



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga-SoCラボ4

SweRVolfでのZephyr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、SweRVolfでZephyrリアルタイムオペレーティングシステム（RTOS）を実行する方法を示します。リアルタイムオペレーティングシステム（RTOS）は、リアルタイムアプリケーションで役割を果たすオペレーションシステムであり、これによって、入来するデータが、ほとんどの場合バッファ遅延なしに処理されます。ラボ2および3で、RISC-Vアセンブリ言語またはC言語で書かれたサンプルプログラムを実行してきました。実際のアプリケーションでは、ほとんどの場合、SoCによってオペレーティングシステムが実行され、アプリケーションはオペレーティングシステムの上で実行されます。

組み込みシステムのオペレーティングシステムの総合的なカテゴリには、次記の2つがあります：組み込みLinuxベースオペレーティングシステムおよびリアルタイムオペレーティングシステム（RTOS）。SoCが特定のCPUで設計される場合、通常その設計はオペレーティングシステムのどれか一つの種類を使うように調製されます。SweRVolfは、リアルタイムオペレーティングシステムを実行する目的で、構築されました。SweRV EH1 CPUにはメモリ管理ユニットがなく、このため、組み込まれているLinuxの実行に苦闘します。

図1にシステム全体のさまざまなハードウェア/ソフトウェア層の説明図が示されています。

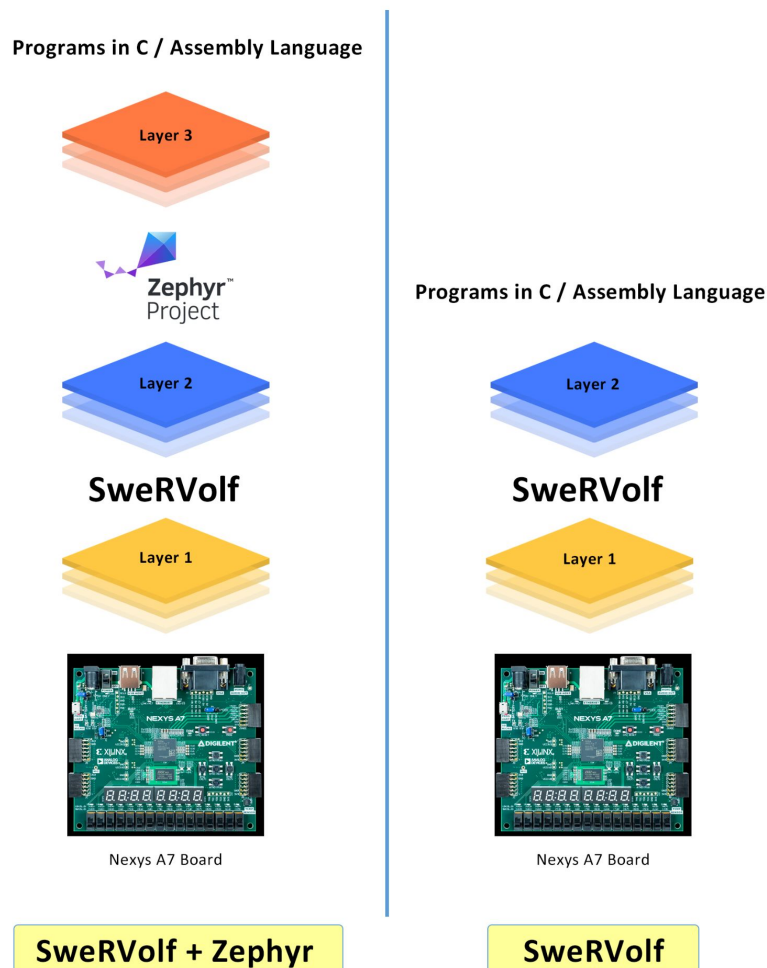


図1：FPGAボード上の層

このラボでは、Zephyr RTOSについて説明し、SweRVolfでZephyr RTOSを構築および実行し、Zephyrアプリケーションを構築および実行します。

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 (11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. Zephyrの概要

Zephyr Projectは、複数のハードウェアアーキテクチャに対応し、リソースの制約を受けるデバイスに対して最適化された、セキュリティに留意して構築された、拡張可能なリアルタイムオペレーティングシステムです。Zephyr OSは、リソースによる制約を受けるシステム（簡単な組み込み環境センサおよびLEDウェアラブルから最新式のスマートウォッチおよびIoT無線ゲートウェイまで）で使用するために設計された、設置面積が小さなカーネルに基づいています。

Zephyrにより、以下のような開発用のよく知られている多くのサービスが、提供されます。マルチスレッディング、メモリ割り当て、スレッド間同期、スレッド間データ受け渡し、パワーマネージメント。

Zephyrは、さまざまなCPUアーキテクチャおよび開発者ツールが含まれている多種多様なボードに、対応します。サポートされるSoC、プラットフォーム、ドライバの数の増加への対応が、追加されてきました。

Zephyrカーネルは以下のような複数のアーキテクチャに対応します。

- RISC-V (32および64ビット)

Zephyrプロジェクトの詳細については、Zephyrプロジェクトドキュメントを<http://docs.zephyrproject.org>で参照してください。

このラボでは、まずZephyrのバージョン2.4をワークスペースに追加する方法を、示します。次に、Zephyrに付属している2~3のサンプル例用のコードを、構築します。このラボでは、Zephyrをハードウェアおよびシミュレーションの両方で使用する例を、示します。

4. ハードウェア/ソフトウェア層を理解する

ラボ2および3では、FPGAボードでプログラムを実行するプロセスは、以下のステップに従いました。

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 : SweRVolfでプログラムを構築および実行する

2番目のステップで、RISC-Vプログラムを構築し、それをSweRVolfにダウンロードします。

このラボでは、これらのステップを修正して、他の層Zephyr RTOS（リアルタイムオペレーティングシステム）をSweRVolfに追加して、Zephyrでプログラムを実行します。これを実行するステップは以下のとおりです。

ステップ1 : SweRVolfをFPGAボードにダウンロードする

上記と同じ。

ステップ2 : Zephyrを構築する

このステップで、Zephyr用アプリケーションを構築します。アプリケーションを構築するプロセスで、内在するZephyr RTOSも構築されます。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。

