



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga-SoCラボ3

SweRVolfおよびFuseSoC入門

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Zephy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を詳細に分析します。FuseSoCも紹介し、これを使用してSweRVolf (<https://github.com/chipsalliance/Cores-SweRVolf/>) を構築する方法を示します。

ラボ1で、個々のモジュールがVerilog/SystemVerilogに書き込まれました。次に、Vivadoのブロック設計ツールを使用してSweRVolfXのサブセットを作成するために、これらのモジュールを接続しました。産業界では、設計者は、モジュール接続の視覚法または厳密にVerilog/SystemVerilogコードを使用します。

以下の表に、ブロック設計アプローチの利点と欠点の簡潔な概要が、示されています。

表2：ブロック設計の長所と短所

ブロック設計の利点	ブロック設計の欠点
理解しやすい	チームワークが困難な課題の場合がある
学習曲線	エコシステムが必要（Vivadoブロック設計）
複雑なモジュールを直観的に分かりやすい方法で組み合わせることが可能	アップグレードツールによってシステムが破損されることがある
	FPGAを超えて実際のSoCチップに入り込むために悪戦苦闘することがある
	機械によって生成されたコードは、読み取りにくく、理解しにくいことがある。
	製作するためやより多くの性能を絞り出すためのSoCの微調整を、Verilog/SystemVerilogレベルで実施する必要がある。
	IPコアプロバイダによってVerilogソースコードが提供されるが、これは必ずしも特定のVivadoブロック設計ツールで機能する形ではない。

以下の表に、ハンドチューンアプローチの利点と欠点の簡潔な概要が、示されています。

表3：ハンドチューンコードの長所と短所

ハンドチューンコードの利点	ハンドチューンコードの欠点
FPGAまたはSoCの製作プロセスに非常に特有になるように、ハンドチューニングできる	急峻な学習曲線
バージョンコントロールシステムを使用可能	Verilog/SystemVerilogシステムの知識が必要。

チームおよび共同作業で簡単に機能できる
(さまざまなモジュールでの多様な人が
作業)

すべてのファイル/フォルダ/モジュールがど
のように構造化されているかをよく把握す
るために、しばらく時間がかかる。

SweRVolfのモジュール（最上位モジュール、swervolf_core、SweRV）は、手で書き込まれています。SweRV CPUコア自体はWestern Digitalによって作成されました。インターコネクトモジュールおよびその他のモジュールは、さまざまなベンダからのIPコアであり、これらのベンダはオープンソースライセンスの下でリリースしています。最上位モジュールはOlof Kindgrenによって書き込まれています。

さまざまなモジュールすべてを接続し、Verilogを手で書き込むことは可能です。ただし、時間が経つとこれに飽き飽きすることがあります。特に、大きなチームが協働するとき、コードのある部分が共有されます。

SoC設計プロセス中、いくつかの大きなチームが同時に協働します。さまざまなチームがさまざまな領域に焦点を合わせますが、彼らはさまざまな構成要素やIPブロックを共有します。例えば、シミュレーションチームは同じIPコード（SoC+インターコネクト+CPU+周辺機器）に関係するでしょうが、ラッパでは命令セットシミュレータやVerilatorが対象です。その一方で、FPGAチームは同じIPコード（SoC+インターコネクト+CPU+周辺機器）を使用することを好むでしょうが、さまざまな周辺機器を使用してさまざまな開発ボードでテストします。

構築システムによってさまざまなチームが同じIPコード（SoC+インターコネクト+CPU+周辺機器）で作業することが容易になり、他者の領域を侵害することはありません。構築システムは十分に柔軟でダイナミックで、さまざまなチームおよびその制約/要件に適合する必要があります。SweRVolfは、FuseSoC（読み方は「ヒューズソック」）と呼ばれる構築システムを使用して作成されています。FuseSoC構築システムは、個々の構成要素からのハードウェア設計をつなぎ合わせて大きなものを作ります。

このラボでは、FuseSoC内の構築バージョンのSweRVolfに目を向けます。FuseSoCの「SweRVolf」ライブラリを追加するステップバイステップの手順を示し、それをシミュレーションおよびボード実装のターゲットとします。その後SweRVolfでプログラム例を実行します。

2. 要件

このラボを完了するには、以下のツール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)
- GTKWave (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 (09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)

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、VerilatorおよびXilinxのVivado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. FuseSoCについて

FuseSoCは受賞実績があるパッケージマネージャであり、HDL（ハードウェア記述言語）コード用ビルドツールのセットです。その主要な目的は、IP（知的財産）コアの再使用を増やし、SoCソリューションを作成、構築、シミュレーションすることです。

FuseSoCでの基本的実体はコアです。コアはFuseSoCパッケージマネージャによってローカルまたはリモートの場所に見つけることができ、構築システムによってフルハードウェア設計に組み込まれます。

FuseSoCコアは必ずしもプロセッサではなく、理に適って自己完結型であり、FIFO実装のようにIPの再使用可能部品です。FuseSoCにより、これらもパッケージとして参照されます。他のモジュールではハードウェアのこれらの再使用可能部品は、モジュールと呼ばれます。

FuseSoCの詳細については、次記にあるそのドキュメントを参照してください：

<https://fusesoc.readthedocs.io/en/latest/user/overview.html#understanding-fusesoc>

FuseSoCでは、「SweRVolf」（SweRV RISC-Vコア用FuseSoCベースのSoC）がライブラリでパッケージとして提供されます。FuseSoCを使用して「**swervolf**」ライブラリを追加することができ、「sim」（シミュレーション）や「Nexys_a7」（ボード）など特定のターゲット用にこれを構築できます。

4. SweRVolf : SweRV Eh1用FuseSoCベースのSoC

SweRVolfはSweRV RISC-Vコア用FuseSoCベースのSoCです。SweRVolfにより、RISC-V 適合テスト、Zephyr OS、またはシミュレータやFPGAボードのその他のソフトウェアを実行できます。FuseSoCは可搬性、拡張性、使いやすさに対応および強化することを目的としており、SweRVユーザーはこれを使用して、ソフトウェアを迅速に実行し、SoCをそのニーズに合わせて変更、または新しい対象デバイスに移植することができます。

