



THE IMAGINATION UNIVERSITY PROGRAMME

# RVfpga-SoCラボ1

## RVfpga-SoC入門

表1 : RVfpga用語

| 名前                     | 説明  |
|------------------------|---|
| <b>コース</b>             |   |
| <b>RVfpga</b>          | 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。   |
| <b>RVfpga-SoC</b>      | 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。   |
| <b>コアおよびSoC</b>        |   |
| <b>SweRV EH1 コア</b>    | Western Digital開発のオープンソース商用RISC-Vコア（ <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ）。   |
| <b>SweRV EH1 コア複合体</b> | 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（ <a href="https://github.com/chipsalliance/Cores-SweRV">https://github.com/chipsalliance/Cores-SweRV</a> ）。  |
| <b>SweRVolfX</b>       | 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。<br><b>SweRVolf</b> （ <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ）：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。<br><b>SweRVolfX</b> ：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。 |
| <b>RVfpgaNexys</b>     | 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。<br>RVfpgaNexysはSweRVolf Nexys（ <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ）と同じですが、例外として後者はSweRVolfに基づいています。   |
| <b>RVfpgaSim</b>       | シミュレーションを目的とした、テストベンチラップおよびAXIメモリ付きSweRVolfX SoC。<br>RVfpgaSimはSweRVolf Sim（ <a href="https://github.com/chipsalliance/Cores-SweRVolf">https://github.com/chipsalliance/Cores-SweRVolf</a> ）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

### 1. System on a Chip入門

このラボでは、RISC-V System on a Chip (SoC) を構成要素から構築する方法を示します。**SoC**は、電子機器全体やコンピュータシステムを統合する統合回路またはICです。SoCには、コアとすべての周辺機器、およびオペレーティングシステムをロードし、プログラムを実行するために必要なインターフェースが、含まれています。内蔵システム（プロセッサコアを初めとして、コアの周辺に構築されたSoC、および最終的にはシステムとボードインターフェース）の一般的な階層機構が、図1に示されています。

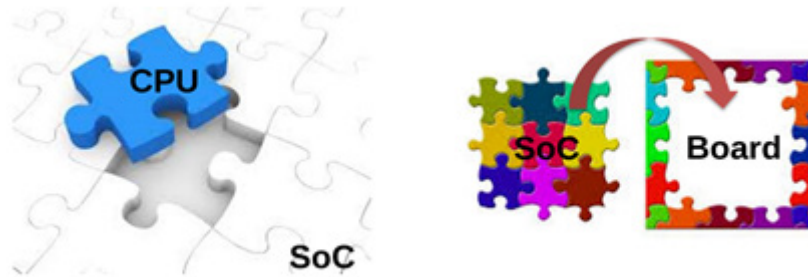


図1：一般的な内蔵システム

SoCの設計プロセスは、FPGAでの試作品製造から始まります。ここでは、SoCの対象をFPGAにすることに焦点を合わせます。

使用しようとしているRISC-V CPUはWestern Digitalの**SweRV EH1コア複合体**であり、このラボで設計するSoCは**SweRVolfX**のサブセットで、これらの対象は**Nexys A7-100T**ボードです。図2に、さまざまな構成要素およびそれらを組み合わせる方法が、示されています。

