプロジェクトファイルとソースファイルが含まれます。



**図16：hello\_world例構築完了**

結果のバイナリ（zephyr.binおよびzephyr.elf）が、次記のパスに生成されます。

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

**ステップ3：**次記のコマンドを入力して、仮想環境を終了できます。

* deactivate

# VerilatorでHello World例を実行する

このセクションでは、「zephyr.bin」ファイルを「.hex」ファイルに変換し、これをSweRVolf用のシミュレータ実行時の初期ramファイルとして、ロードします。

**ステップ1：**「hello\_world」プロジェクトディレクトリに移動します。そのディレクトリを入力するために、次記のコマンドを入力します。

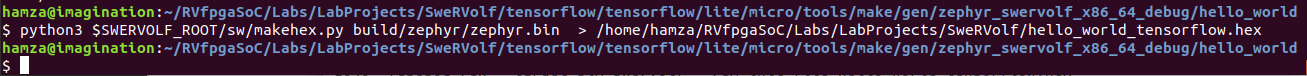
* cd tensorflow/lite/micro/tools/make/gen/zephyr\_swervolf\_x86\_64\_debug/hello\_world/



**図17：「hello\_world」プロジェクトパス**

**ステップ2：**「**.bin**」ファイルを「**.hex**」ファイルに変換します。「**.hex**」ファイルを作成するには、hello\_worldディレクトリから次記のコマンドを実行します。

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



**図18：「.bin」を「.hex」に変換する**

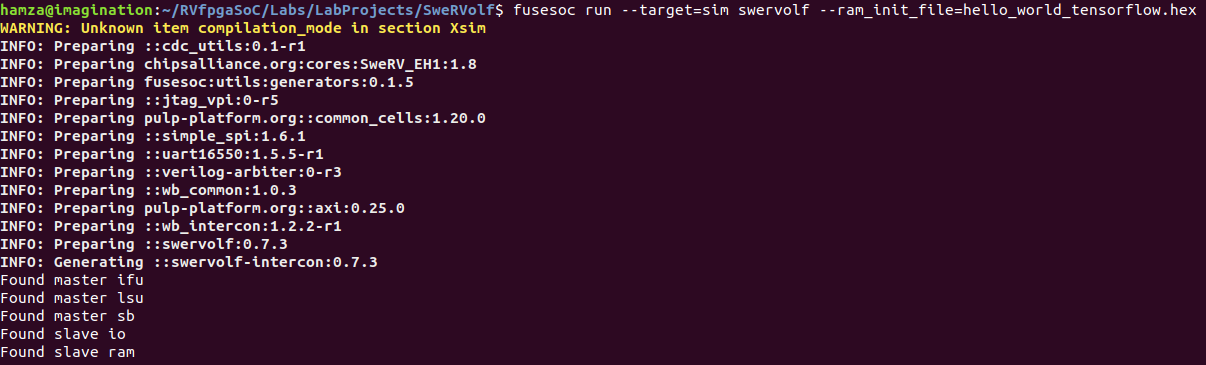
**ステップ3：**「WORKSPACE」ディレクトリに移動して戻ります。



**図19：venv-zephyrにパッケージをインストールする**

**ステップ4：**「**.hex**」ファイルをシミュレータでロードします。

* fusesoc run --target=sim swervolf --ram\_init\_file=hello\_world\_tensorflow.hex