プログラムを実行する両モードを横に並べた図が、上の図1に示されています。

ここでは、Zephyrアプリケーションを構築する方法を示し、これらのアプリケーションをZephyrで実行します。

5. SweRVolfでZephyrサポートを追加する

ラボのこのセクションでは、ZephyrをWORKSPACEに追加する方法を示します。

Ubuntuターミナルを開き、以下のコマンドを実行します。

ステップ1 : 以前のラボでワークスペースを作成した場所であるディレクトリ「**SweRVolf**」に移動して、プロジェクトのルートとして使用します。これを**\$WORKSPACE**と呼ぶことにします。ここで、同じシェル変数をもう一度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`

「`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=$WORKSPACE/fusesoc_libraries/swervolf
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図2：シェル変数を設定

ステップ2： ZephyrおよびSweRVolf固有のドライバを追加します。

次記を実行することによって、FuseSoCワークスペースと同じディレクトリに、West（Zephyrの構築ツール）ワークスペースを作成します。

➤ west init

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ west init
=== Initializing in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf
--- Cloning manifest repository from https://github.com/zephyrproject-rtos/zephyr, rev. master
Initialized empty Git repository in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/.west/manifest-tmp/.git/
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 551804 (delta 2), reused 1 (delta 1), pack-reused 551800
Receiving objects: 100% (551804/551804), 375.87 MiB | 435.00 KiB/s, done.
Resolving deltas: 100% (418833/418833), done.
From https://github.com/zephyrproject-rtos/zephyr
* branch                master                                -> FETCH_HEAD
* [new branch]          backport-23821-to-v1.14-branch        -> origin/backport-23821-to-v1.14-branch
* [new branch]          backport-24971-to-v1.14-branch        -> origin/backport-24971-to-v1.14-branch
* [new branch]          backport-25852-to-v1.14-branch        -> origin/backport-25852-to-v1.14-branch
* [new branch]          backport-26571-to-v1.14-branch        -> origin/backport-26571-to-v1.14-branch
* [new branch]          backport-29181-to-v2.4-branch        -> origin/backport-29181-to-v2.4-branch
* [new branch]          backport-31759-to-v2.5-branch        -> origin/backport-31759-to-v2.5-branch
* [new branch]          backport-31908-to-v2.4-branch        -> origin/backport-31908-to-v2.4-branch
.....
* [new tag]             zephyr-v2.2.0                    -> zephyr-v2.2.0
* [new tag]             zephyr-v2.2.1                    -> zephyr-v2.2.1
* [new tag]             zephyr-v2.3.0                    -> zephyr-v2.3.0
* [new tag]             zephyr-v2.4.0                    -> zephyr-v2.4.0
* [new tag]             zephyr-v2.5.0                    -> zephyr-v2.5.0
b0b20112e80187705a08240919613ca9937baae6 refs/remotes/origin/master
Branch 'master' set up to track remote branch 'master' from 'origin'.
Already on 'master'
--- setting manifest.path to zephyr
=== Initialized. Now run "west update" inside /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図3：初期化されたwest

ステップ3： 次記のコマンドを使用して、SweRVolf固有のドライバおよびボードサポートパッケージ（BSP）を追加します。

➤ west config manifest.path fusesoc_libraries/swervolf

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ west config manifest.path fusesoc_libraries/swervolf
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図4：west config

➤ west update

インターネットのダウンロード速度によって異なりますが、ダウンロードプロセスを完了するまでに数分かかることがあります。


```

hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ west update
=== updating zephyr (zephyr):
HEAD is now at 7a3b253ced release: Zephyr 2.4.0
WARNING: left behind zephyr branch "master"; to switch back to it (fast forward):
  git -C zephyr checkout master
=== updating cmsis (modules/hal/cmsis):
--- cmsis: initializing
Initialized empty Git repository in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/modules/hal/cmsis/.git/
--- cmsis: fetching, need revision 542b2296e6d515b265e25c6b7208e8fea3014f90
remote: Enumerating objects: 563, done.
remote: Counting objects: 100% (563/563), done.
remote: Compressing objects: 100% (291/291), done.
remote: Total 563 (delta 288), reused 528 (delta 267), pack-reused 0
Receiving objects: 100% (563/563), 2.21 MiB | 618.00 KiB/s, done.
Resolving deltas: 100% (288/288), done.
From https://github.com/zephyrproject-rtos/cmsis
* [new branch]      master      -> refs/west/master
HEAD is now at 542b229 DSP: Integrate CMSIS-DSP 1.8.0 (CMSIS 5.7.0)
HEAD is now at 542b229 DSP: Integrate CMSIS-DSP 1.8.0 (CMSIS 5.7.0)
=== updating hal_atmel (modules/hal/atmel):

=====
=== updating trusted-firmware-m (modules/tee/tfm):
--- trusted-firmware-m: initializing
Initialized empty Git repository in /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/modules/tee/tfm/.git/
--- trusted-firmware-m: fetching, need revision 143df675557305b61f7930a50459a53a8d2bb097
remote: Enumerating objects: 1970, done.
remote: Counting objects: 100% (1970/1970), done.
remote: Compressing objects: 100% (1249/1249), done.
remote: Total 16030 (delta 633), reused 1498 (delta 536), pack-reused 14060
Receiving objects: 100% (16030/16030), 36.00 MiB | 872.00 KiB/s, done.
Resolving deltas: 100% (7538/7538), done.
From https://github.com/zephyrproject-rtos/trusted-firmware-m
* [new branch]      master      -> refs/west/master
HEAD is now at 143df67 CMakeLists.txt: make BL2 configurable
HEAD is now at 143df67 CMakeLists.txt: make BL2 configurable
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$

```

図5 : westの更新

ワークスペースは以下のように表示されます。

```

$WORKSPACE
├── fusesoc_libraries
│   ├── ...
│   └── swervolf
├── ...
└── zephyr

```

6. ZephyrアプリケーションのVerilatorでの構築および実行

このセクションでは、Zephyrで実行できるプログラムを構築する方法を説明します。次に、そのようなプログラムをVerilatorシミュレータでシミュレーションする方法を示します。このセクションでは、プログラム例を示します。

1. Zephyr Hello World例

この例では、「Hello World」 + 「Configured Board Name」がターミナルに印刷されます。ソースコードについては、図6を参照してください。

```

1
2 #include <zephyr.h>
3 #include <sys/printk.h>
4
5 void main(void)
6 {
7     printk("Hello World! %s\n", CONFIG_BOARD);
8 }
9