図1にSweRVolfの高レベルブロック図が示されています。

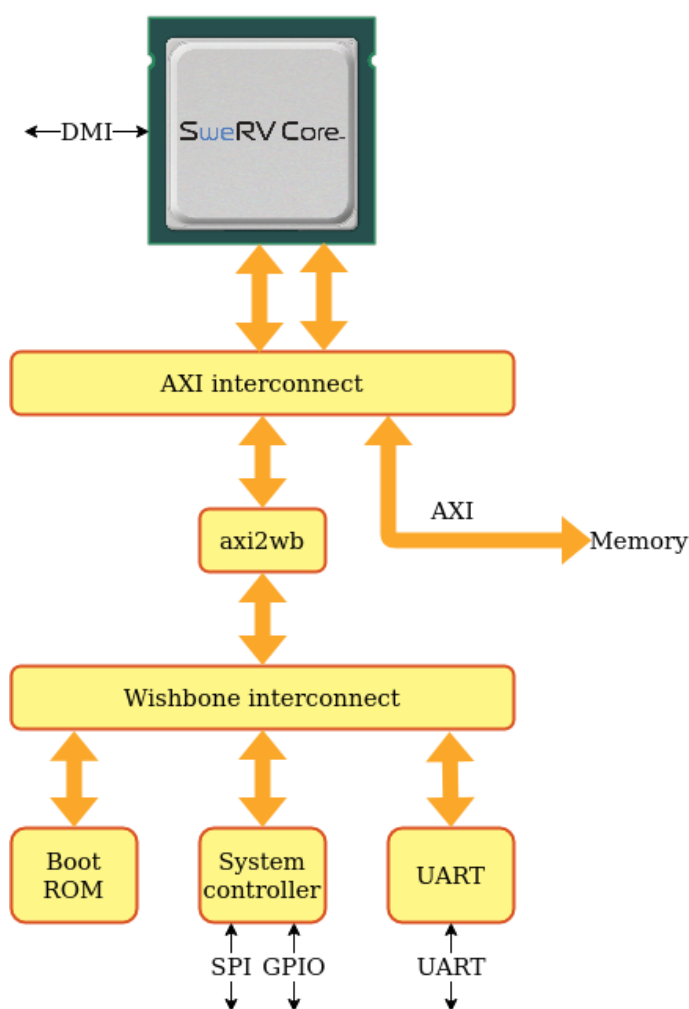


図1 : SweRVolfコア

SweRVolfのコアは、ブートROM付きSweRV CPU、AXI4インターコネクト、UART、SPI、RISC-Vタイマ、GPIOで構成されています。このコアにはRAMは含まれませんが、ターゲット固有のラップによって適切なメモリコントローラに接続されるメモリバスにアクセス可能です。他の外部接続は、クロック、リセット、UART、GPIO、SPI、DMI（デバッグモジュールインターフェース）です。

表4に、SweRV EH1コアにWishboneインターコネクトおよびAXIインターコネクト経由で接続されている周辺機器のメモリにマッピングされているアドレスが、示されています。

表4 : SweRVolfのメモリにマッピングされたアドレス

システム	アドレス
RAM	0x00000000-0x07FFFFFF
ブートROM	0x80000000-0x80000FFF
システムコントローラ	0x80001000-0x80001FFF

UART	0x80002000-0x80002FFF
GPIO	0x80001010-0x80001013

ラボ2のSystemVerilogモジュールRVfpgaSimおよびRVfpgaNexysと同様に、SweRVolfによって等価な以下の2つのバージョンのSweRVolfも提供されます。シミュレーション用SweRVolf Sim、およびDigilent Nexys A7ボード用SweRVolf Nexys。

1. SweRVolf Sim

SweRVolf Simはシミュレーションターゲットであり、これによって、SweRVolfコアがテストベンチでラップされて、Verilatorまたはその他のイベント駆動シミュレータ（QuestaSimなど）によって使用されます。これは、SweRVプロセッサで実行されるプログラムを実行するフルシステムシミュレーション用に、使用できます。これは、デバuggaのOpenOCDおよびJTAG VPIを介する接続にも、対応します（図2を参照）。

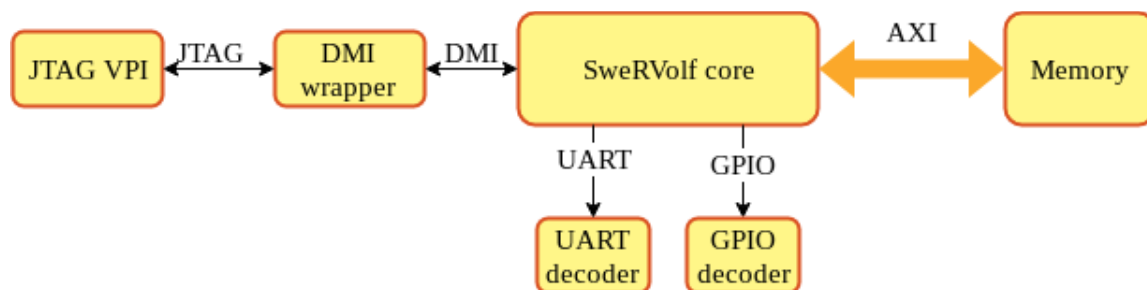


図2 : SweRVolf Sim

2. SweRVolf Nexys

SweRVolf Nexysは、Digilent Nexys A7ボードをターゲットとするSweRVolf SoCの1バージョンです。これによってRAM用オンボード128 MB DDR2が使用され、GPIOがLEDに接続されており、SPIフラッシュからの起動がサポートされ、UARTおよびJTAGの通信用にmicro-USBポートが使用されます。SweRVolf Nexysターゲット用のデフォルトのブートローダにより、デフォルトでSPIフラッシュに保管されているプログラムのロードが、試行されます（図3）。

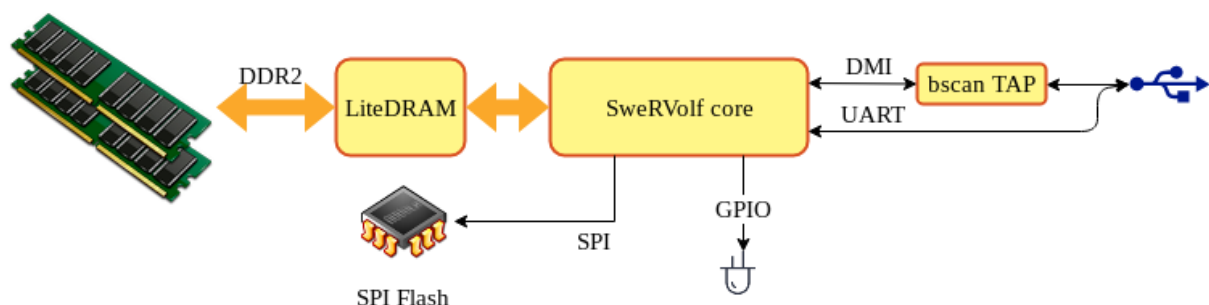


図3 : SweRVolf Nexys

SweRVolf SoCはシミュレーションまたはハードウェアで実行できます - つまり、Digilent Nexys A7ボードで実行できます。どちらの場合も、FuseSoCは、シミュレーションの起動、またはFPGA構築の構築および実行に、使用できます。

前のラボのRVfpgaSimのように、**SweRVolf Sim**はシミュレーションのターゲットであり、これによってSweRVolfコアがVerilatorによって使用されるように、テストベンチでラップされます。同様に、前のラボのRVfpgaNexysは**SweRVolf Nexys**と同様であり、Digilent Nexys A7ボードのターゲットであるSweRVolf SoCの1バージョンです。

5. 環境をセットアップ

このセクションでは、Sim構築およびNexys構築の環境のセットアップ方法を、示します。

ステップ1: プロジェクトのルートとして使用するため、「**SweRVolf**」という名前のディレクトリへ移動します。

このディレクトリは、以後\$WORKSPACEと呼ばれます。特に明記されないかぎり、今後のコマンドはすべて、\$WORKSPACEから実行されます。ワークスペースのディレクトリを入力したら、実行します。

```
> export WORKSPACE=$(pwd)
```