**図20：「.hex」ファイルをシミュレータでロードする**

hello\_world例の出力が表示されます（図21を参照）。「正弦」関数のX値およびY値が、プログラムによって印刷されます。



**図21：「hello\_world」の出力**

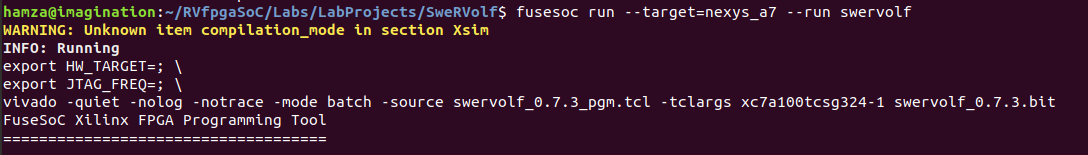
「ctrl + c」を押して、プログラムを終了します。

# Nexys A7ボードでのHello World例の実行

このセクションでは、OpenOCDを使用して、ボードで、「hello\_world」プロジェクトを実行します。

**ステップ1**：Nexys A7ボードをコンピュータに接続し、WorkspaceディレクトリのFPGAビルドコマンドを実行します。

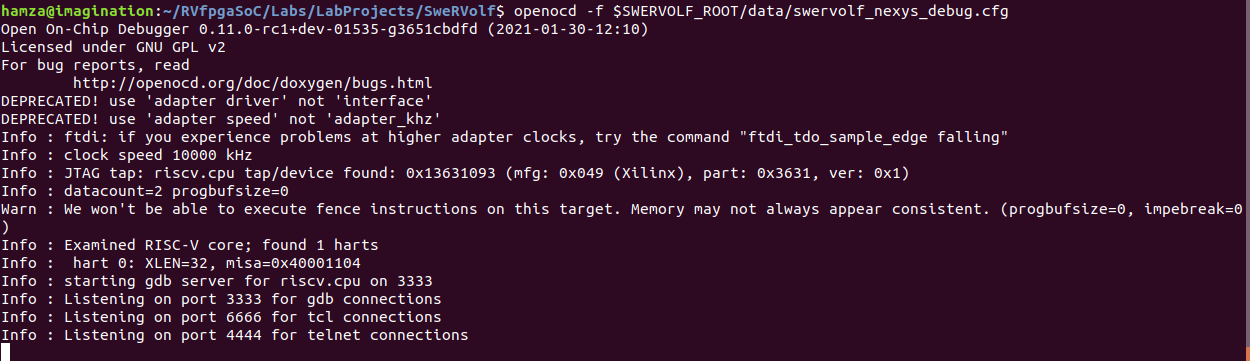
* fusesoc run --target=nexys\_a7 --run swervolf



**図22：FPGA構築の実行**

**ステップ2：**OpenOCDをSweRVolfに接続します。

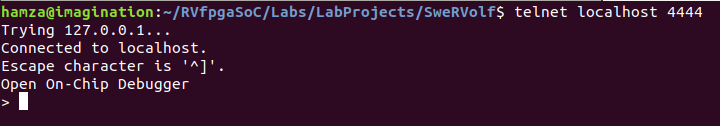
* openocd -f $SWERVOLF\_ROOT/data/swervolf\_nexys\_debug.cfg



**図23：接続されているOpenOCD**

**ステップ3**：「Ctrl + Shift + t」を使用して新しいターミナルを開き、次記のコマンドを使用してOpenOCDを介してデバッグセッションに接続します。

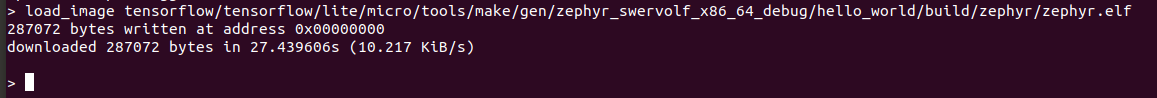
* telnet localhost 4444



**図24：telnet localhost 4444**

OpenOCDにより、*load\_image /path/to/file.elf*を実行することによって、ELFプログラムファイルのロードがサポートされます。パスは、OpenOCDを起動したディレクトリに関連していることを、忘れないでください。

* load\_image tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr\_swervolf\_x86\_64\_debug/hello\_world/build/zephyr/zephyr.elf



