



THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga-SoCラボ2

RVfpga-SoCでの ソフトウェアの実行

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、インターコネクト（AXI インターコネクト、Wishboneインターコネクト、AXI-to-Wishboneブリッジ）、SPIコントローラが追加されます。 SweRVolfX ：これにより次記の4つの新しい周辺機器がSweRVolfに追加されます：GPIO、PTC、追加のSPI、および8桁の7セグメント表示用コントローラ。
RVfpgaNexys	Nexys A7ボードおよびその周辺機器を対象とするSweRVolfX SoC。これにより、DDR2インターフェース、CDC（クロックドメイン交差）ユニット、BSCANロジック（JTAGインターフェース用）、クロックジェネレータが追加されます。 RVfpgaNexysはSweRVolf Nexys（ https://github.com/chipsalliance/Cores-SweRVolf ）と同じですが、例外として後者はSweRVolfに基づいています。
RVfpgaSim	シミュレーションを目的とした、テストベンチラッパおよびAXIメモリ付きSweRVolfX SoC。 RVfpgaSimはSweRVolf Sim（ https://github.com/chipsalliance/Cores-SweRVolf ）と同じですが、例外として後者はSweRVolfに基づいています。

1. はじめに

このラボでは、ラボ1でVivadoブロック設計ツールを使用して作成したSweRVolfXサブセットでC言語またはアセンブリ言語で書かれたプログラムを実行する方法を、示します。Verilatorを使用してこの設計をシミュレーションするか、Nexys A7ボードでこの設計を実行するかを、選択できます。FPGAボードへのアクセス権がない場合は、Verilatorを使用するシミュレーションでのみ、このラボを完了できます。このラボを完了するには、ブロック設計の「BD.v」 Verilogファイル、およびラボ1でVivadoのブロック設計を使用して生成された「rvfpga.bit」ビットファイルを使用します。

このラボでは、RVfpgaSim用のシミュレーションバイナリを生成する方法を示します。このシミュレーションバイナリは、後でプログラム例のシミュレーショントレースを作成するために、使用されます。GTKWaveを使用してシミュレーショントレースの分析もします。

オプションのステップとして、ラボ1で作成したビットストリームによって定義されるようにRVfpgaNexysをNexys A7ボードにPlatformIOを使用してダウンロードし、PlatformIOを使用してプログラム例をデバッグする方法を示します。このステップはオプションですが、推奨します。

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。

- VSCode (インストールガイド (06ページ) を参照)
- PlatformIO (インストールガイド (06ページ) を参照)
- GTKWave (インストールガイド (08ページ) を参照)
- Verilator (インストールガイド (08ページ) を参照)
- Cygwin (Windowsユーザー専用) (インストールガイド (13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、まだ実行していない場合は、VScode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. ブロック設計で作成されたSoCの実行

この最初のラボでは、プロセッサコア、インターコネクト、周辺機器を、Vivadoのブロック設計を使用して相互に接続することによって、SweRVolfX SoCのサブセットを作成しました。次に、ブロック設計によって、ブロック設計モジュール全体としてのVerilogファイルが生成されます。この場合、生成されたのは「BD.v」ファイルでした。

ブロック設計のSoCを実行するには2つのオプションおよび経路があります。

- ブロック設計のSoCをNexys A7 100Tボードで実行する。
- ブロック設計のSoCをVerilatorシミュレータで実行する。

ブロック設計のSoCには2つの最上位レベルのモジュールがあり、これは以下に説明されている次記のターゲットそれぞれ用です：RVfpgaSim (rvfpgasim) およびRVfpga Nexys (rvfpga)。

1. RVfpgaSim (rvfpgasim.v)

RVfpgaSimモジュールは、シミュレーション用RVfpgaシステムの最上位モジュールとして使用されます。RVfpgaシステムのシミュレーションにVerilator（RVfpgaを定義するVerilogをシミュレーションするハードウェア記述言語（HDL）シミュレータ）を使用します。シミュレーションでSoCを実行することにより、システムの内部信号を掘り下げて分析できます。このラボで後ほど、RVfpgaSim用シミュレーションバイナリを生成するとき、「rvfpgasim.v」を最上位モジュールファイルとして使用します。

最上位モジュール「rvfpgasim」の構造が図1に示されています。

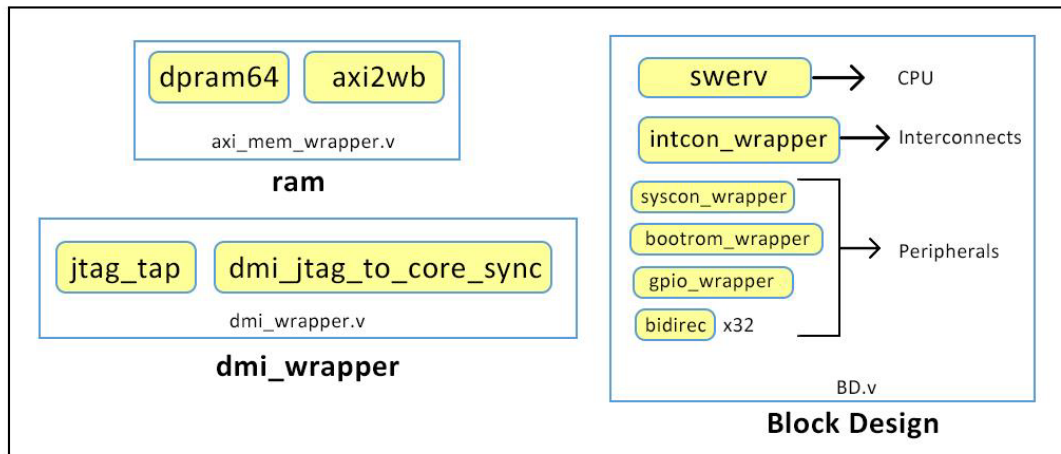


図1 : RVfpgaSim

これには以下の3つのモジュールが含まれています。

- ブロック設計 (BD.v)
これは、Vivadoのブロック設計を使用して作成したSoCモジュールです。
- ram (axi_mem_wrapper.v)
これはメモリモジュールです。
- dmi_wrapper (dmi_wrapper.v)
これはデバッグモジュールのインターフェースです。

ラボ1では、ピンをVivadoのブロック設計を使用して接続するとき、いくつかの外部ピン接続をしました。「ブロック設計」モジュールのこれらの外部接続は、最上位モジュール「rvfpgasim」で他のモジュールと接続されます。例えば、「ブロック設計」モジュールの「DMI」外部接続は「dmi_wrapper」モジュールと接続され、「ブロック設計」モジュールの「RAM」外部接続は「ram」モジュールと接続されます。

2. RVfpga Nexys (rvfpga.sv)

RVfpga Nexysモジュールはハードウェア用のRVfpgaシステムの最上位モジュールとして使用され（オンボード実装）、Digilent Nexys A7ボード（または相互に交換可能な古いNexys 4 DDRボード）用です。