\$WORKSPACEシェル変数を設定します（図4を参照）。以下のコマンドを使用して、シェル変数が正常に追加されたかを確認できます。

```
> printenv WORKSPACE
```

```
hamza@imagination:~$ cd RVfpgaSoC/Labs/LabProjects/SweRVolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export WORKSPACE=$(pwd)
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ printenv WORKSPACE
/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図4：ワークスペースの確立

ステップ2: FuseSoCが既にインストールされていることを、確認します。FuseSoCがマシンにインストールされていない場合は、以下を実行してインストールします。

```
> sudo pip3 install fusesoc
```

ステップ3: FuseSoCベースライブラリを次記のワークスペースに追加します。

```
> fusesoc library add fusesoc-cores
https://github.com/fusesoc/fusesoc-cores
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc library add fusesoc-cores https://github.com/fusesoc/fusesoc-cores
INFO: Cloning library into fusesoc_libraries/fusesoc-cores
Cloning into 'fusesoc_libraries/fusesoc-cores'...
remote: Enumerating objects: 202, done.
remote: Counting objects: 100% (202/202), done.
remote: Compressing objects: 100% (140/140), done.
remote: Total 651 (delta 85), reused 163 (delta 50), pack-reused 449
Receiving objects: 100% (651/651), 136.59 KiB | 81.00 KiB/s, done.
Resolving deltas: 100% (229/229), done.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図5：FuseSoCベースライブラリの追加

ステップ4: swervolfライブラリを追加します。

```
> fusesoc library add swervolf
https://github.com/chipsalliance/Cores-SweRVolf
```



```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc library add swervolf https://github.com/chipsalliance/Cores-SweRVolf
INFO: Cloning library into fusesoc_libraries/swervolf
Cloning into 'fusesoc_libraries/swervolf'...
remote: Enumerating objects: 130, done.
remote: Counting objects: 100% (130/130), done.
remote: Compressing objects: 100% (81/81), done.
remote: Total 1035 (delta 52), reused 89 (delta 42), pack-reused 905
Receiving objects: 100% (1035/1035), 1.17 MiB | 206.00 KiB/s, done.
Resolving deltas: 100% (580/580), done.
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図6 : swervolfライブラリの追加

ステップ5 : swervolfディレクトリを次記の「SWERVOLF_ROOT」シェル変数に設定します。

```
> export
    SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ export SWERVOLF_ROOT=$WORKSPACE/fusesoc_libraries/swervolf
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$
```

図7 : シェル変数の設定

6. シミュレーションの構築

このセクションでは、FuseSoCを使用してSweRVolf Simを、以下のステップで構築する方法を示します。

何を実行するかを選択するには、--targetパラメータで「**fusesoc run**」コマンドを使用します。

ステップ6 : シミュレーションでの実行に、以下を使用します :

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

このコマンドにより、以下のパス内に「**Vswervolf_core_tb**」と呼ばれるシミュレーションバイナリが、生成されます。SweRVolf/build/swervolf_0.7.3/sim-verilator/

これは、ラボ2で、次記のパスでメイクコマンドを使用して「**Vrvfpgasim**」シミュレーションバイナリを生成したときに行ったのと同様であることに、注意してください。
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM

ただし、ラボ2で使ったSoCはSweRVolfXのサブセットでしたが、ここで使用するSoCはSweRVolf一式です。

```

hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=sim swervolf
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
=====
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 hello.vh
Releasing reset
SweRV+FuseSoC rocks

Finito
- ../src/swervolf_0.7.3/rtl/swervolf_syscon.v:151: Verilog $finish
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$

```

図8 : SweRVolf Simの構築

このコマンドにより、ワークスペースディレクトリに以下の階層構造が作成されます。

- \$WORKSPACE
 - fusesoc_libraries
 - build
 - swervolf_0.7.3
 - sim-verilator
 - Vswervolf_core_tb
 - src

7. Nexys A7の構築

このセクションでは、FuseSoCを使用してSweRVolf Nexysを構築する方法を、示します。

ステップ1 : Nexys A7ボード用のイメージを構築（およびオプションでプログラミング）し、実行する方法

➤ `fusesoc run --target=nexys_a7 swervolf`

注意：このコマンドを実行する前に、必ずNexys A7ボードをマシンに接続し、ボードがオンになっていることを確認します。

このコマンドにより、「**swervolf_0.7.3.bit**」と呼ばれるFPGAをプログラムするために使用するビットストリームファイル（「.bit」で終わる）が、以下のパス内に生成されます。
SweRVolf/build/swervolf_0.7.3/nexys_a7-vivado/

これにより、マシンに接続されているNexys A7ボードへのビットストリームも更新されます。

これは、ラボ1で、「Generate Bitstream」オプションを使用してブロック設計を作成した後、Vivadoで「**rvfpga.bit**」ファイルを作成したときに実施したことと、同様です。ラボ2でその後、PlatformIOを使用して「**rvfpga.bit**」ファイルをNexys A7ボードにアップロードしました。

FuseSoCにより、ラボ1およびラボ2で上記のコマンドを使用して手動で実施したこの退屈な作業すべてが、実行されます。

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ fusesoc run --target=nexys_a7 swervolf
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 pulp-platform.org::common_cells:1.20.0
INFO: Preparing ::serv:1.0.2
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 swervolf:litedram: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
```

.....

```
FuseSoC Xilinx FPGA Programming Tool
=====
INFO: Programming part xc7a100tcs9324-1 with bitstream swervolf_0.7.3.bit
INFO: [Labtools 27-2285] Connecting to hw_server url TCP:localhost:3121
INFO: [Labtools 27-2222] Launching hw_server...
INFO: [Labtools 27-2221] Launch Output:

***** Xilinx hw_server v2019.2
**** Build date : Nov 6 2019 at 22:13:42
** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: [Labtools 27-3415] Connecting to cs_server url TCP:localhost:3042
INFO: [Labtools 27-3417] Launching cs_server...
INFO: [Labtools 27-2221] Launch Output:

***** 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9 : SweRVolf Nexysの構築

上記のコマンドの完了後、ワークスペースディレクトリは以下のようになります。

- \$WORKSPACE
 - fusesoc_libraries
 - build
 - swervolf_0.7.3
 - sim-verilator
 - nexys_a7-vivado
 - **swervolf_0.7.3.bit**
 - src

FuseSoCでは**Vivado**が使用され、プロジェクトが作成され、グローバル変数が設定され、制約ファイルが追加され、ビットストリームが自動的に追加されます。

Vivadoプロジェクトを開いて階層構造をラボ1で見たように可視化できます。

ステップ2 : 以下のコマンドによって**nexys_a7-vivado**ディレクトリを入力します。

```
> cd build/swervolf_0.7.3/nexys_a7-vivado/
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd build/swervolf_0.7.3/nexys_a7-vivado/
~/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/nexys_a7-vivado$
```

図10 : 「nexys_a7-vivado」ディレクトリに移動する

ステップ3 : 以下のコマンドを入力して、Vivadoプロジェクトを開きます。

