



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ツールチェーン（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（AXIインターコネク、Wishboneインターコネク、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。

1. はじめに

このラボでは、Zephyr（リア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。

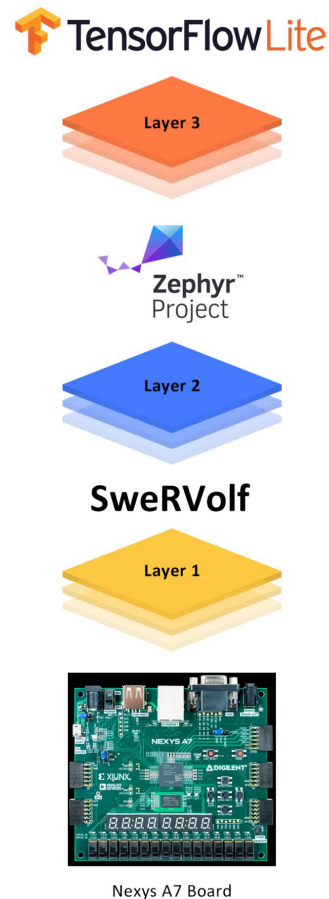


図1：FPGAボード上の層

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。

2. 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。

3. 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、マイクロコントローラ用TensorFlow Liteの初歩中の初歩の使用を実際示すために、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](#)のTensorFlowの公式ドキュメントを参照してください。

4. 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。

```
~/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=$WORKSPACE/zephyr
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

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`

```
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.
```

図3：Tensorflow

「tensorflow」ディレクトリに移動します。

➤ `cd tensorflow`

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

```
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$
```

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/

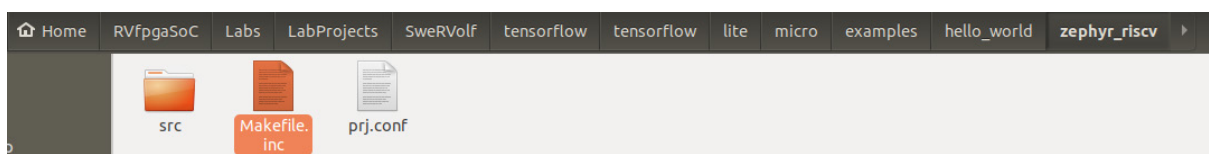


図6：Makefile.inc

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/

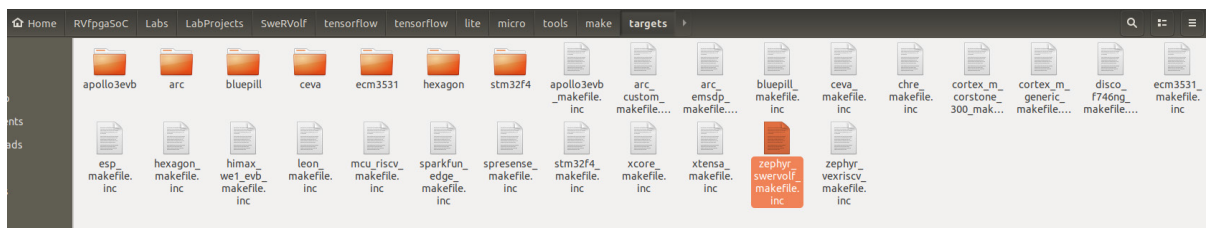


図7 : zephyr_swervolf_makefile.inc

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 ..`

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

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`


```
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).
```

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`

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

```
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$
```

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`

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

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`


```
(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
```

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。

5. 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/
```

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

```
(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 '47268dff45019863e20438ca3746c6c62df6ef09'.
```

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ー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。

```
[ 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/hamza/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/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/
tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build'
make[1]: Leaving directory '/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/
tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world/build'
```

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

6. 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/

```
/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$
```

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
d_tensorflow.hex

```
hamza@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/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/hello_world_tensorflow.hex
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/tensorflow/tensorflow/lite/micro/tools/make/gen/zephyr_swervolf_x86_64_debug/hello_world
$
```

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。

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

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
```

```
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
```

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。

```
g++ jtagServer.o jtag_common.o tb.o verilated.o verilated_dpi.o verilated_vcd.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
```

図21: 「hello_world」の出力

「ctrl + c」を押して、プログラムを終了します。

7. 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
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=nexys_a7 --run swervolf
WARNING: Unknown item compilation_mode in section Xsin
INFO: Running
export HW_TARGET=; \
export JTAG_FREQ=; \
vivado -quiet -nolog -notrace -mode batch -source swervolf_0.7.3_pgm.tcl -tclargs xc7a100tcs324-1 swervolf_0.7.3.bit
Fusesoc Xilinx FPGA Programming Tool
=====
```

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`

```
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
```

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`

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

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`

```
> 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)
> █
```

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`

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

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

```
> 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