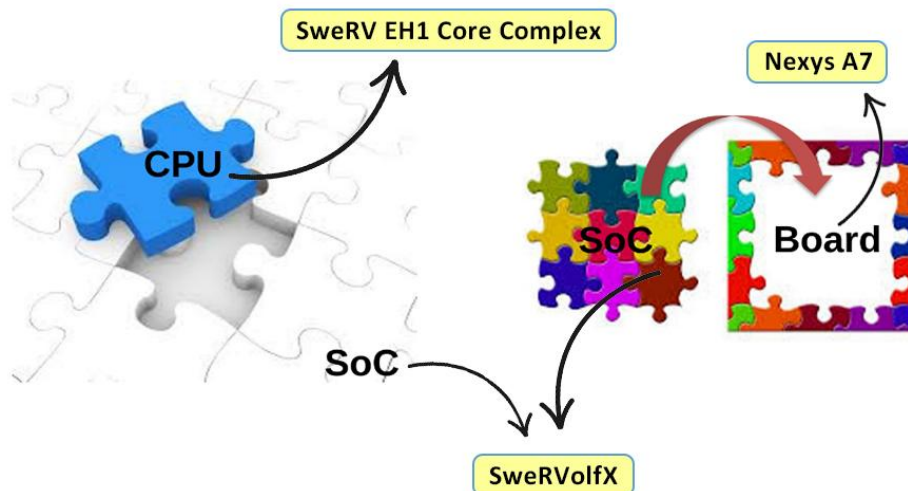


図2：RVfpgaシステムベースの内蔵システム

### 2. SweRVolfXシステムおよびRVfpgaシステム入門

このラボを開始する前に、受講者はRVfpgaコースの入門ガイドを検討して、RVfpgaシステム全体を理解することを、強くお勧めします。以下は、RVfpgaコースで紹介されているRVfpgaシステムの簡潔な説明です。

表1に、SweRV EH1コアからRVfpgaNexysおよびRVfpgaSimまでのRVfpgaシステムの階層構造が示されています。**SweRVolfX**と呼ばれ図3に示されているRVfpgaシステムで使用するSystem on a Chip (SoC) は、**SweRVolf**バージョン0.7.3

(<https://github.com/chipsalliance/Cores-SweRVolf/releases/tag/v0.7.3>) に基づいており、これは**SweRV EH1コア複合体**の上に構築されます。SweRV EH1コア複合体に加えて、SweRVolf SoCにはブートROM、UART、システムコントローラ、SPIコントローラも含まれています。SweRV EH1コアではAXIバスが使用され、周辺機器ではWishboneバスが使用され、SoCにはAXI-Wishboneブリッジもあります。

RVfpgaシステムでは、SweRVolf SoCは、他のSPIコントローラ (SPI2)、GPIO (汎用入力/出力) コントローラ、PTC (PWM/タイマ/カウンタ) モジュールなどのもう少しの機能で拡張されます。(図3にこれらの新しい周辺機器が赤色で示されています)。このSystem on a Chipは**SweRVolfX** (XはeXtendedを意味する) と呼ばれます。

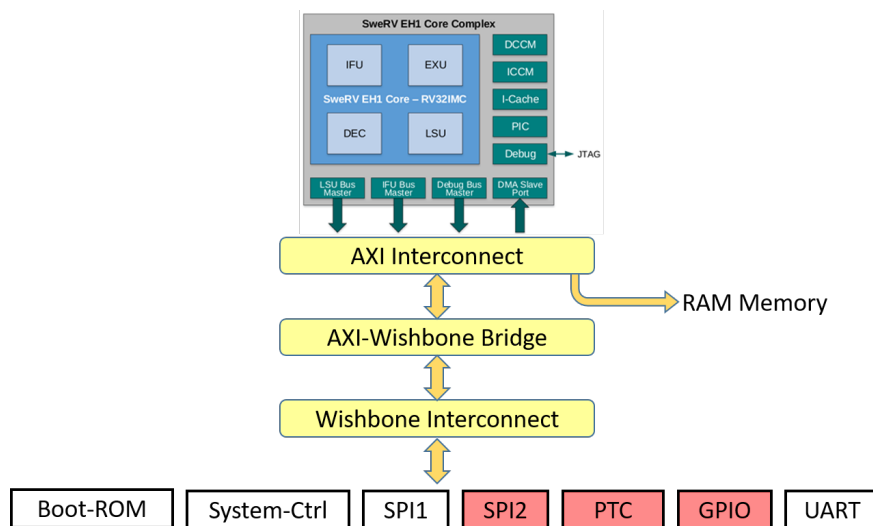


図3 : SweRVolfX

表4に、SweRV EH1コアにWishbone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                    |
|------------|-------------------------|
| ブートROM     | 0x80000000 - 0x80000FFF |
| システムコントローラ | 0x80001000 - 0x8000103F |
| SPI1*      | 0x80001040 - 0x8000107F |
| SPI2*      | 0x80001100 - 0x8000113F |
| タイマ*       | 0x80001200 - 0x8000123F |
| GPIO*      | 0x80001400 - 0x8000143F |
| UART       | 0x80002000 - 0x80002FFF |