```
> vivado swervolf_0.7.3.xpr
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/nexys_a7-vivado$ vivado swervolf_0.7.3.xpr
***** Vivado v2019.2 (64-bit)
**** SW Build 2708876 on Wed Nov  6 21:39:14 MST 2019
**** IP Build 2700528 on Thu Nov  7 00:09:20 MST 2019
** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

start_gui
```

図11 : Vivadoプロジェクトを開く

ソースパネルに設定されているグローバル変数および制約ファイルを、可視化できます。

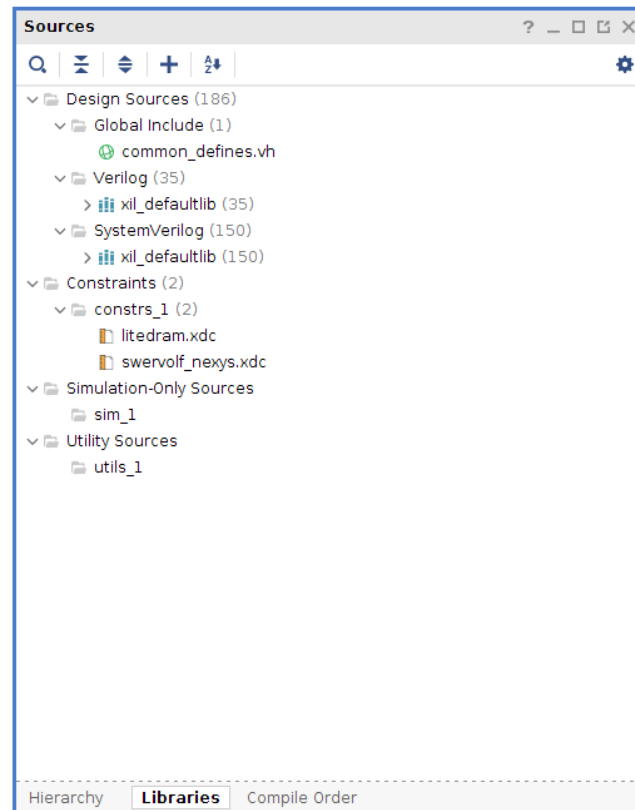


図12 : ソースパネル

「Design Runs」タブに、合成および実装が既に正常に実行されていることが、表示されます。

Tcl Console Messages Log Reports Design Runs x															
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP	
✓ synth_1	constrs_1	synth_design Complete!								34263	18537	44.0	0	4	
✓ impl_1	constrs_1	write_bitstream Complete!	0.453	0.000	0.052	0.000	0.000	0.929	0	33663	18538	44.0	0	4	

図13 : 実行を設計

8. SweRVolf SimでのAL_Operations例の実行

このセクションでは、Verilatorを使用してSweRVolf SimでAL_Operationsプログラムを実行する方法を、示します。

ステップ1: パスが次記であるディレクトリ例を入力します。

```
[RvfpgaSoCPath]/RvfpgaSoC/Labs/LabResources/Lab3/examples/  
AL_Operations/CommandLine/
```

```
> cd ../../LabResources/Lab3/examples/AL_Operations/commandLine/
```

```
~/RVfpgaSoC/Labs/LabProjects/SweRVolf$ cd ../../LabResources/Lab3/examples/AL_Operations/commandLine/  
~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

図14 : AL_Operationsディレクトリ

ステップ2: シミュレーション用のhexadecimalプログラムを作成 :

```
> make clean
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ make clean  
rm -f *.elf *.bin *.vh *.dis *.mem *.vcd  
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

図15 : make clean

```
> make AL_Operations.vh
```

```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ make AL_Operations.vh  
/home/hamza/.platformio/packages/toolchain-riscv/bin/riscv64-unknown-elf-gcc -nostartfiles -march=rv32i -mabi=ilp32 -  
Tlink.ld -oAL_Operations.elf AL_Operations.S  
/home/hamza/.platformio/packages/toolchain-riscv/bin/riscv64-unknown-elf-objcopy -O binary AL_Operations.elf AL_Opera  
tions.bin  
python3 makehex.py AL_Operations.bin > AL_Operations.vh  
rm AL_Operations.bin AL_Operations.elf  
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$
```

図16 : AL_Operations.vhファイルの作成

ステップ3: 次記のシミュレータを実行します。

```
> ../../../../../../LabProjects/SweRVolf/build/swervolf_0.7.3/sim-  
verilator/Vswervolf_core_tb +ram_init_file=AL_Operations.vh +vcd=1
```

「ram_init_file」パラメータによってVerilog hexファイルが、オンチップRAMの初期内容として使用するために、ロードされます。オプション (+ram_init_file) を使用して作成したばかりのVerilog hexファイル「AL_Operations.vh」をロードして、シミュレーションバイナリ「Vswervolf_core_tb」を実行します。vcd (値変更ダンプ) も1に設定して、モジュール内のすべての変数をダンプします。

```
hamza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ ../../../../../../LabProjects/SweRVolf  
/build/swervolf_0.7.3/sim-verilator/Vswervolf_core_tb +ram_init_file=AL_Operations.vh +vcd=1  
Loading RAM contents from AL_Operations.vh  
Releasing reset
```

図17 : シミュレータを実行

「ctrl + c」を押してシミュレーションを停止します。「trace.vcd」ファイルを生成する必要があります。

ステップ4 : 次記のトレースファイルを開きます。

➤ `gtkwave trace.vcd`

```
hanza@imagination:~/RVfpgaSoC/Labs/LabResources/Lab3/examples/AL_Operations/commandLine$ gtkwave trace.vcd

GTKWave Analyzer v3.3.86 (w)1999-2017 BSI

Warning! File size is 758 MB. This might fail in recoding.
Consider converting it to the FST database format instead. (See the
vcd2fst(1) manpage for more information.)
To disable this warning, set rc variable vcd_warning_filesize to zero.
Alternatively, use the -o, --optimize command line option to convert to FST
or the -g, --giga command line option to use dynamically compressed memory.