```

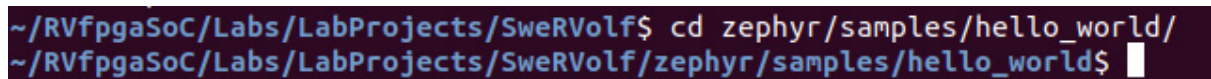
図6 : hello_world例のmain.c

ステップ1: 以下のパスにあるこの例のディレクトリに進みます。

```
$WORKSPACE/zephyr/samples/hello_world
```

これを実行するには、以下のコマンドを実行します。

```
> cd zephyr/samples/hello_world
```

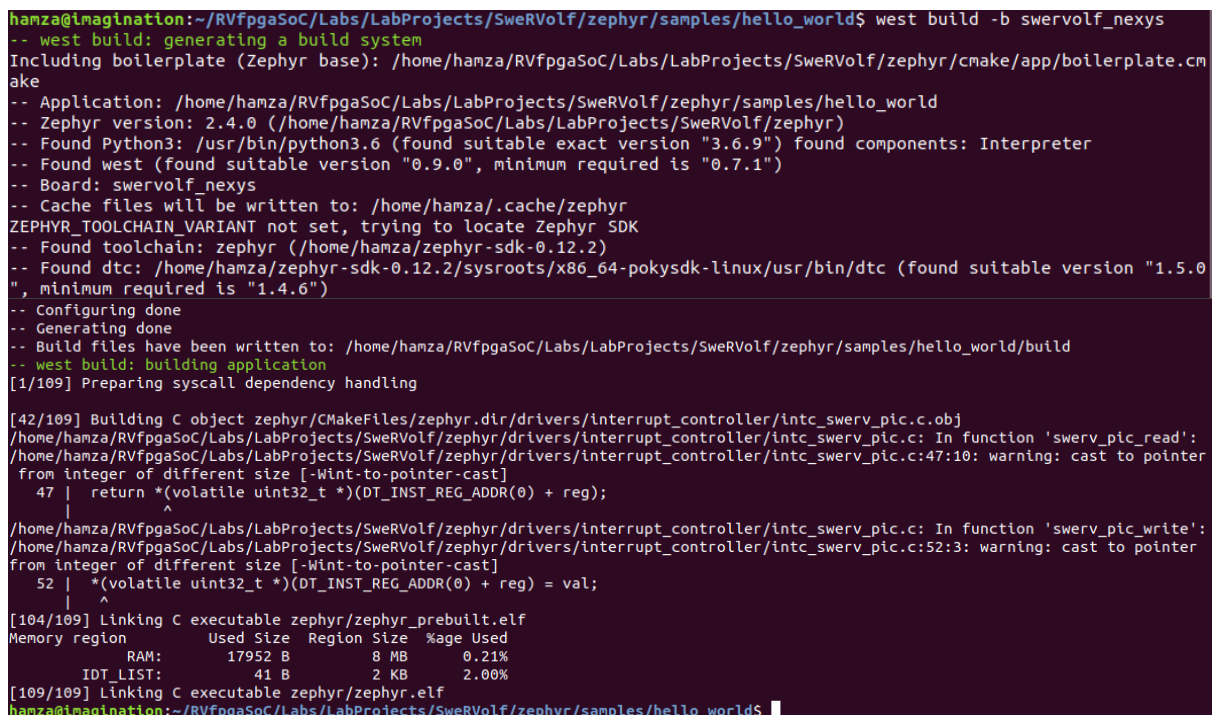


```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd zephyr/samples/hello_world/  
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$
```

図7: hello_worldディレクトリに移動する

ステップ2: 次記のコマンドを使用して、「hello_world」例用のコードを構築します。

```
> west build -b swervolf_nexys
```



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$ west build -b swervolf_nexys  
-- west build: generating a build system  
Including boilerplate (Zephyr base): /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/cmake/app/boilerplate.cmake  
-- Application: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world  
-- Zephyr version: 2.4.0 (/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr)  
-- Found Python3: /usr/bin/python3.6 (found suitable exact version "3.6.9") found components: Interpreter  
-- Found west (found suitable version "0.9.0", minimum required is "0.7.1")  
-- Board: swervolf_nexys  
-- Cache files will be written to: /home/hamza/.cache/zephyr  
ZEPHYR_TOOLCHAIN_VARIANT not set, trying to locate Zephyr SDK  
-- Found toolchain: zephyr (/home/hamza/zephyr-sdk-0.12.2)  
-- Found dtc: /home/hamza/zephyr-sdk-0.12.2/sysroots/x86_64-pokysdk-linux/usr/bin/dtc (found suitable version "1.5.0", minimum required is "1.4.6")  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world/build  
[1/109] Preparing syscall dependency handling  
[42/109] Building C object zephyr/CMakeFiles/zephyr.dir/drivers/interrupt_controller/intc_swerv_pic.c.obj  
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_read':  
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:47:10: warning: cast to pointer  
from integer of different size [-Wint-to-pointer-cast]  
47 | return *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg);  
    | ^  
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_write':  
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:52:3: warning: cast to pointer  
from integer of different size [-Wint-to-pointer-cast]  
52 | *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg) = val;  
    | ^  
[104/109] Linking C executable zephyr/zephyr_prebuilt.elf  
Memory region      Used Size  Region Size  %age Used  
RAM:                17952 B      8 MB         0.21%  
IDT_LIST:           41 B        2 KB         2.00%  
[109/109] Linking C executable zephyr/zephyr.elf  
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$
```

図8: hello_worldの構築

これにより、**hello_world**例用のzephyr.elfファイルおよびzephyr.binファイルが作成されます。「.bin」ファイルをシミュレータで使用しますが、適切なVerilog hexファイルに変換する必要があります。

ステップ3: 「.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/zephyr/samples/hello_world/App.hex`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$ python3 $SWERVOLF_ROOT/sw/makehex.py
build/zephyr/zephyr.bin > /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world/App.hex
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$
```

図9：作成されたhello_world hexファイル

ステップ4：WORKSPACEディレクトリに移動します：

- `cd $WORKSPACE`

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/hello_world$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図10：メインWORKSPACEディレクトリに移動する

ステップ5：「.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=zephyr/samples/hello_world/App.hex`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
--ram_init_file=zephyr/samples/hello_world/App.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
=====
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
```

図11：fusesocの実行

ターミナルにより、以下の出力が表示されます（図12を参照）。