最上位モジュール「rvfpga.sv」の構造が図2に示されています。

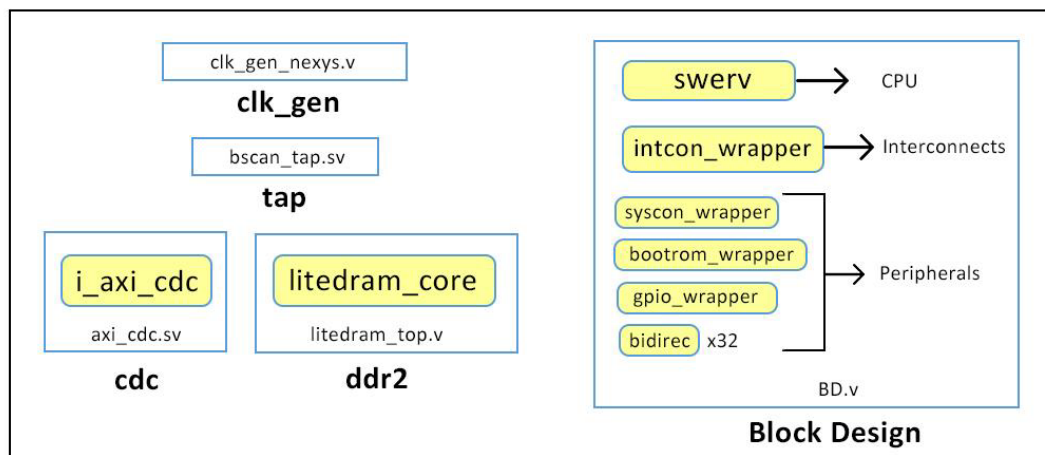


図2 : RVfpga Nexys

RVfpga Nexysには以下の5つのモジュールが含まれています。

- ブロック設計 (BD.v)
これは、ブロック設計を使用して作成したSoCモジュールです。
- ddr2 (litedram_top.v)
これはDDRメモリコントローラモジュールです。
- clk_gen (clk_gen_nexys.v)
これはクロックジェネレータモジュールです。
- tap (bscan_tap.v)
これはjtagデバッグモジュールです。詳細については、次記の[リンク](#)を参照してください。
- cdc (axi_cdc.v)
これはクロック領域交差モジュールです。

「rvfpga.v」最上位モジュールで、「ブロック設計」モジュールの「RAM」外部接続は、「ddr2」モジュールと接続されます。「ブロック設計」の「DMI」外部接続は、「bscan_tap.v」モジュールと接続されます。「clk」外部接続は、「clk_gen」モジュールと接続されます（図3を参照）。

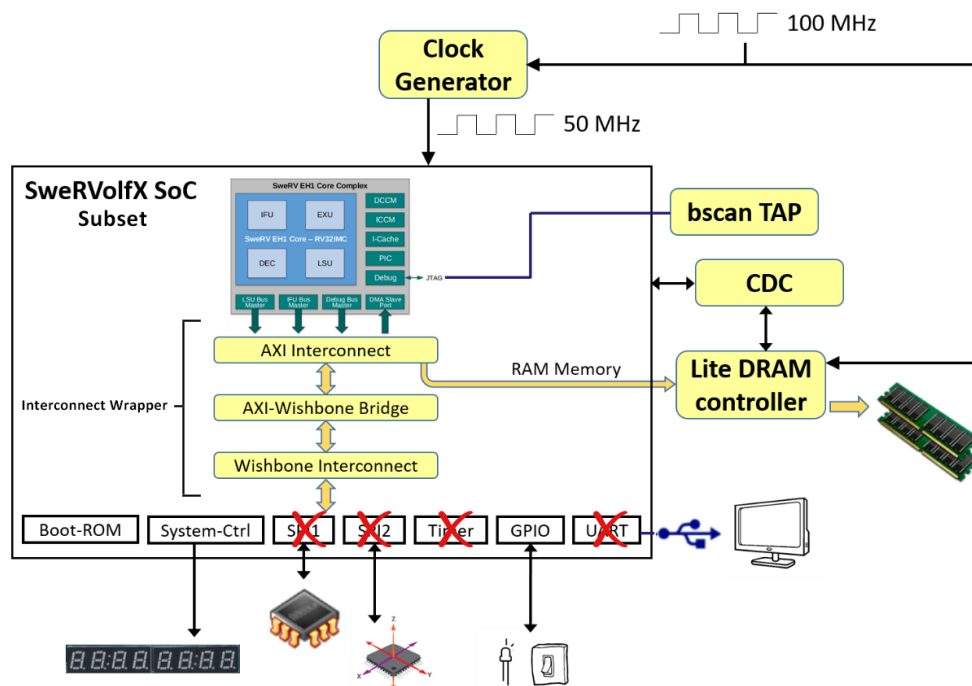


図3 : RVfpga Nexys (SweRVolfX SoCサブセット)

4. Verilatorでのプログラムの実行

このセクションで皆様は、Verilatorを使用してRVfpgaSim で、最初のプログラム (AL_Operations) を実行する方法のプロセスを、体験されます。

注意：すべてのRVfpga Verilogソースモジュールは、ブロック設計で生成された「BD.v」ファイルで機能するために、接頭辞「BD_」が付いている必要があります。この「BD.v」ファイルは、Verilatorでのシミュレーション用最上位モジュール「rvfpgasim」によって使用されます。このソースモジュールは「BD.v」ファイルでインスタンス化され、すべてのモジュールに接頭辞「BD_」が付いている必要があります。これは別個の「src」フォルダで、次記のパスで既に実施しました。

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab2/src.

まず、「ブロック設計」モジュールファイルBD.vを、最上位モジュールファイル「rvfpgasim.v」などの他のソースファイルすべてが含まれているフォルダへ、移動する必要があります。

以前のラボでHDLラップを作成したとき、そのフルパスが提供されました（図4を参照）。このパスに移動して、このファイルをコピーする必要があります。

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabProjects/Lab1/Lab1.srcs/sources_1/bd/BD/synth/BD.v