[0] start time.
[2481540] end time.
```

図18 : gtkwaveでtrace.vcdファイルを開く

これで、**gtkwave**が起動します。ラボ2のプロセスを繰り返してグラフに信号を追加し、これを分析する必要があります。

ステップ5 : 信号をグラフに追加できるように、GTKWaveの左上のペインで、SoC階層構造を展開します。階層構造をTOP → swervolf_core_tb → swervolf → rvtop → swervに展開し、モジュールifuをクリックし、信号clk（コアに使用されるクロック）を選択し、これを白色のSignalsペインまたは右の黒色のWavesペインの中に、ドラッグします。

ステップ6 : ifuモジュールが強調表示されていることを確認し（図19を参照）、フィルタ検索に「inst」を入力してから、「ifu_i0_instr[31:0]」および「ifu_i1_instr[31:0]」信号を挿入します。

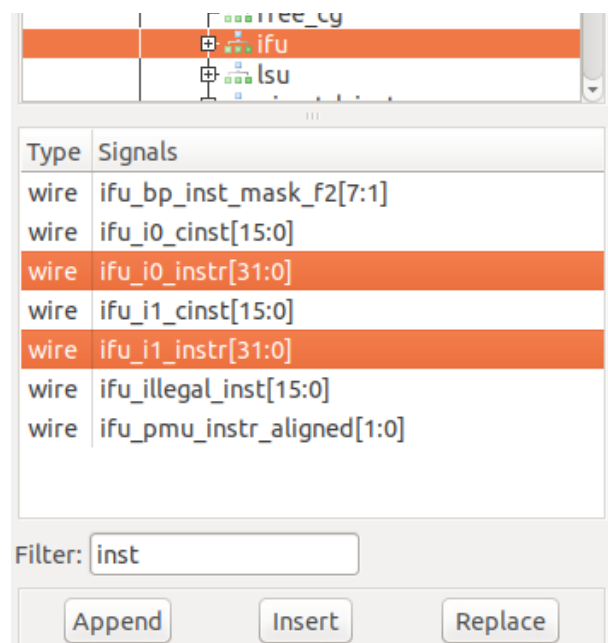


図19 : 「instr」 信号

ステップ7 : これからTOP → swervolf_core_tb → swervolf → rvtop → swerv → dec → arf → gpr_banks(0) → gpr(28) → gprffに移動して、続いて「dout[31:0]」信号をクリックし、これを挿入します（図20を参照）。

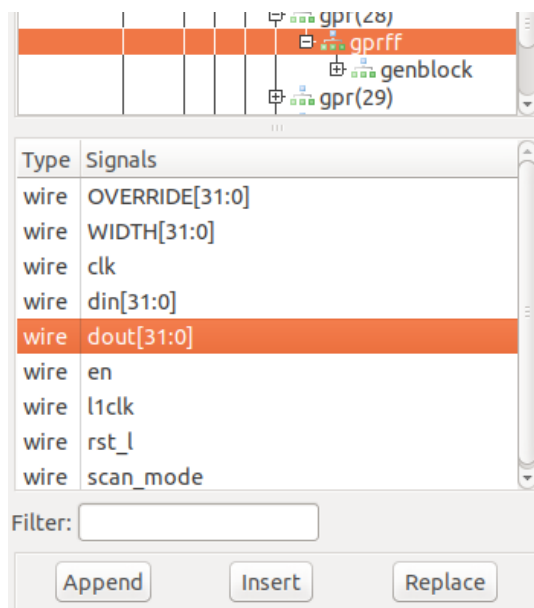


図20 : 「dout」 信号

「+」 ボタンを使用して信号でズームインでき、波形を分析できます（図21を参照）。

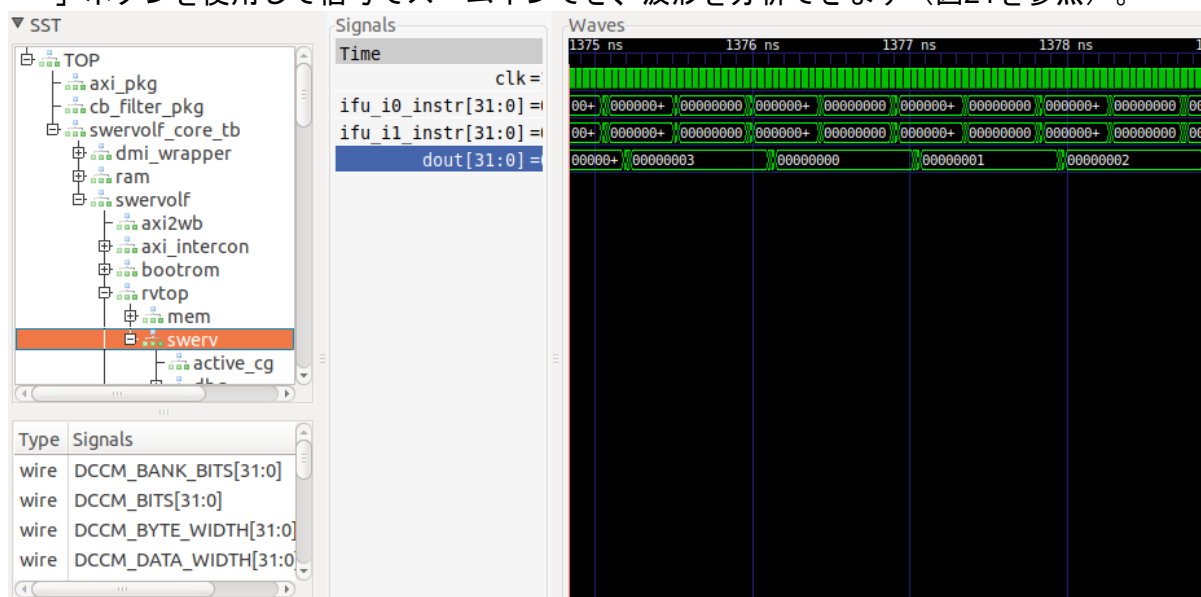


図21 : 信号を分析

9. Blinky例をSweRVolf Nexysで実行

このセクションでは、SweRVolf NexysでBlinkyプログラムを実行する方法を、示します。このラボの前半でFuseSoCによって生成された「.bit」ファイルを使用します。

注意： ラボ1およびラボ2で使用するSweRVolfXモジュール用のLEDおよびI/Oのアドレスメモリマップは、SweRVolf（SweRV EH1用FuseSoCベースのSoC）とは異なります。このため、ラボ2およびラボ3の同じプログラム例（Blinky）の、メモリアドレスが変わっている別個のディレクトリ例があります。

表5 : SweRVolfおよびSweRVolfXのメモリマップGPIOアドレス

SoC	システム	アドレス
SweRVolfX (ラボ1)	GPIO	0x80001400 - 0x8000143F
SweRVolf (ラボ3)	GPIO	0x80001010 - 0x80001013

以下のステップを実行して、SweRVolf Nexysを使用してNexys A7ボードをプログラムし、Blinkyプログラムを実行します。

ステップ1 : Nexys A7ボードをコンピュータに接続します。

ステップ2 : 左上のスイッチを使用して、Nexys A7ボードの電源を入れます。(図22を参照)

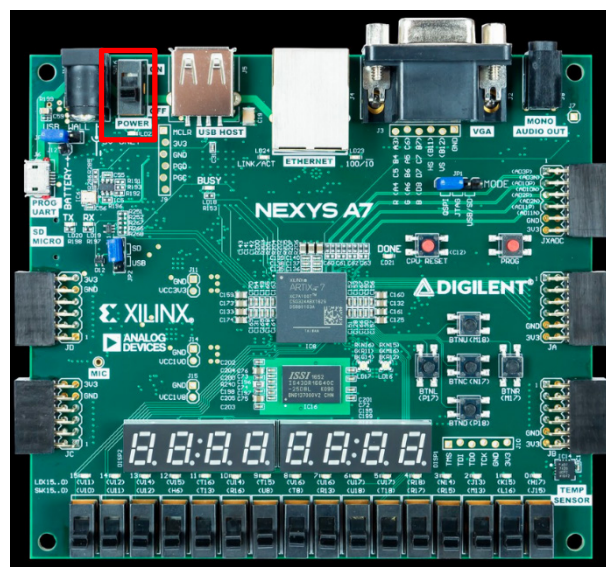


図22 : Nexys A7ボードのON/OFFボタン

ステップ3 : VSCodeおよびPlatformIOがまだ開いていない場合は、これを開きます。

ステップ4 : 上部のメニューバーで、File → Open Folder (図23を参照) をクリックして、ディレクトリ `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab3/examples/` を参照します。

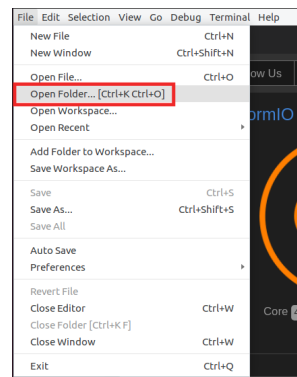


図23：フォルダを開く

ステップ5：ディレクトリ *Blinky_FuseSoC* を選択して（開かないで選択します）、ウィンドウ上部でOKをクリックします。これで、PlatformIOによって例が開かれます。

ステップ6：左のサイドバーの *platformio.ini* をクリックして、*platformio.ini* を開きます（図24を参照）。後続する行を編集して、使用するシステムのRVfpgaビットストリームへのパスを確立します（図24を参照）。

ステップ7：FuseSoCを使用して作成された「*swervolf_0.7.3.bit*」ファイルは、次記のパスにあります。