**図25：「.elf」ファイルをロード**

プログラムをロードしたら、次記のコマンドを使用して、プログラムカウンタをアドレスゼロに設定します。

* reg pc 0



**図26：プログラムカウンタをゼロに設定**

次記のコマンドを使用して、プログラムを開始します。

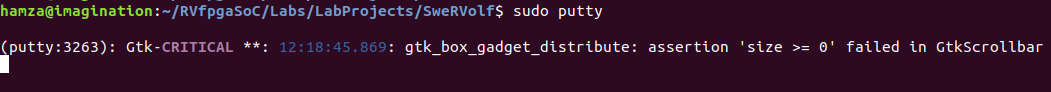
* resume



**図27：プログラムを開始する**

**ステップ4**：「Ctrl + Shift + t」を使用して新しいターミナルを開きます。次記のコマンドを使用して「PuTTY」を開きます。

* sudo putty

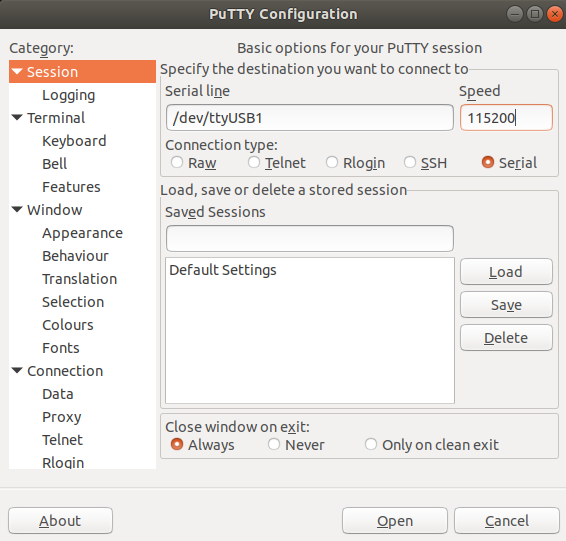


**図28：PuTTYを開く**

ここでPuTTYを、使用するNexys A7ボードのシリアルコンソールとして使用します。

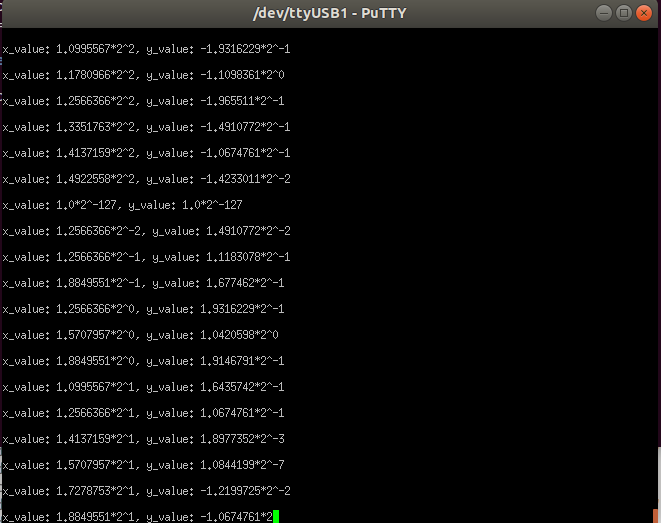
**ステップ5**：以下の構成を設定します。

接続種類に「**Serial**」を選択し、次にシリアルラインとして「**/dev/ttyUSB1**」を入力し、速度を「**115200**」に設定します。「Open」をクリックしてシリアルコンソールを起動します。



**図29：PuTTYの構成**

シリアルコンソールに、hello\_world例の出力が表示されます（図30を参照）。



**図30：シリアルコンソール**

シミュレーションセクションで見たように再び、TensorFlowモデルでプロットされる正弦関数の「X」座標および「Y」座標が、プログラムによって印刷されます。

|  |
| --- |
| **注意**：シリアルコンソールを開けない場合は、シリアルラインとして「/dev/ttyUSB0」を試行します。 |

このラボで、TensorFlowの「hello\_world」例をZephyrアプリケーションとして正常に構築し、その例をSweRVolfで実行しました。