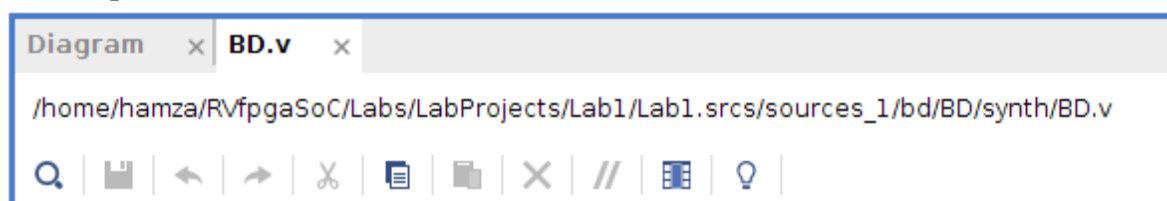


図4 : 「ブロック設計」モジュールの「BD.v」Verilogファイルのパス

ステップ1: (図4) に示されているパスから「BD.v」 ファイルをコピーして、「BD.v」 ファイルを次記のパスに貼り付けます (図5を参照)。

[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab2/src/SweRVolfSoC/

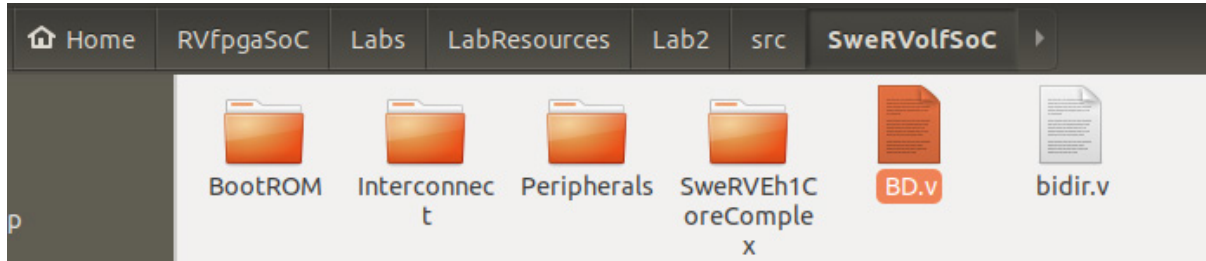


図5 : 「SweRVolfSoC」 ディレクトリに貼り付けられた「BD.v」。

ステップ2: 「BD.v」 ファイルを開いて、以下のモジュールの名前が「_0_0」で終わることを確認します (図6を参照)。

- BD_bootrom_wrapper_0_0
- BD_gpio_wrapper_0_0
- BD_intcon_wrapper_bd_0_0
- BD_swerv_wrapper_verilog_0_0
- BD_syscon_wrapper_0_0

注意: これは、シミュレーション用モジュールの命名での一貫性を維持するために、行われます。これらが一貫していない場合は、次のステップでRvfpgaSimのためのシミュレーションバイナリを生成するときに、エラーが表示されます。

```
BD_bootrom_wrapper_0_0 bootrom_wrapper_0
(.i_clk(clk_0_1),
 .i_rst(rst_0_1),
 .i_wb_adr(intcon_wrapper_bd_0_wb_rom_adr_o),
 .i_wb_cyc(intcon_wrapper_bd_0_wb_rom_cyc_o),
 .i_wb_dat(intcon_wrapper_bd_0_wb_rom_dat_o),
 .i_wb_sel(intcon_wrapper_bd_0_wb_rom_sel_o),
 .i_wb_stb(intcon_wrapper_bd_0_wb_rom_stb_o),
 .i_wb_we(intcon_wrapper_bd_0_wb_rom_we_o),
 .o_wb_ack(bootrom_wrapper_0_o_wb_ack),
 .o_wb_rdt(bootrom_wrapper_0_o_wb_rdt));
```

図6 : 「BD.v」

それでは、AL_Operationsプログラムをブロック設計のSoCで実行するプロセスから、始めます。

まず、PlatformIOを使用してシミュレーショントレースを生成してから、クロック、スーパスカラプロセッサの両方向用命令、レジスタ $\times 28$ (つまり、レジスタ $t3$) の信号をシミュレーション波形に追加し、手順のGTKWaveを表示し、プログラム実行時の信号変化を登録します。

これを実行するには、以下のステップを実行します。

ステップ3 : RvfpgaSim用シミュレーションバイナリを生成

ディレクトリ `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM` には、RVfpgaSim用シミュレーションバイナリを生成するための `Makefile` およびスクリプト (`swervolf_0.7.vc`) が含まれます。スクリプトには、とりわけ、SoC用のソースはどこにあるか、今回の場合のどれが次記で利用できるかの、Verilatorが把握する情報が含まれています。 `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/src`。

次に、RVfpgaSim用のバイナリを生成します。これは、後で、RVfpgaで動作するプログラム `AL-Operations` のシミュレーショントレースを作成するために、使用されます。

ターミナルウィンドウで、次記のコマンドを実行してシミュレータバイナリを生成します。

```
➤ cd [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/
  verilatorSIM
➤ make clean
➤ make
```

ファイル `Vrvfpgasim` (RVfpgaシミュレーションバイナリ) は、ディレクトリ `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/verilatorSIM` の中に生成する必要があります。

Windows : Windowsを使用している場合、これらと同じステップを、Cygwinターミナル内で実行する必要があります (詳細な手順については、RVfpga-SoC入門ガイド付録Bを参照)。C: Windowsフォルダは `cygdrive/c` のCygwinの中にあることに、注意してください。このセクションの他の手順すべては、Linux用に記載されているものと同じです。

ステップ4 : PlatformIOからシミュレーショントレースを生成

シミュレーションバイナリ (`Vrvfpgasim`) が生成されたら、プログラム `AL_Operations` のシミュレーショントレース (`trace.vcd`) を生成するために、PlatformIOの中でこれを使用します。

1. 使用するコンピュータでVSCodeを開いてから、PlatformIOを開きます。
2. 上部のバーで `File→Open Folder` (図7) をクリックし、ディレクトリ `[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples` を参照します。

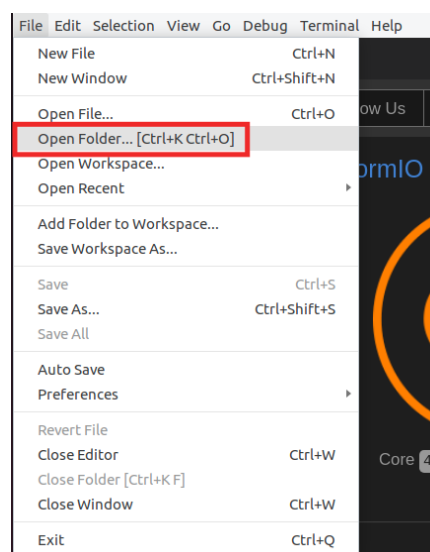


図7 : `AL_Operations.S` の例を開く

3. ディレクトリ *AL_Operations* を選択して（開かないでただ選択します）、OK をクリックします。この例が PlatformIO で開きます。
4. *platformio.ini* を開きます。次記の行を編集して、最初のステップで生成された RVfpga シミュレーションバイナリ（*Vrvfpgasim*）へのパスを確立します（図8を参照）。

```
board_debug.verilator.binary = [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/
Lab2/verilatorSIM/Vrvfpgasim
```