```
board_build.bitstream_file =
/home/<Username>/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0
.7.3/nexys_a7-vivado/swervolf_0.7.3.bit
```

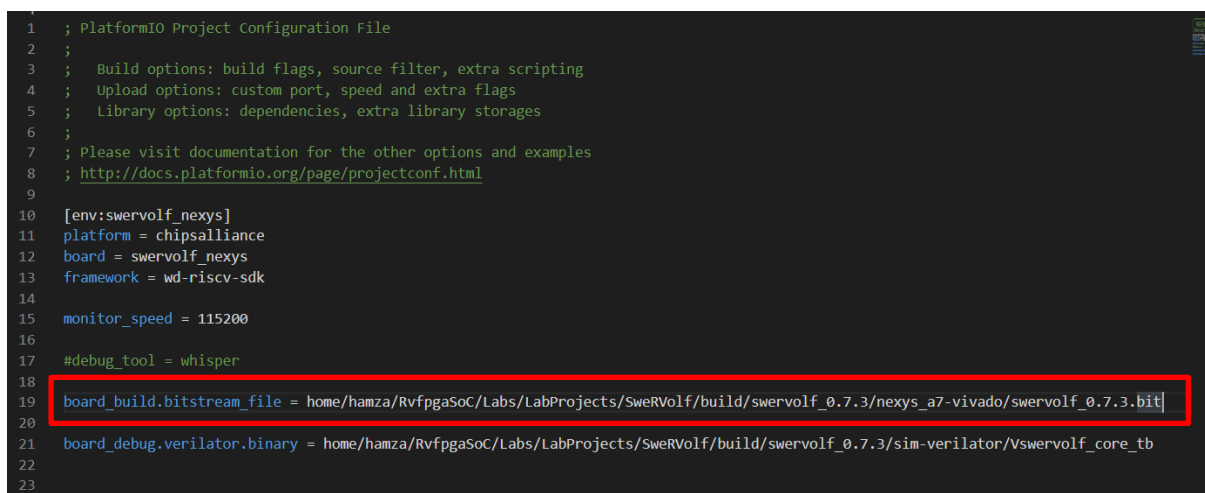


図24：PlatformIO初期化ファイル：platformio.ini



ステップ8：左のメニューボタンでPlatformIOアイコン  をクリックします（図25を参照）。



図25 : PlatformIOアイコン

Project Tasksウィンドウが空の場合（図26）、まずをクリックしてプロジェクトタスクを更新する必要があります。これには数分間かかることがあります。

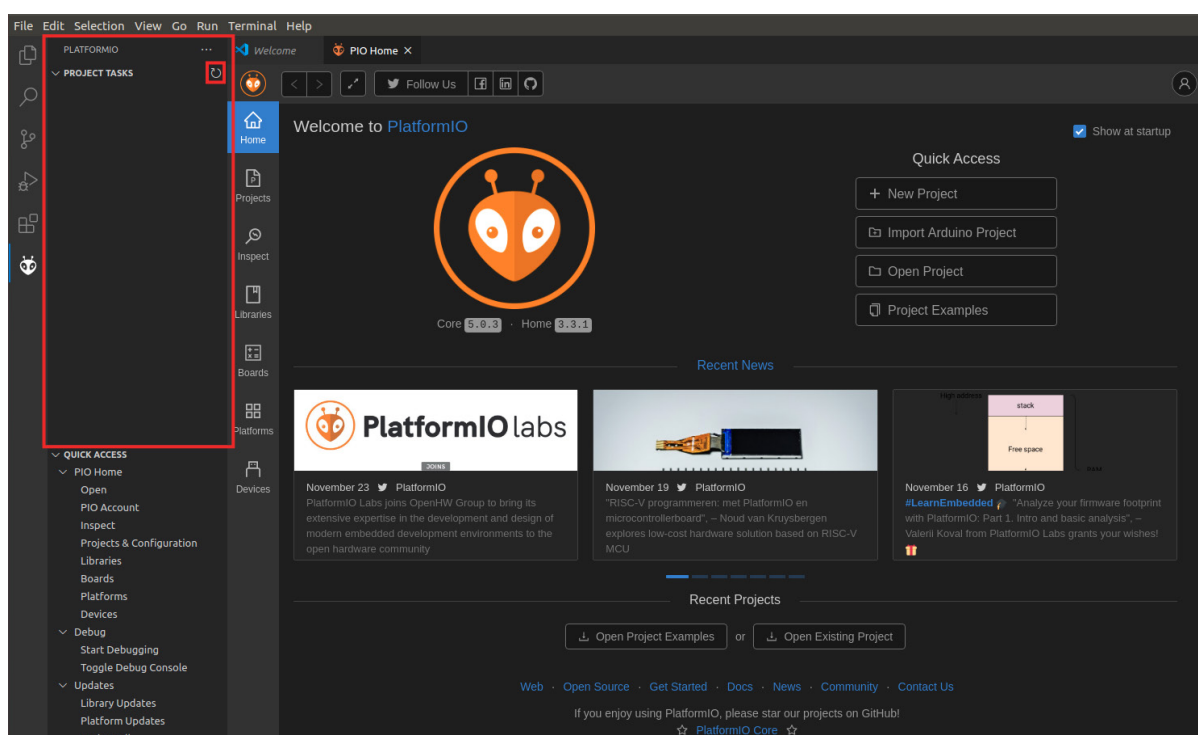


図26 : PROJECT TASKSウィンドウが空 - 更新

ステップ9 : Project Tasks → env:swervolf_nexys → Platformを展開して、図27に示されているように、Upload Bitstreamをクリックします。

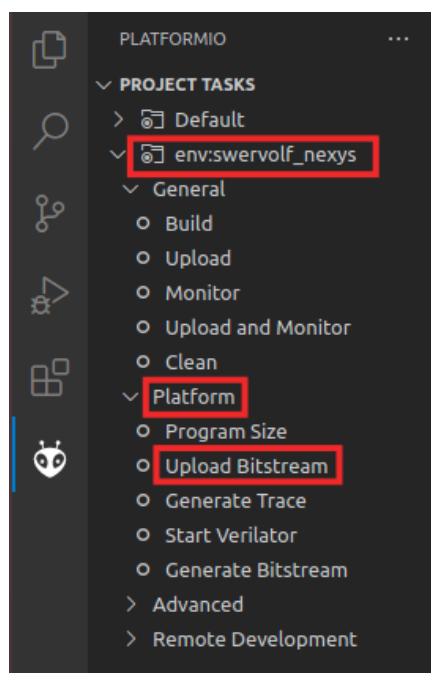