```
hamza@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
```

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。

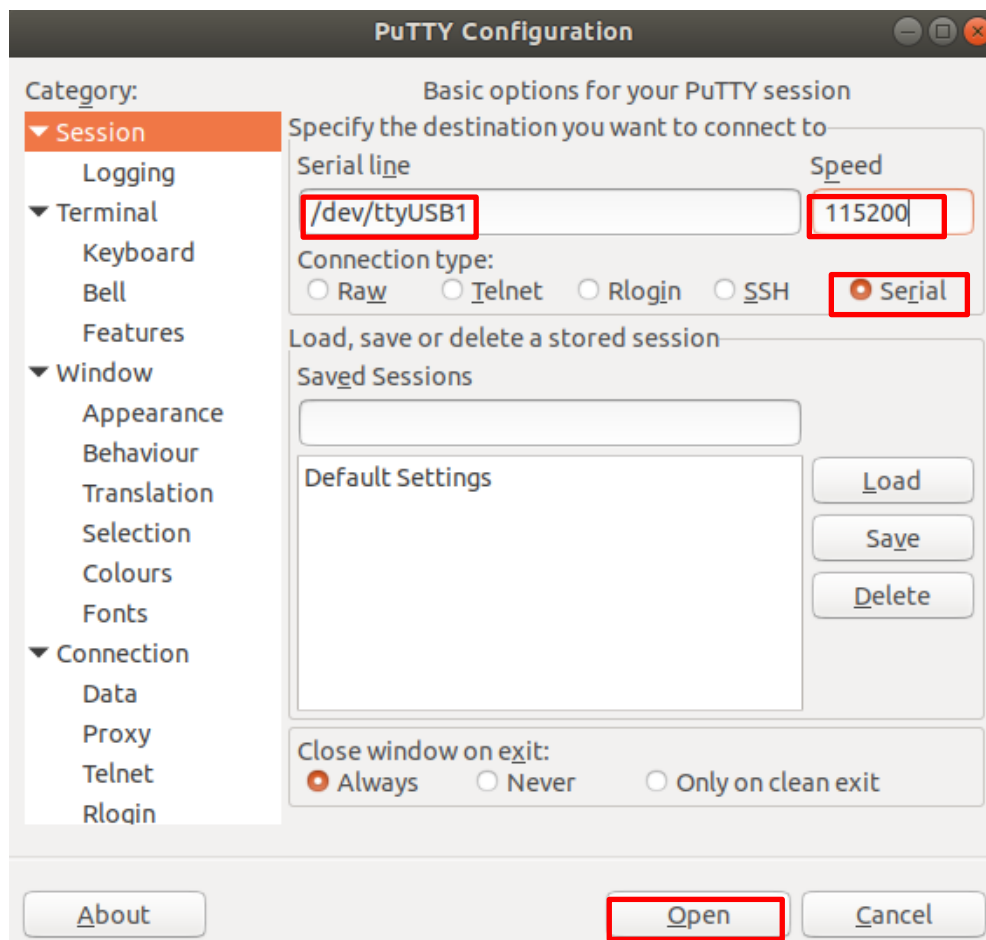
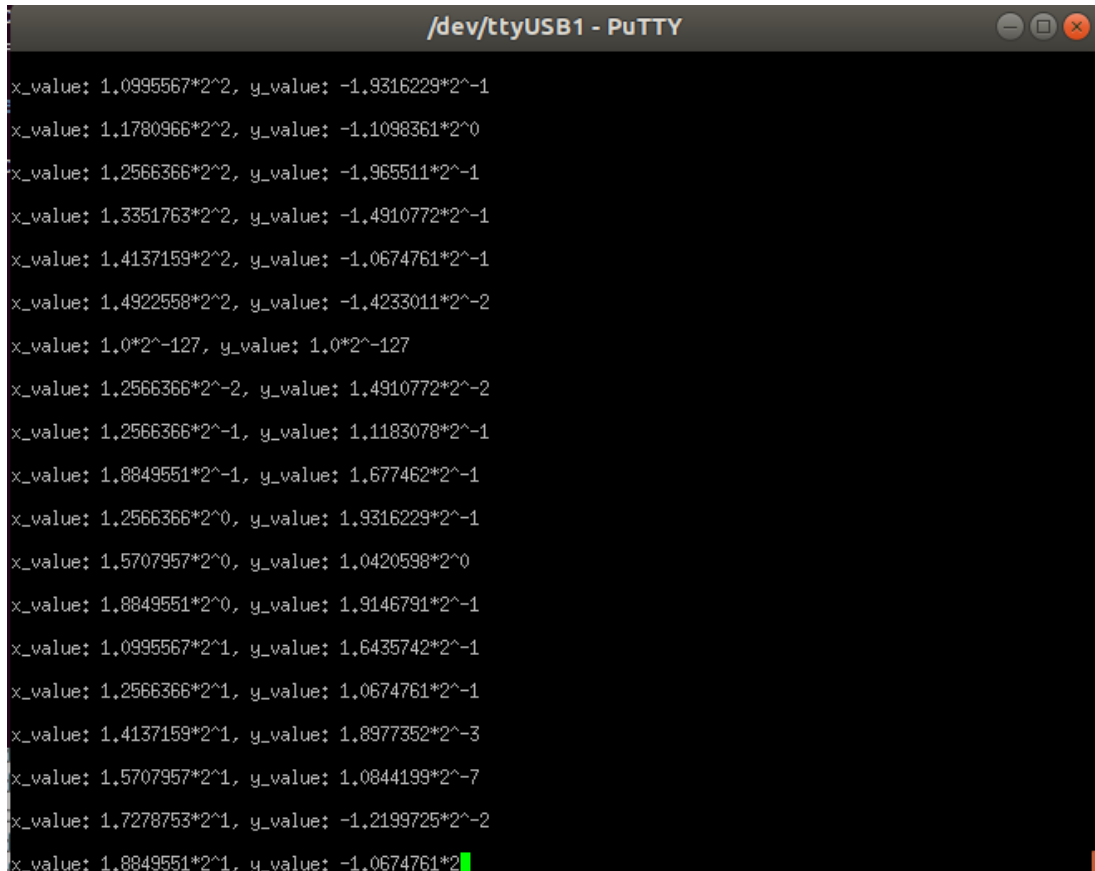


図29 : PuTTYの構成

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）。



```

/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

```

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。