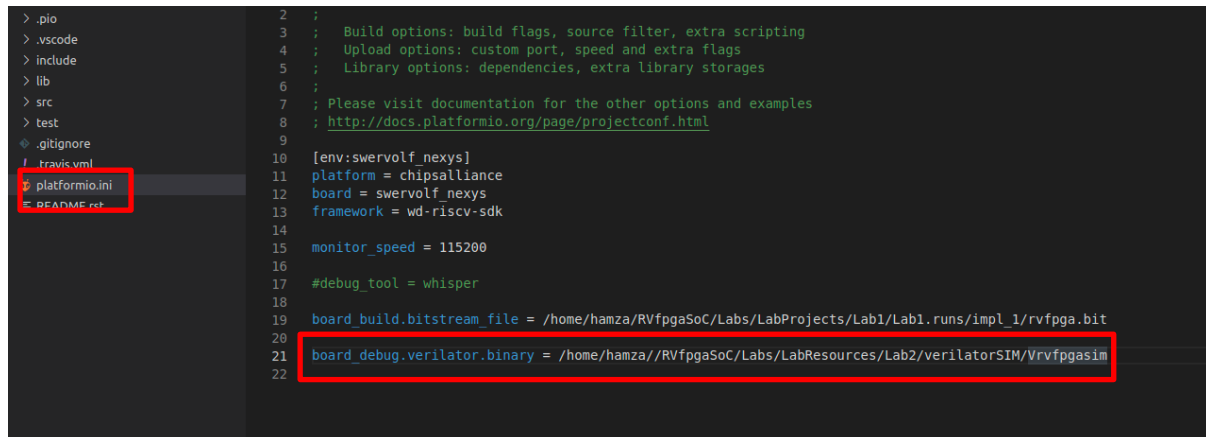



図8 : PlatformIO 初期化ファイル : *platformio.ini*

Windows : Windows では、RVfpga シミュレーション実行可能ファイルは *Vrvfpgasim.exe* と呼ばれます。つまり、

```
board_debug.verilator.binary = [RVfpgaSoCPath]\RVfpgaSoC\Labs\LabResources\Lab2\
verilatorSIM\Vrvfpgasim.exe
```

5. 左メニューリボン  で PlatformIO アイコンをクリックして、Project Tasks → env:swervolf_nexys → Platform を展開して、図9に示されているように Generate Trace をクリックします。

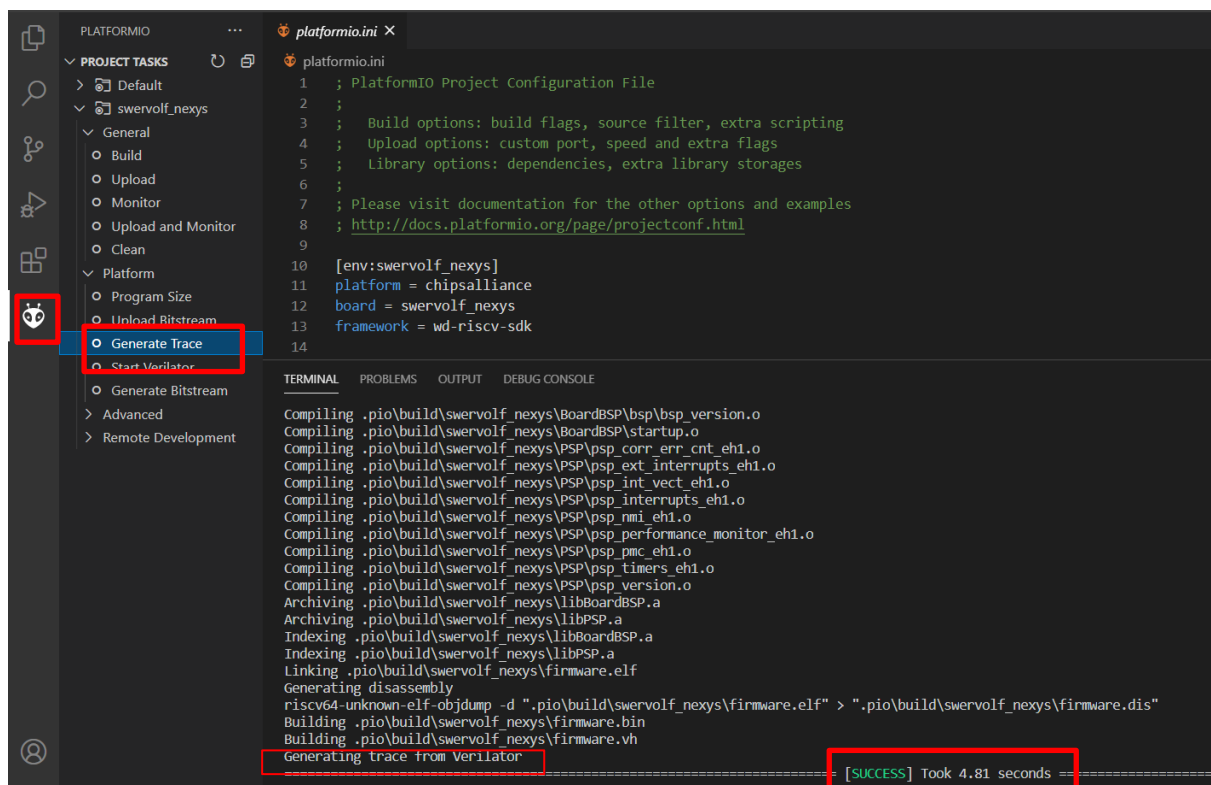



図9 : Verilatorでトレースを生成する

代替として、PlatformIOウィンドウからトレースを生成できます。このためには、新しいターミナルウィンドウを開くために、PlatformIOウィンドウの下部にある  ボタン（PlatformIO: New Terminalボタン）をクリックします。次に次記のコマンドをPlatformIOターミナルに入力（またはコピー）します。pio run --target generate_trace

6. 前のステップの数秒後にファイルtrace.vcdが [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/AL_Operations/.pio/build/swervolf_nexysの内側に生成されているはずであり、これをGTKWaveを使用して開くことができます。Ubuntuターミナルを開いて、次記を入力します。

```
gtkwave
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/AL_Operations/.pio/build/swervolf_nexys/trace.vcd
```

WINDOWS : ダウンロードしたフォルダgtkwave64にbinフォルダの中のgtkwave.exeと呼ばれるアプリケーションが、含まれています。そのアプリケーションをダブルクリックして、GTKWaveを起動します。このアプリケーションの上部でFile - Open New Tabをクリックして、フォルダ [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/AL_Operations/.pio/build/swervolf_nexysに生成されているtrace.vcdファイルを開きます。

ステップ5 : GTKWaveでのシミュレーションを分析する

1. ここで、クロック、命令、レジスタ信号を追加します。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 → rvfpgasim → swervolf → swerv_wrapper_verilog_0 → swerv_eh1_2 → swervに展開し、

モジュールifu（図10に示されているように、強調表示されます）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。