```
make[1]: Leaving directory '/home/hamza/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Loading RAM contents from /home/hamza/SweRVolf/zephyr/samples/hello_world/App.hex
Releasing reset
*** Booting Zephyr OS build zephyr-v2.4.0 ***
Hello World! swervolf_nexys
```

図12 : hello_world例の出力

「**ctrl + c**」を押して、プログラムを停止します。

2. Zephyr Philosophers例

ソリューションのDining Philosophers Problem（古典的なマルチスレッド同期の問題）での実施。この特定の実施により、異なる優先順位の複数の先行実施可能で協力的なスレッドの使用、およびダイナミックミューテックスとスレッドのスリープの発生が、実証されます。

哲学者は常に、最も低いフォークを最初に手にしようとします（f1の次にf2）。これが済むと、フォークを逆の順に戻します（f2の次にf1）。2本のフォークがある場合、彼は食事をします。そうでない場合、彼は考えます。過渡的な状態も同じです。哲学者が空腹のとき、飢えていますが、フォークは使用できません。哲学者が2本目のフォークを使えるのを待っているとき、彼は1本のフォークを手にはしています。

各哲学者は食事中状態と思考中状態をランダムに繰り返します。

次記のパスに移動して、この例のソースコードを表示します。

```
$WORKSPACE/zephyr/samples/philosophers/src/main.c
```

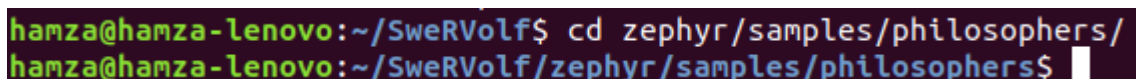
この例では、同じプロセスをもう一度繰り返しますが、哲学者のディレクトリで行います。

ステップ1：このプログラム例は次記のディレクトリにあります。

```
$WORKSPACE/zephyr/samples/philosophers
```

次記のコマンドを使用して、そのディレクトリに変更します。

```
> cd zephyr/samples/philosophers
```



```
hamza@hamza-lenovo:~/SweRVolf$ cd zephyr/samples/philosophers/
hamza@hamza-lenovo:~/SweRVolf/zephyr/samples/philosophers$
```

図13 : 哲学者のディレクトリに移動する

ステップ2：次記のコマンドを使用して、哲学者例用のコードを構築します。

```
> west build -b swervolf_nexys
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$ west build -b swervolf_nexys
-- west build: generating a build system
Including boilerplate (Zephyr base): /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/cmake/app/boilerplate.cmake
-- Application: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers
-- Zephyr version: 2.4.0 (/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr)
-- Found Python3: /usr/bin/python3.6 (found suitable exact version "3.6.9") found components: Interpreter
-- Found west (found suitable version "0.9.0", minimum required is "0.7.1")
-- Board: swervolf_nexys
-- Cache files will be written to: /home/hamza/.cache/zephyr
ZEPHYR_TOOLCHAIN_VARIANT not set, trying to locate Zephyr SDK
-- Found toolchain: zephyr (/home/hamza/zephyr-sdk-0.12.2)
```

```
-- Configuring done
-- Generating done
-- Build files have been written to: /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers/build
-- west build: building application
[1/110] Preparing syscall dependency handling

[44/110] Building C object zephyr/CMakeFiles/interrupt_controller/intc_swerv_pic.c.obj
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_read':
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:47:10: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   47 |     return *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg);
      |           ^
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c: In function 'swerv_pic_write':
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/drivers/interrupt_controller/intc_swerv_pic.c:52:3: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   52 |     *(volatile uint32_t *) (DT_INST_REG_ADDR(0) + reg) = val;
      |     ^
[105/110] Linking C executable zephyr/zephyr_prebuilt.elf
Memory region      Used Size  Region Size  %age Used
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$
```

図14：哲学者の構築

これにより、哲学者例用のzephyr.elfファイルおよびzephyr.binファイルが作成されます。再び、「.bin」ファイルを適切なVerilog hexファイルに変換します。

ステップ3：「.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、哲学者の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/{YourUsername}/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers/App.hex

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$ python3 $SWERVOLF_ROOT/sw/makehex.py build/zephyr/zephyr.bin > /home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers/App.hex
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$
```

図15：哲学者のhexファイルを作成する

ステップ4：WORKSPACEディレクトリに移動します：

- cd \$WORKSPACE

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/philosophers$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図16：メインディレクトリ

ステップ5 : .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=zephyr/samples/philosophers/App.hex`

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
--ram_init_file=zephyr/samples/philosophers/App.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
=====
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
```

図17 : fusesocの実行

以下の出力が表示されます。

```
make[1]: Leaving directory '/home/hamza/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Loading RAM contents from /home/hamza/SweRVolf/zephyr/samples/philosophers/App.hex
Releasing reset
*** Booting Zephyr OS build zephyr-v2.4.0 ***

Philosopher 0 [P: 3]    HOLDING ONE FORK
Philosopher 1 [P: 2]    EATING  [ 125 ms ]
Philosopher 2 [P: 1]    THINKING [ 175 ms ]
Philosopher 3 [P: 0]    EATING  [ 325 ms ]
Philosopher 4 [C:-1]    THINKING [ 400 ms ]
Philosopher 5 [C:-2]    STARVING

Demo Description
-----
An implementation of a solution to the Dining Philosophers
problem (a classic multi-thread synchronization problem).
This particular implementation demonstrates the usage of multiple
preemptible and cooperative threads of differing priorities, as
well as dynamic mutexes and thread sleeping.
```

図18 : Zephyr哲学者の出力

7. ハードウェア用Zephyrアプリケーションの構築

ここで、ハードウェアでZephyrを実行するSweRVolf用プログラムの構築方法を、示します。

1. Zephyr Blinky例

Blinkyは、次記を使用してLEDが点滅し続けるようにする、簡単なアプリケーションです：`GPIO API <gpio_api>`。ソースコードにより、GPIOピンを出力として設定し、これをオンおよびオフする方法が、示されます。