図27 : ビットストリームのアップロード

これでビットストリームがアップロードされており、デバッグプロセスを開始します。

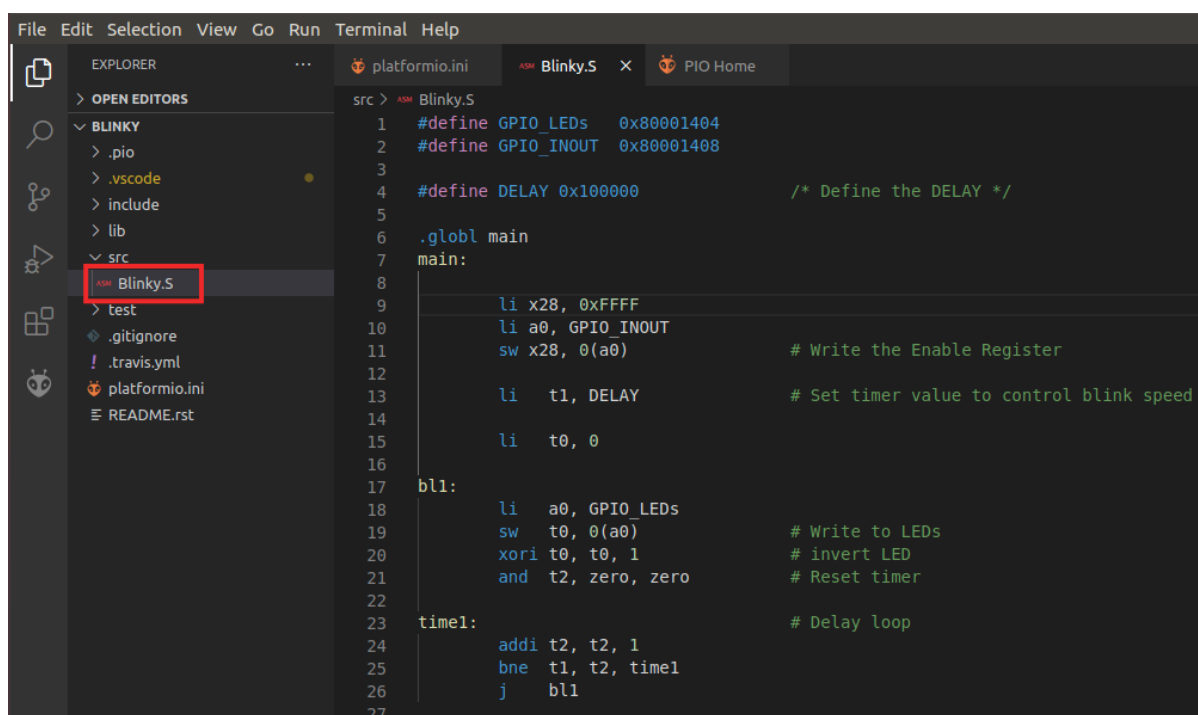

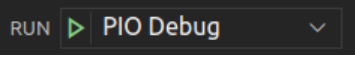



図28 : PlatformIOでのblinky.S

ステップ10 :  をクリックして、プログラムを実行およびデバッグします。次にplayボタン  をクリックしてデバッグを開始します。PlatformIOにより、main関数の最初に一時的なブレークポイントが設定されます。Continueボタン  をクリックして、プログラムを実行します。

ステップ11 : ボードで、右端のLEDが点滅を開始します。

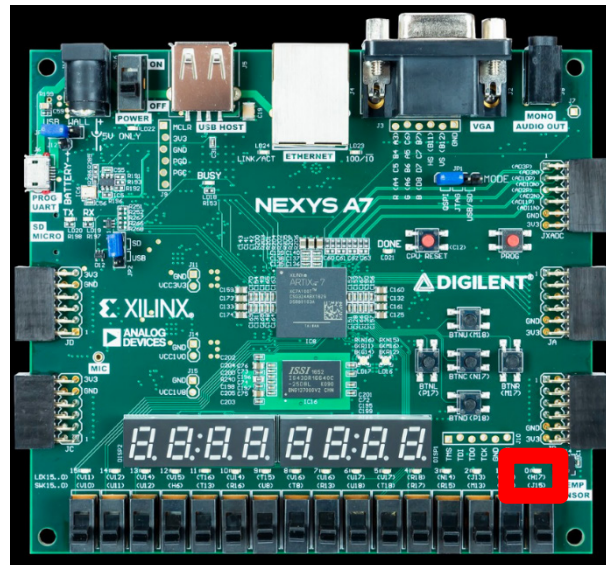
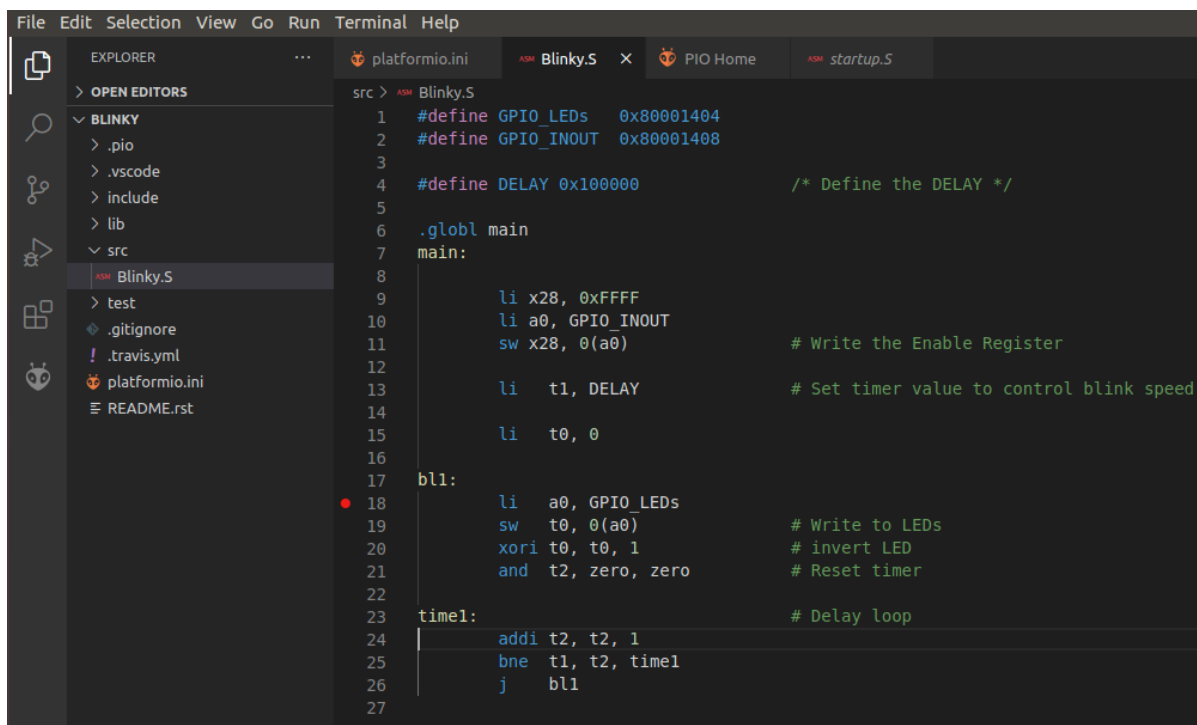


図29 : 右端のLEDの点滅

ステップ12 : pauseボタン  をクリックして、実行を一時停止します。実行は、無限ループ内のどこか（多分time1遅延ループ内）で、停止します。

ステップ13 : 行番号18の左をクリックして、ブレークポイントを確立します。赤色の点が表示され、ブレークポイントがBREAKPOINTSタブに追加されます（図30を参照）。