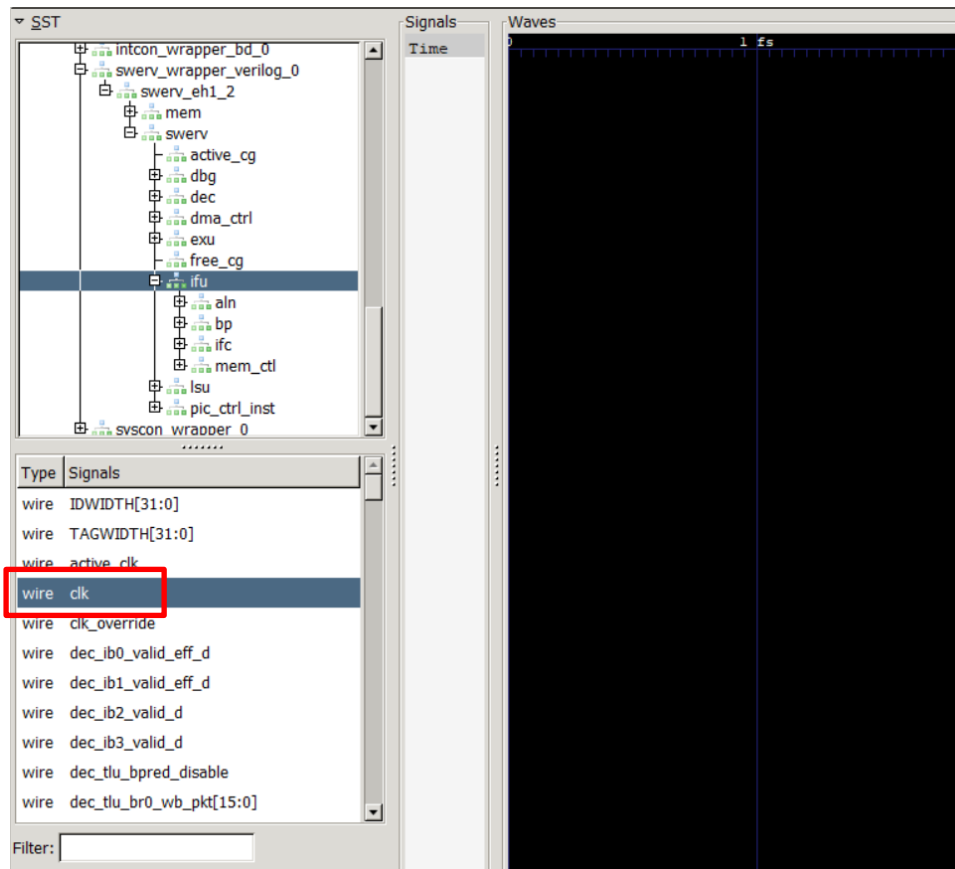


図10：信号clkをグラフに追加する

2. クロック信号の変化を表示できるように、Zoom Fitを実行してからZoom inを数回実行します（図11）。

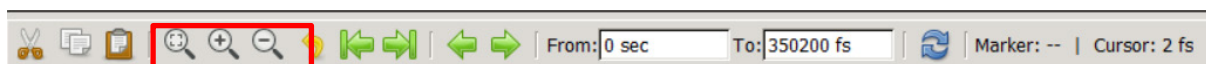


図10：Zoom in

3. スーパースカラRISC-Vコアの2つのウェイのそれぞれを実行する命令を示す信号を、追加します。同じモジュール（ifu）で信号ifu_i0_instr[31:0]およびifu_i1_instr[31:0]（図12）を探して、これらを黒色のWavesペインの中にドラッグします。プレフィックスifuは命令のフェッチユニットを示し、i0はスーパースカラのウェイ0を示し、i1はスーパースカラのウェイ1を示し、instr[31:0]は32ビットの命令を示します。
4. 検索フィルタを使用して信号を素早く見つけることができます（図12を参照）。

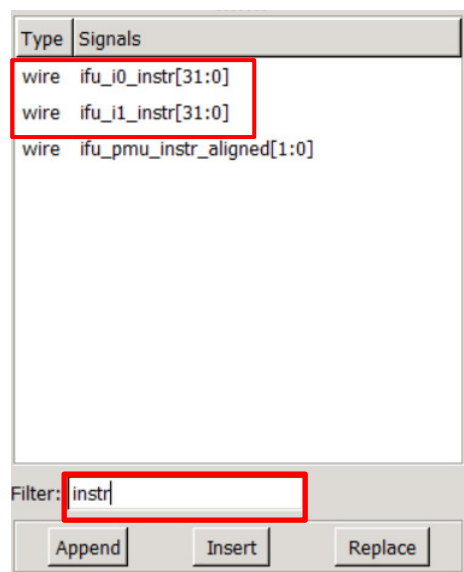


図12 : 信号`ifu_i0_instr[31:0]`および`ifu_i1_instr[31:0]`をタイミング波形に追加する

- レジスタ`t3`（つまりレジスタ番号28、`x28`）の値を保持する信号を追加します。**swerv**の下階層構造を`dec` → `arf` → `gpr_banks(0)` → `gpr(28)`に展開して、モジュール`gprff`（下図に示されているように強調表示されます）をクリックし、信号`dout[31:0]`（これによってレジスタ`x28`の内容が示され、`AL_Operations.S`例で使用する）を選択して、これを黒色のWavesペイン（図13）にドラッグします。

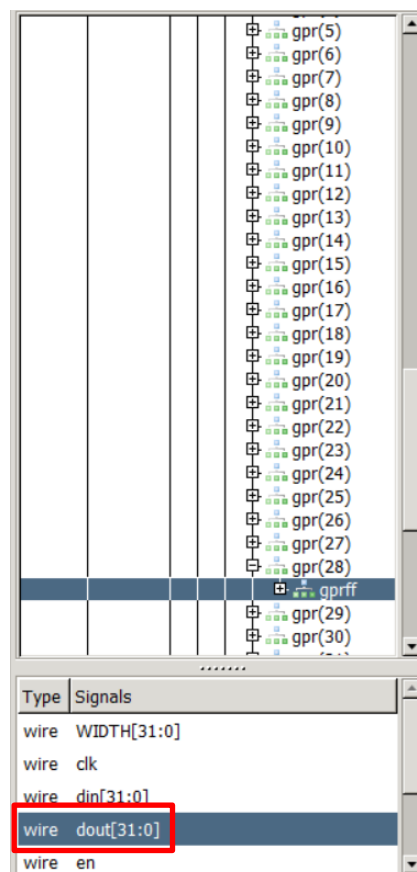


図13 : 信号`dout[31:0]`をグラフに追加する

6. GTKWaveで信号を表示する別の方法は、`.tcl`を使用する方法です。ファイル`gtkwave_signals.tcl`は`[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/`で提供されます。ファイルを開いてこれを分析します。各ラインに、グラフで表示する各信号のパスおよび名前が、表示されます。

```
gtkwave::addSignalsFromList rvfpgasim.clk
gtkwave::addSignalsFromList
rvfpgasim.swervolf.swerv_wrapper_verilog_0.swerv_ehl_2.swerv.ifu.ifu_i0_instr
gtkwave::addSignalsFromList
rvfpgasim.swervolf.swerv_wrapper_verilog_0.swerv_ehl_2.swerv.ifu.ifu_i1_instr
gtkwave::addSignalsFromList
rvfpgasim.swervolf.swerv_wrapper_verilog_0.swerv_ehl_2.swerv.dec.arf.gpr_banks(0).gpr(28).gprff.dout
```