```

1
2  /*
3   * Copyright (c) 2016 Intel Corporation
4   *
5   * SPDX-License-Identifier: Apache-2.0
6   */
7
8  #include <zephyr.h>
9  #include <device.h>
10 #include <devicetree.h>
11 #include <drivers/gpio.h>
12
13 /* 1000 msec = 1 sec */
14 #define SLEEP_TIME_MS 1000
15
16 /* The devicetree node identifier for the "led0" alias.*/
17 #define LED0_NODE DT_ALIAS(led0)
18
19 #if DT_NODE_HAS_STATUS(LED0_NODE, okay)
20 #define LED0 DT_GPIO_LABEL(LED0_NODE, gpios)
21 #define PIN DT_GPIO_PIN(LED0_NODE, gpios)
22 #define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
23 #else
24 /* A build error here means your board isn't set up to blink an LED.*/
25 #error "Unsupported board: led0 devicetree alias is not defined"
26 #define LED0 ""
27 #define PIN 0
28 #define FLAGS 0
29 #endif
30
31 void main(void)
32 {
33     const struct device *dev;
34     bool led_is_on = true;
35     int ret;
36
37     dev = device_get_binding(LED0);
38     if (dev == NULL) {
39         return;
40     }
41
42     ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
43     if (ret < 0) {
44         return;
45     }
46
47     while (1) {
48         gpio_pin_set(dev, PIN, (int)led_is_on);
49         led_is_on = !led_is_on;
50         k_msleep(SLEEP_TIME_MS);
51     }
52 }

```

図19 : blinky例のmain.c

この例のパスは次記のとおりです。

```
$WORKSPACE/zephyr/samples/basic/blink/
```

上記のパスに移動し、ターミナルで次記のコマンドを実行して例を構築し、「.elf」および「.bin」ファイルを生成します。

```
> west build -b swervolf_nexys
```

コードを構築すると、実行可能な.elfファイルがbuild/zephyr/zephyr.elfに、.binファイルが build/zephyr/zephyr.binにあるようになります。

次のセクションで説明されているように、実行可能ファイルをSweRVolfにデバッガでロードでき、バイナリファイルを.hexファイルに変換してシミュレーション用にRAMにロードできます。

8. ハードウェアでZephyrアプリケーションを実行

このアプリケーションをNexys A7ボードで実行するには、OpenOCDを使用してプログラムをロードする必要があります。

ステップ1 : Nexys A7ボードをコンピュータに接続して電源をオンにして、WorkspaceディレクトリのFPGA構築コマンドを実行します。

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

```
***** 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 xc7a100tcs324-1 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 xc7a100tcs324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図20 : FPGA構築の実行

ステップ2 : OpenOCDを使用してボードをプログラムします。

```
> 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
```



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

図21 : OpenOCDの実行

ステップ3 : 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22 : 接続されているOpenOCD

ステップ4 : 「Ctrl + Shift + t」を使用して3番目のターミナルを開き、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23 : telnet

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 zephyr/samples/basic/blink/build/zephyr/zephyr.elf`

```
> load_image zephyr/samples/basic/blink/build/zephyr/zephyr.elf
14848 bytes written at address 0x00000000
downloaded 14848 bytes in 1.420706s (10.206 KiB/s)
>
```

図24 : image .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25：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26：プログラムを開始する

Nexys A7ボードの右端のLEDが点滅を開始します。

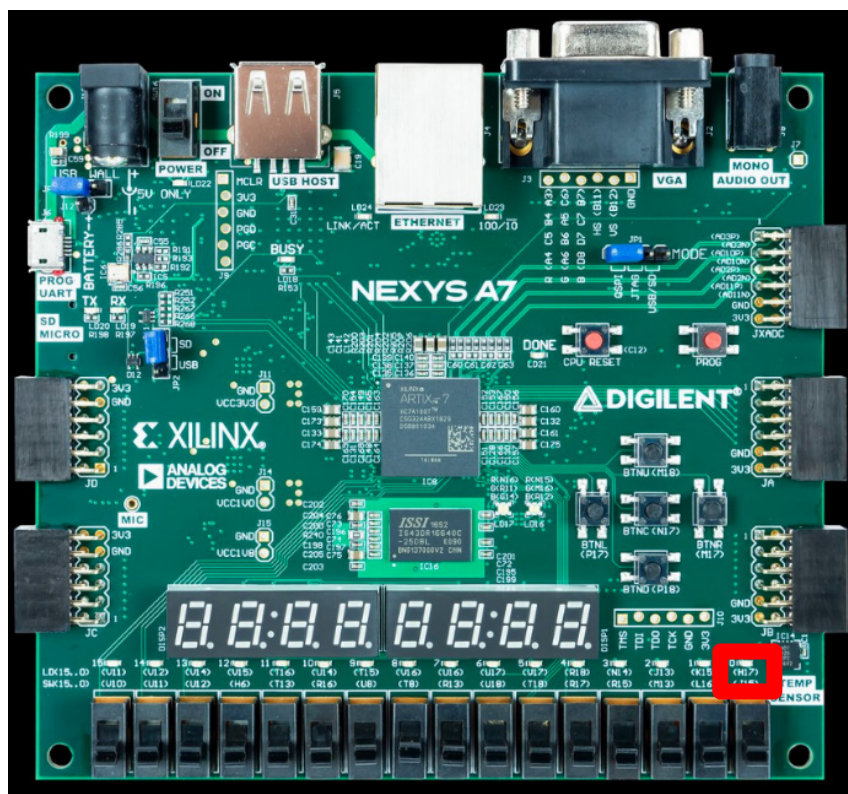


図27：点滅するLED

これで、「ctrl + c」を押して、プログラムを終了できます。

9. Zephyrアプリケーション開発の概要

Zephyrの構築システムは、CMakeに基づいています。構築システムはアプリケーション中心であり、カーネルソースツリーの構築を起動するZephyrベースのアプリケーションが必要です。アプリケーション構築により、構成がコントロールされ、アプリケーションおよびZephyr自体の両方のプロセスが構築され、これらが単一のバイナリにコンパイルされます。

Zephyrのベースディレクトリにより、Zephyrのソースコード、そのカーネル構成オプション、その構築定義が、ホストされます。

アプリケーションディレクトリのファイルにより、Zephyrがアプリケーションとリンクされます。このディレクトリにはすべてのアプリケーション固有のファイル（構成オプション、ソースコードなど）が含まれています。

最も簡単な形式のアプリケーションでは、内容がここにリストされており、下に説明されています。

```
/App
├── CMakeLists.txt
├── prj.conf
└── src
    └── main.c
```