```

1  #define GPIO_LEDS    0x80001404
2  #define GPIO_INOUT   0x80001408
3
4  #define DELAY 0x100000      /* Define the DELAY */
5
6  .globl main
7  main:
8
9      li x28, 0xFFFF
10     li a0, GPIO_INOUT
11     sw x28, 0(a0)          # Write the Enable Register
12
13     li  t1, DELAY          # Set timer value to control blink speed
14
15     li  t0, 0
16
17  b1:
18     li  a0, GPIO_LEDS
19     sw  t0, 0(a0)          # Write to LEDs
20     xori t0, t0, 1         # invert LED
21     and t2, zero, zero    # Reset timer
22
23  time1:
24     addi t2, t2, 1         # Delay loop
25     bne t1, t2, time1
26     j   b1
27

```

図30 : blinky.Sでのブレークポイントの設定

ステップ14：次に、Continueボタン  をクリックして、実行を続行します。実行が続行され、右端のLEDに1（または0）を書き込む単語登録（sw）命令の後に、停止します。

ステップ15：実行を数回続行します。右端のLEDを駆動する値がその都度変わって表示されます。

ステップ16：デバッグ  を停止し、 をクリックして Explorerウィンドウに戻ります。File → Close Folderを選択して、プログラムを閉じます。

10. SweRVolfのデバッグ

SweRVolfは、ハードウェアおよびシミュレーションの両方でのデバッグに対応します。デバッグの接続にはさまざまな手順がありますが、いったん接続すると、同じコマンドを使用できます（プログラムの実行はシミュレーションではるかに遅くなりますが）。

シミュレーション：

ステップ1：WORKSPACEのディレクトリ「SweRVolf」を入力して、次記のシミュレーションコマンドを実行します。

```
➤ fusesoc run --target=sim swervolf --jtag_vpi_enable
```

「**--jtag_vpi_enable**」パラメータにより、OpenOCDを接続できる先であるJTAGが有効になります。SweRVolfシミュレーションが「**--jtag_vpi_enable**」によって起動されると、これによって、クライアントによって接続され、JTAGコマンドが送信されるのを待っているJTAGサーバが起動されます。


```
make[1]: Leaving directory '/home/hamza/RVfpgaSoC/Labs/LabProjects/SweRVolf/build/swervolf_0.7.3/sim-verilator'
INFO: Running
INFO: Running simulation
Starting jtag_vpi server: interface 127.0.0.1 (loopback), port 5555/tcp ...
jtag_vpi server created.
Waiting for client connection...
```

図31 : シミュレーションコマンドを実行

ステップ2 : 「Ctrl + Shift + t」を使用して新しいターミナルを開いてから、次記を実行することによって、Workspaceディレクトリからデータディレクトリに移動します。

```
> cd fusesoc_libraries/swervolf/data/
```

ステップ3 : 次記のコマンドを使用して、OpenOCDをシミュレーションインスタンスに接続します。

```
> openocd -f swervolf_sim.cfg
```

正常に行われると、OpenOCDによって以下が出力されるはずです（図32を参照）。

```
hamza@imagination:~/RVfpgaSoC/Labs/LabProjects/SweRVolf/fusesoc_libraries/swervolf/data$ openocd -f swervolf_sim.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
Info : only one transport option; autoselect 'jtag'
Info : Set server port to 5555
Info : Set server address to 127.0.0.1
Info : Connection to 127.0.0.1 : 5555 succeed
Info : This adapter doesn't support configurable speed
Info : JTAG tap: riscv.cpu tap/device found: 0x00000001 (mfg: 0x000 (<invalid>), part: 0x0000, ver: 0x0)
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32 : OpenOCDの実行

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、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/fusesoc_libraries/swervolf/data$ telnet localhost 4444
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^['.
Open On-Chip Debugger
>
```

図33 : telnet

ステップ5 : コマンドを入力して、命令します。3番目のターミナルを使用して、SweRVolfにライブの命令を与えることができます。

このFuseSoCベースのSoCでは、16のLEDがメモリマップGPIOによってアドレス0x80001010-0x80001011でコントロールされます。これらのアドレスはラボ1およびラボ2で使ったRVfpgaシステムモジュールとは別です。

メモリで、アドレス指定する値「1」を書き込むことができます。

```
> mwb 0x80001010 1
```

最初のターミナルを開くと、gpio0がONで表示されします（図34を参照）。

```
jtag_vpi server created.
Waiting for client connection...
Client connection accepted.
JTAG VPI enabled. Not loading RAM
Releasing reset
298548240: gpio0 is on
```

図34 : gpio0がON。

再び3番目のターミナルに戻って、同じメモリアドレスで値「0」を書き込んで、gpio0をOFFにします。

```
> mwb 0x80001010 0
```

```
jtag_vpi server created.
Waiting for client connection...
Client connection accepted.
JTAG VPI enabled. Not loading RAM
Releasing reset
298548240: gpio0 is on
528686180: gpio0 is off
```

図35 : gpio0がOFF。

Nexys A7ボードで同じプロセスを完了します。

ハードウェア :

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。ビットストリームを、Nexys A7ボード用に再構築せずに再度アップロードするには、「--run」パラメータを追加できます。

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

```
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36 : 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`

図37に示されている出力が得られるはずですが。

```
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37 : 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38 : 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、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39 : telnet

ステップ5：コマンドを入力して、ボードに直接命令を与えます。3番目のターミナルを使用して、SweRVolfにライブの命令を与えることができます。

16のLEDが、メモリマップGPIOによってアドレス0x80001010-0x80001011でコントロールされます。

メモリで値「1」を0x80001010に書き込んで、右端のLEDをオンにします。

```
➤ mwb 0x80001010 1
```

次記のコマンドを実行して、右端のLEDをオフにします。

```
➤ mwb 0x80001010 0
```

これによって、ボードの右端のLEDがオフになります。

このラボを急いでサット振り返るには、最初にブロック設計コードをハンドチューンコードと比較しました。次にSweRVolf（SweRV Eh1用FuseSoCベースのSoC）パッケージが含まれているパッケージマネージャであるFuseSoCの説明を受けました。

次にその2つのバージョン「SweRVolf Sim」および「SweRVolf Nexys」を詳しく調べました。

前のラボのRVfpgaSimのように、**SweRVolf Sim**はシミュレーションのターゲットであり、これによってSweRVolfコアがVerilatorによって使用されるように、テストベンチでラップされます。同様に、前のラボのRVfpgaNexysは**SweRVolf Nexys**と同様であり、Digilent Nexys A7ボードのターゲットであるSweRVolf SoCの1バージョンです。

次に、FuseSoCの実行を使用して、シミュレーション用SweRVolf Sim、およびNexys A7ボード用SweRVolf Nexysを構築しました。その後、RVfpgaSimおよびRVfpgaNexysに関する以前のラボで実行したこれらのビルドで、同じ例を実行しました。