`.tcl`ファイルをGTKWaveで使用するには、単にFile - Read Tcl Script Fileをクリックして`RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/gtkwave+signals.tcl` ファイルを選択するだけです。

図14に、AL_Operations.Sプログラムおよびその等価なマシン言語命令が示されています。

# RISC-V assembly	# comment (t3 = x28)	# machine code
li t3, 0x0	# t3 = 0	# 0x00000E13
REPEAT:		
addi t3, t3, 6	# t3 = t3 + 6	# 0x006E0E13
addi t3, t3, -1	# t3 = t3 - 1	# 0xFFFE0E13
andi t3, t3, 3	# t3 = t3 AND 3	# 0x003E7E13
beq zero, zero, REPEAT	# Repeat the loop	# 0xFE000CE3
nop	# nop	# 0x00000013

図14 : 等価なマシンコードによるAL_Operations.S

それでは、信号の変化をプログラムの実行として表示します。プログラムが実行されると、命令およびt3（レジスタx28）は図15に示されている値になることが、予測されます。

	li t3, 0x0	# t3 = 0	# 0x00000E13
REPEAT:	addi t3, t3, 6	# t3 = 0 + 6 = 6	# 0x006E0E13
	addi t3, t3, -1	# t3 = 5	# 0xFFFE0E13
	andi t3, t3, 3	# t3 = 5 & 3 = 1	# 0x003E7E13
	beq zero, zero, REPEAT	# Repeat the loop	# 0xFE000CE3
	nop	# nop	# 0x00000013
REPEAT:	addi t3, t3, 6	# t3 = 1 + 6 = 7	# 0x006E0E13
	addi t3, t3, -1	# t3 = 7 - 1 = 6	# 0xFFFE0E13
	andi t3, t3, 3	# t3 = 6 & 3 = 2	# 0x003E7E13
	beq zero, zero, REPEAT	# Repeat the loop	# 0xFE000CE3
	...		

図15 : AL_Operations実行中の命令のフローおよびレジスタt3（x28）の値

7. 約10,100 nsにズームインし、ループの1回目および2回目の繰り返しの3つの算術論理命令の実行結果を分析します（図16）。最初の2つの命令（`li t3, 0x0 = 0x00000E13` および `addi t3, t3, 6 = 0x006E0E13`）が、信号 `ifu_i0_instr[31:0]` および `ifu_i1_instr[31:0]` に示されているように、スーパースカラRISC-Vプロセッサのそれぞれに対する方法で、最初にフェッチされます。次の2つの命令（`addi t3, t3, -1 = 0xFFFE0E13` および `andi t3, t3, 3 = 0x003E7E13`）が、次のサイクルでフェッチされます。最後の2つの命令（`beq zero, zero, REPEAT = 0xFE000CE3` および `nop = 0x00000013`）が、次のサイクルでフェッチされます。

SweRVコアの9ステージのパイプライン化されたプロセッサおよび依存性により、命令の影響は、命令がフェッチされてから8サイクル以上後に現れます。1つ目および2つ目の命令がフェッチされてから8サイクル後に、1つ目の命令：`li t3, 0x0` (`0x00000E13`) によって、`x28 (t3)` は0になります（既に0になっていた）。1サイクル後に`x28`は、次の命令`addi t3, t3, 6` (`0x006E0E13`) によって0x6に更新されます。次に`x28`は、次の命令`addi t3, t3, -1` (`0xFFFE0E13`) によって5に更新されます。最後に、`x28`は、次の命令`andi t3, t3, 3` (`0x003E7E13`) によって1に更新されます。次の2つの命令：`beq zero, zero, REPEAT` (`0xFE000CE3`) および`nop` (`0x00000013`) がフェッチされ、分岐が成立してループが繰り返されます。

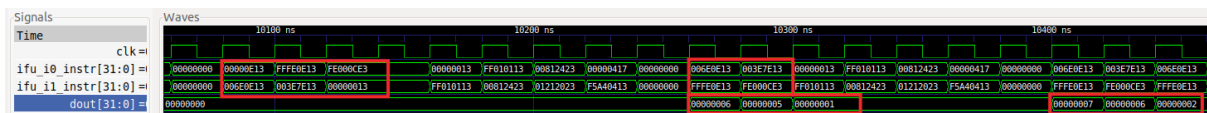


図16：例からの3つの算術論理命令の実行

5. Nexys A7ボードでのプログラムの実行

このセクションでは、PlatformIOを使用するRVfpgaNexysで、FPGAをプログラムします。RVfpgaNexysを使用してFPGAをプログラミングする以下のステップに従います。

以下のステップに従います。

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17を参照）

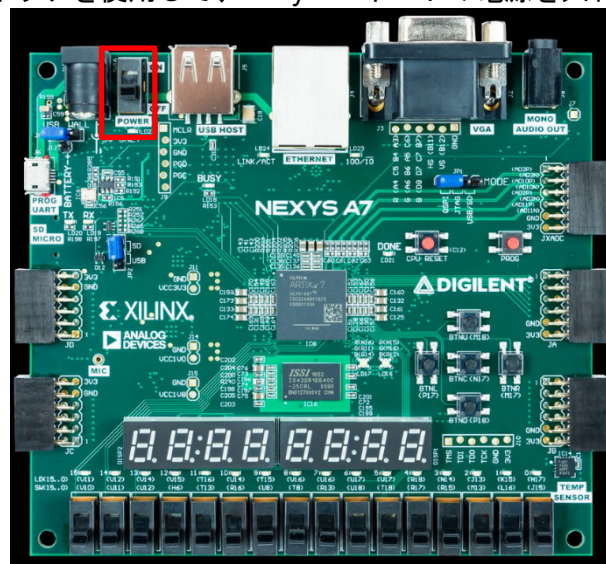


図17：Nexys A7ボードのON/OFFボタン

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18を参照）をクリックして、ディレクトリ`[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab2/examples/`を参照します。

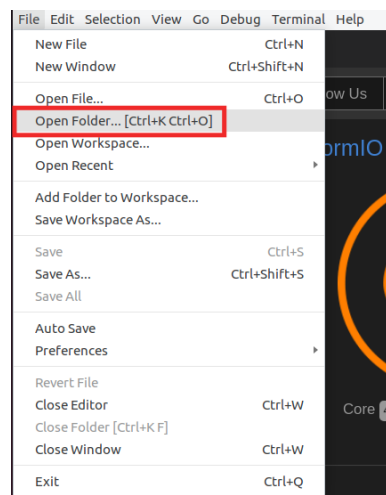


図18：フォルダを開く

ステップ5：ディレクトリ *Blinky* 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、ファイル *platformio.ini* を開きます（図19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19を参照）。

ステップ7：Vivadoブロック設計を使用して作成された「rvfpga.bit」ファイルは、次記のパスにあります。

```
board_build.bitstream_file =
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabProjects/Lab1/Lab1.runs/impl_1/
rvfpga.bit
```