CMakeLists.txt：このファイルにより、他のアプリケーションファイルがある構築システムが示され、アプリケーションディレクトリがZephyrのCMake構築システムとリンクされます。このリンクにより、Zephyrの構築システムによってサポートされる次記の機能が、提供されます：ボード固有のカーネル構成ファイル、実際のまたはエミュレートしたハードウェアのコンパイルされたバイナリを実行またはデバッグする能力など。

カーネル構成ファイル：通常アプリケーションによって、1つ以上のカーネル構成ファイルのアプリケーション固有の値を指定するKconfig構成ファイル（通常prj.confと呼ばれる）が、提供されます。これらのアプリケーション設定は、ボード固有の設定と統合され、カーネル構成が作成されます。

アプリケーションソースコードファイル：アプリケーションによって通常、C言語またはアセンブリ言語で書かれたアプリケーション固有のファイルが1つ以上、提供されます。これらのファイルは、通常srcと呼ばれるサブディレクトリ配置されます。

10. Zephyrアプリケーションの新規作成

以下のステップに従って、アプリケーションディレクトリを新規作成します。

ステップ1：次記のサンプルディレクトリに変更します。

```
> cd zephyr/samples
```

ステップ2: アプリケーションの次記のディレクトリを新規作成します。

```
> mkdir my_first_app
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd zephyr/samples
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples$ mkdir my_first_app
```

図28: プロジェクトディレクトリを作成

ステップ3: すべてのアプリケーションソースコードをsrcと言う名前のサブディレクトリに配置することを、推奨します。これによって、以下のプロジェクトファイルとソースファイルの区別が簡単になります。

```
> cd my_first_app
> mkdir src
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples$ cd my_first_app
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$ mkdir src
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$
```

図29: srcディレクトリをプロジェクトディレクトリの内側にする

ステップ4: srcディレクトリを入力し、アプリケーションのメインソースファイル「main.c」を作成します。

```
> cd src
> nano main.c
```

```
~/SweRVolf/zephyr/samples/my_first_app$ cd src/
~/SweRVolf/zephyr/samples/my_first_app/src$ nano main.c
```

図30: 「main.c」ファイルの作成

Nano Editorがubuntuターミナルで、下図に示されているように開きます。



図31: GNU nano Editor

ステップ5：以下のコードをnano editorでコピーします。このコードは、「hello_world」と「blinky」ソースコード例の混合体です。

```
#include <zephyr.h>
#include <sys/printf.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>

/* 1000 msec = 1 sec */
#define SLEEP_TIME_MS 1000

/* The devicetree node identifier for the "led0" alias.*/
#define LED0_NODE DT_ALIAS(led0)

#if DT_NODE_HAS_STATUS(LED0_NODE, okay)
#define LED0 DT_GPIO_LABEL(LED0_NODE, gpios)
#define PIN DT_GPIO_PIN(LED0_NODE, gpios)
#define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
#else
/* A build error here means your board isn't set up to blink an LED.*/
#error "Unsupported board: led0 devicetree alias is not defined"
#define LED0 ""
#define PIN 0
#define FLAGS 0
#endif

void main(void)
{
    const struct device *dev;
    bool led_is_on = true;
    int ret;

    dev = device_get_binding(LED0);
    if (dev == NULL) {
        return;
    }

    ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
    if (ret < 0) {
        return;
    }

    while (1) {
        gpio_pin_set(dev, PIN, (int)led_is_on);
        led_is_on = !led_is_on;
        k_msleep(SLEEP_TIME_MS);
        printf("This Zephyr Application is Running on %s\n", CONFIG_BOARD);
    }
}
```

図32：「main.c」コード

コードの書き込みが終了したら、「ctrl + x」を押して、終了します。

```
GNU nano 2.9.3

#include <zephyr.h>
#include <sys/printf.h>
#include <device.h>
#include <devicetree.h>
#include <drivers/gpio.h>

/* 1000 msec = 1 sec */
#define SLEEP_TIME_MS 1000

/* The devicetree node identifier for the "led0" alias. */
#define LED0_NODE DT_ALIAS(led0)

#if DT_NODE_HAS_STATUS(LED0_NODE, okay)
#define LED0 DT_GPIO_LABEL(LED0_NODE, gpios)
#define PIN DT_GPIO_PIN(LED0_NODE, gpios)
#define FLAGS DT_GPIO_FLAGS(LED0_NODE, gpios)
#else
/* A build error here means your board isn't set up to blink an LED. */
#error "Unsupported board: led0 devicetree alias is not defined"
#define LED0 ""
#define PIN 0
#define FLAGS 0
#endif

void main(void)
{
    const struct device *dev;
    bool led_is_on = true;
    int ret;

    dev = device_get_binding(LED0);
    if (dev == NULL) {
        return;
    }

    ret = gpio_pin_configure(dev, PIN, GPIO_OUTPUT_ACTIVE | FLAGS);
    if (ret < 0) {
        return;
    }

    while (1) {
        gpio_pin_set(dev, PIN, (int)led_is_on);
        led_is_on = !led_is_on;
        k_msleep(SLEEP_TIME_MS);
        printf("This Zephyr Application is Running on %s\n", CONFIG_BOARD);
    }
}
```

図33 : main.c ファイルコード

そのファイルを保存するかこれによって尋ねられ、Yesの場合は「y」を押す必要があります。

```
Save modified buffer? (Answering "No" will DISCARD changes.)
Y Yes
N No      ^C Cancel
```

図34 : main.c ファイルの保存

「Enter」を押して、そのファイルを名前「main.c」で保存します。

```
File Name to Write: main.c
^G Get Help
^C Cancel
```

図35 : 名前main.cを確認する

ステップ6 : srcディレクトリの外へ移動して、「CMakeLists.txt」ファイルおよび「prj.conf」ファイルを作成する必要があります。

```
> cd ..
> nano CMakeLists.txt
```

```
~/SweRVolf/zephyr/samples/my_first_app/src$ cd ..
~/SweRVolf/zephyr/samples/my_first_app$ nano CMakeLists.txt
```

図36 : CMakeLists.txtを作成する

以下のコードをnano editorにコピーします。

```
cmake_minimum_required(VERSION 3.13.1)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(my_first_app)

target_sources(app PRIVATE src/main.c)
```

図37 : 「CMakeLists.txt」 ファイルコード

「main.c」 ファイルを保存するために実行したのと同じステップを、実行します。




図38 : nano editor

ここで、プロジェクト構成ファイルを作成します。

```
~/SweRVolf/zephyr/samples/my_first_app$ nano prj.conf
~/SweRVolf/zephyr/samples/my_first_app$
```

図39 : プロジェクト構成ファイルを作成

アプリケーション構成オプションは、アプリケーションディレクトリでprj.conf内に設定されます。ソースコードでLEDを使用しているため、「CONFIG_GPIO」パラメータをyesに設定する必要があります。

```
CONFIG_GPIO=y
```

図40 : 「prj.conf」コード

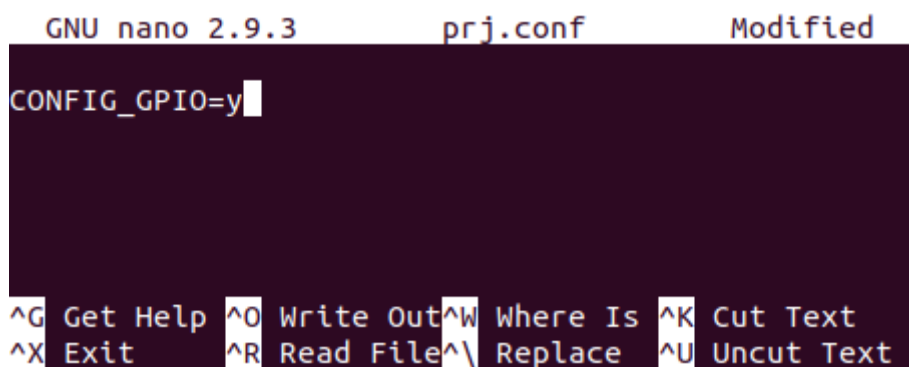


図41 : 「prj.conf」 nano editor

「prj.conf」ファイルを保存します。

ステップ7 : 次記の「my_first_app」用コードを構築します。

```
> west build -b swervolf_nexys
```

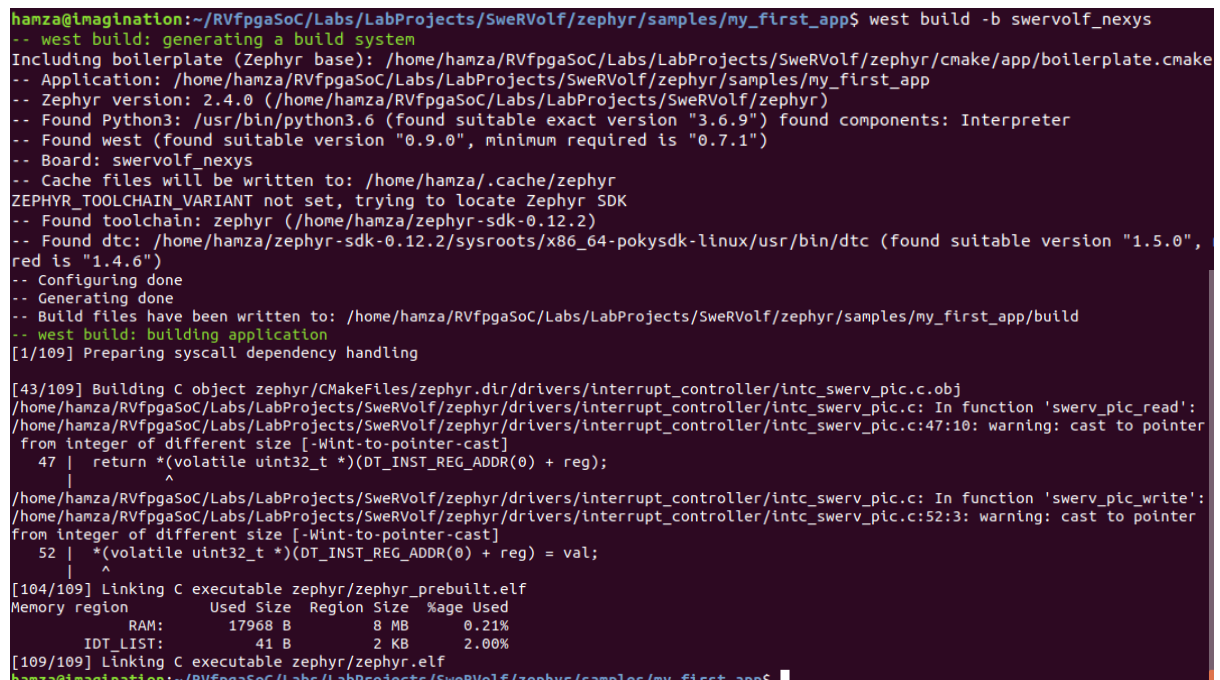


図42 : 「my_first_app」の構築

バイナリは正常に生成されました。「my_first_app」プログラムをNexys A7ボードで実行します。

ステップ9 : WORKSPACEディレクトリに移動します :

```
> cd $WORKSPACE
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/zephyr/samples/my_first_app$ cd $WORKSPACE
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図43 : Workspaceディレクトリに移動する

ステップ10 : 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
```

```
***** 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 xc7a100tcs9324-1 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 xc7a100tcs9324-1 successfully programmed with bitstream swervolf_0.7.3.bit.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図44 : FPGA構築の実行

ステップ11 : OpenOCDを使用してボードをプログラムします。

```
> 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
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ openocd -c "set BITFILE build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit"
-f $SWERVOLF_ROOT/data/swervolf_nexys_program.cfg
Open On-Chip Debugger 0.11.0-rc1+dev-01535-g3651c9df (2021-01-30-12:10)
Licensed under GNU GPL v2
For bug reports, read
    http://openocd.org/doc/doxygen/bugs.html
build/swervolf_0.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
DEPRECATED! use 'adapter driver' not 'interface'
DEPRECATED! use 'adapter speed' not 'adapter_khz'
Info : ftdi: if you experience problems at higher adapter clocks, try the command "ftdi_tdo_sample_edge falling"
Info : clock speed 10000 kHz
Info : JTAG tap: xc7.tap tap/device found: 0x13631093 (mfg: 0x049 (Xilinx), part: 0x3631, ver: 0x1)
Warn : gdb services need one or more targets defined
shutdown command invoked
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図45 : OpenOCDの実行