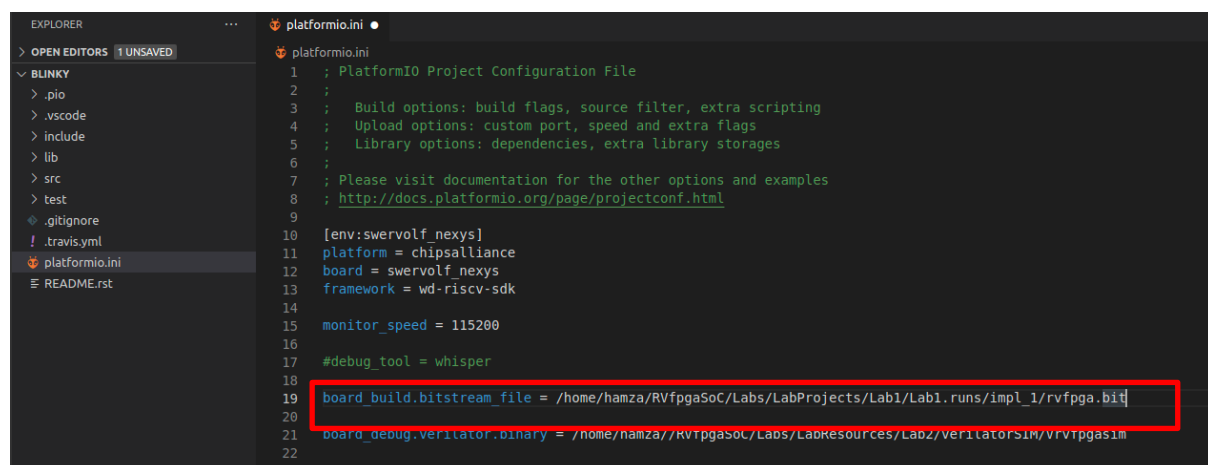


図19：PlatformIO初期化ファイル：platformio.ini

プロジェクト構成ファイル（*platformio.ini*）で利用できる多くのさまざまなコマンドがあり、これらの詳細については、<https://docs.platformio.org/en/latest/projectconf/>を参照してください。



ステップ8：左のメニューリボンでPlatformIOアイコン  をクリックします。（図20を参照）。



図20 : PlatformIOアイコン

Project Tasksウィンドウが空の場合（図21）、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。

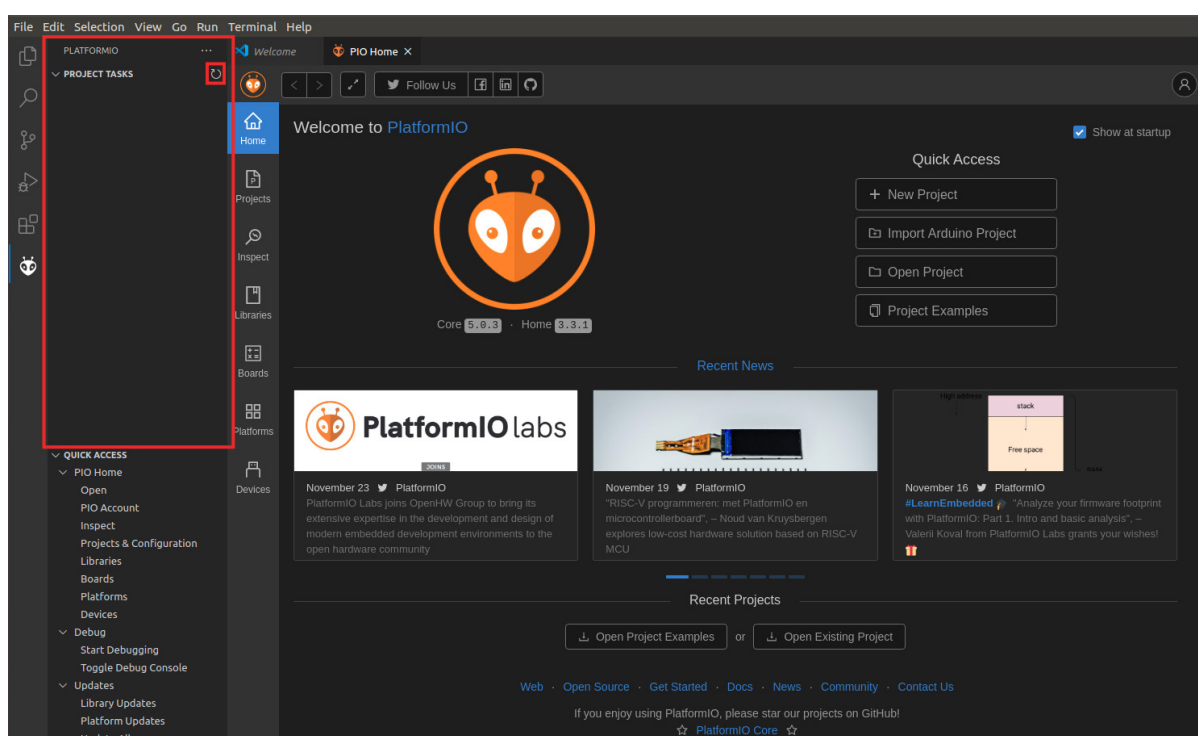


図21 : PROJECT TASKSウィンドウが空 - 更新

Project Tasks → env:swervolf_nexys → Platformを展開して、図22に示されているように、Upload Bitstreamをクリックします。1～2秒後に、FPGAはブロック設計SoCによってプログラムされます（ボードで使用可能な7セグメント表示によって8つのゼロが出力される必要があります）。