ステップ12 : 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46 : 接続されているOpenOCD

ステップ13 : 「Ctrl + Shift + t」を使用して3番目のターミナルを開き、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47 : telnet

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 zephyr/samples/my_first_app/build/zephyr/zephyr.elf`

```
> load_image zephyr/samples/my_first_app/build/zephyr/zephyr.elf
14672 bytes written at address 0x00000000
downloaded 14672 bytes in 1.410859s (10.156 KiB/s)
> █
```

図48 : image.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49 : 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50 : プログラムを開始する

ボードのLEDが点滅を開始します。

ステップ14 : 「Ctrl + Shift + t」を使用して新しいターミナルタブを開き、次記のコマンドを使用してPuTTYを開きます。

➤ sudo putty

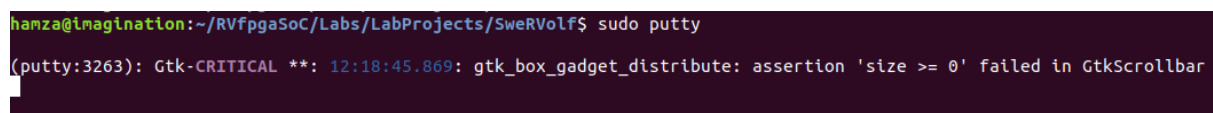


図51 : PuTTYを開く

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15 : 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。

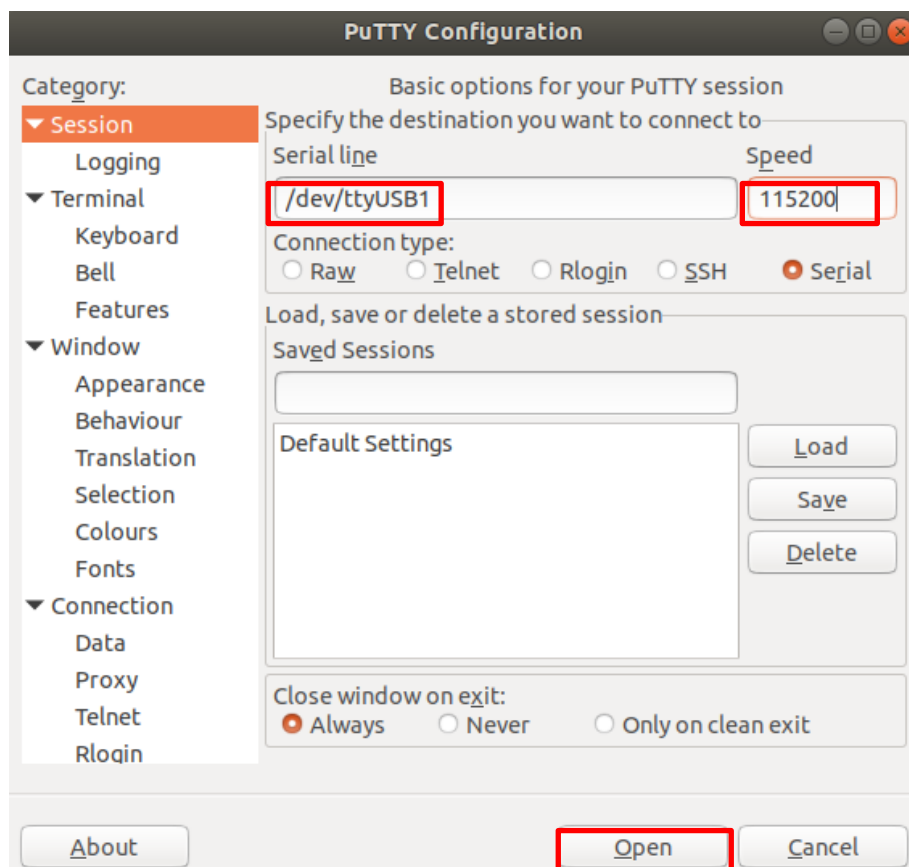


図52 : PuTTYの構成

シリアルコンソールに、プログラム「my_first_app」の出力が表示されます（図53を参照）。

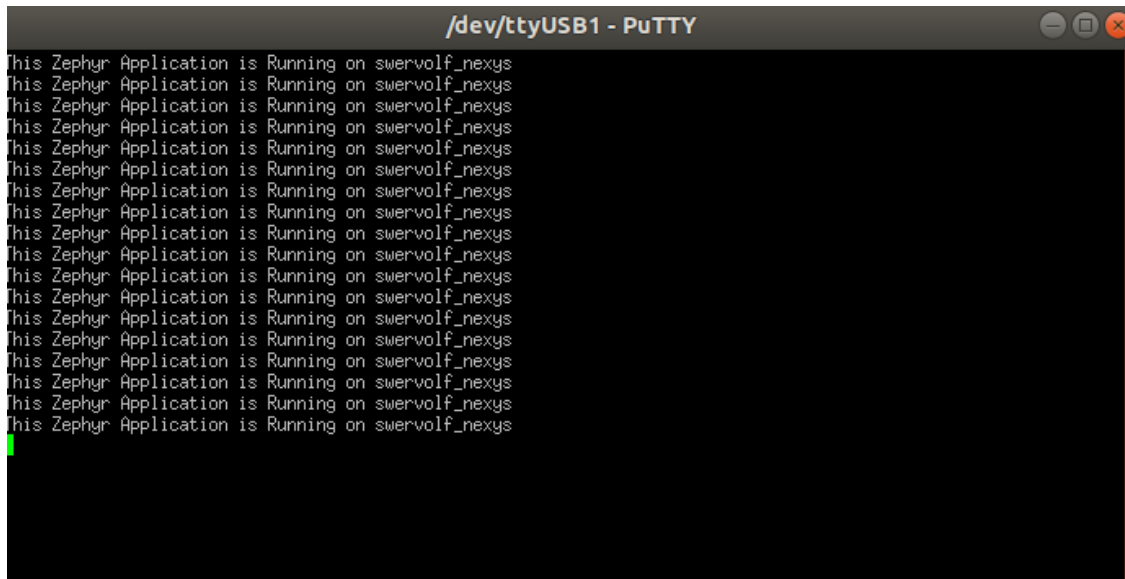


図53：シリアルコンソール出力

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。

シリアルコンソールの出力テキストに表示されているように、このzephyrアプリケーションはSweRVolf Nexysで実行中です。