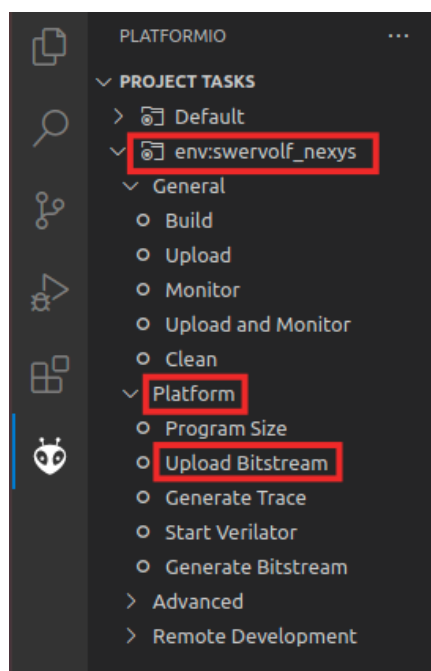


図22 : ビットストリームのアップロード

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。

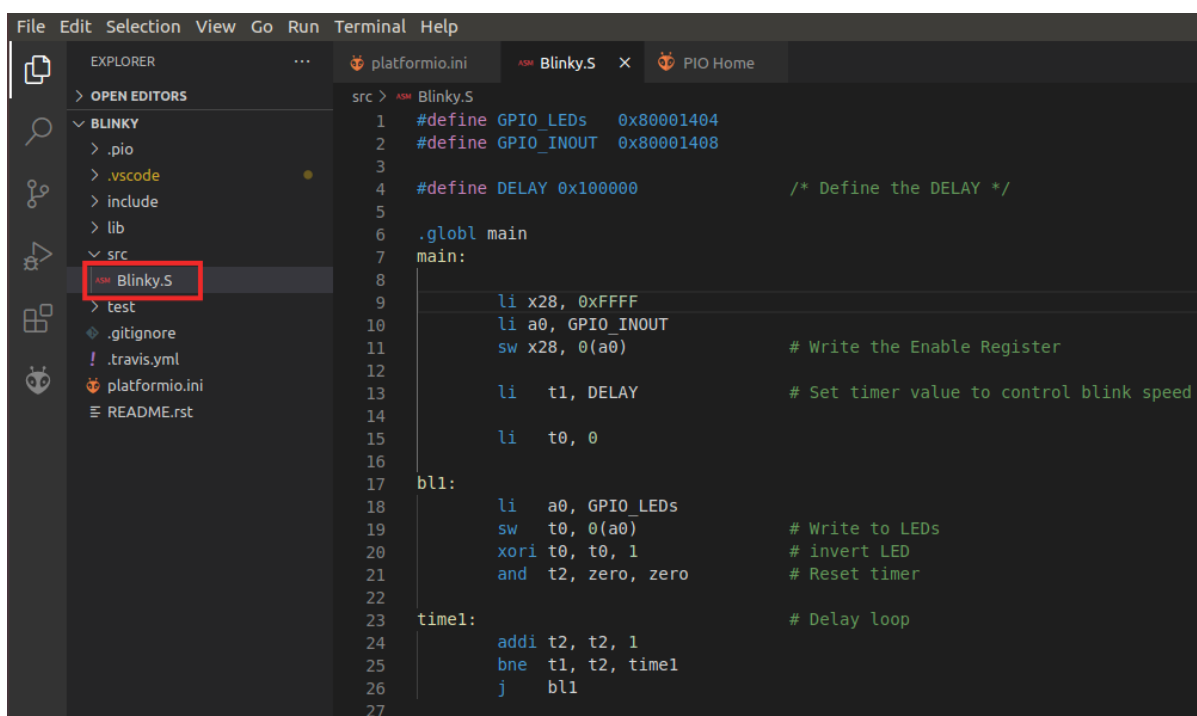

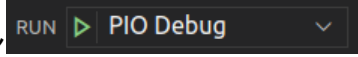



図23 : PlatformIOでのblinky.S

ステップ9 :  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10：ボードで、右端のLEDが点滅を開始します。

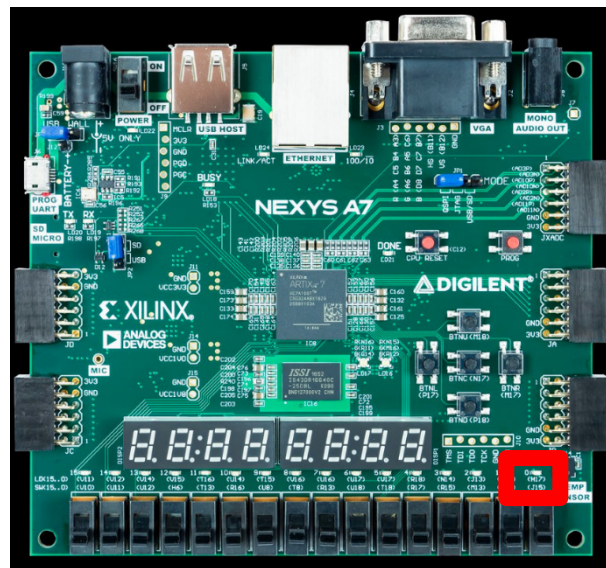


図24：右端のLEDの点滅

ステップ11：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12：行番号18の左をクリックして、ブレークポイントを作成します。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25を参照）。

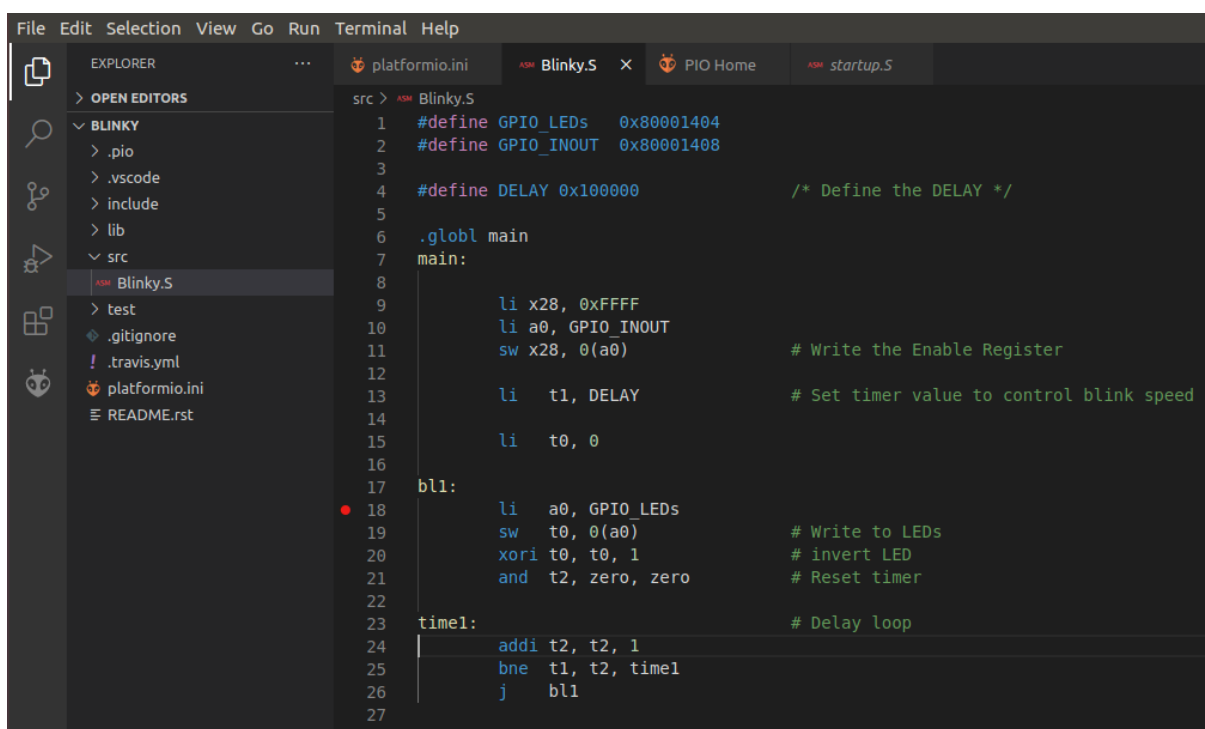



図25：blinky.Sでのブレークポイントの設定

ステップ13：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14：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15：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。

これで、ラボ1で作成しブロック設計モジュールを使用して、RVfpgaSIMおよびRVfpgaNexysで、プログラム例を正常に実行しました。