\* SweRVolfXで追加された周辺機器

### 3. RVfpga-SoC入門

RVfpgaでは、SweRVolfXは、SweRVolfXが作成された方法と関連する詳細なしに、導入されました。RVfpga-SoCコースでは、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SweRVolfX SoCのサブセットを構築する方法が示されています。

このラボは、CPU（SweRV EH1コア複合体として）から始めてSoCに組み込む方法を示す、ステップバイステップのガイドです。Vivadoブロック設計ツールを使用します。Vivadoのブロック設計ツールにより、構成部品の配線がグラフィックスで容易になり、プロセスの理解および可視化が簡単になります。視覚的アプローチにより、各モジュールを他のモジュールと接続してSoCを形成する方法も示されます。

モジュールは、以下の3つの主要なブロックまたはカテゴリに分類できます。

1. CPU（SweRV EH1コア複合体）
2. インターコネクト（AXIインターコネクト、AXI2WB、WBインターコネクト）
3. 周辺機器（ブートROM、GPIOコントローラ、システムコントローラ）

SweRVolfXには多くのさまざまなモジュールがあり、一部は必要最小限のRISC-V SoCには必要ではありません。このため、余分なモジュールが削減されてラボが簡素化され、CPUを生き生きさせるために必要な必要最小限の機能に焦点が合わされています。含めないモジュール（UART、PTC、SPI1、SPI2）が図4に示されています。

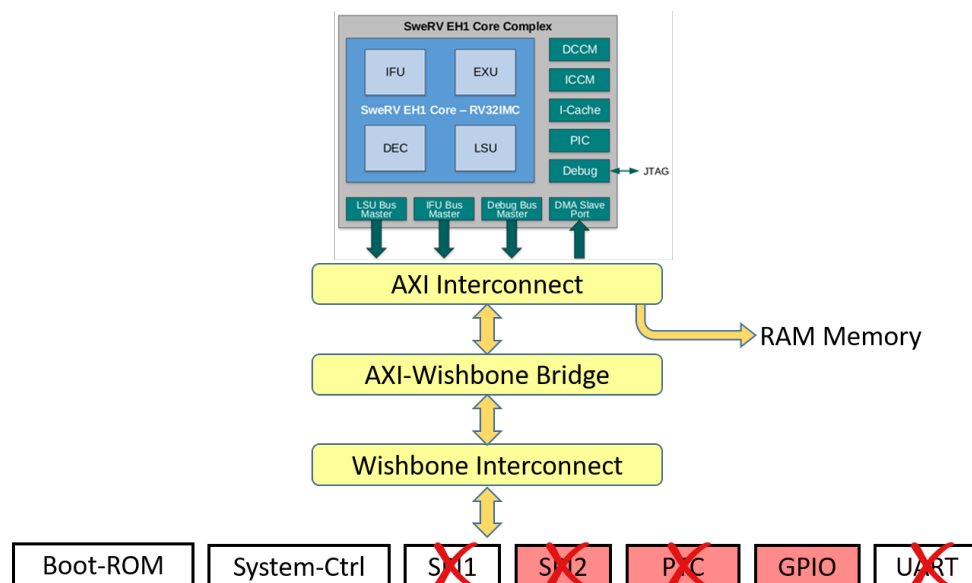


図4：SweRVolfXのサブセット

図5に、実装するSoCの高レベルのブロック図が示されています。

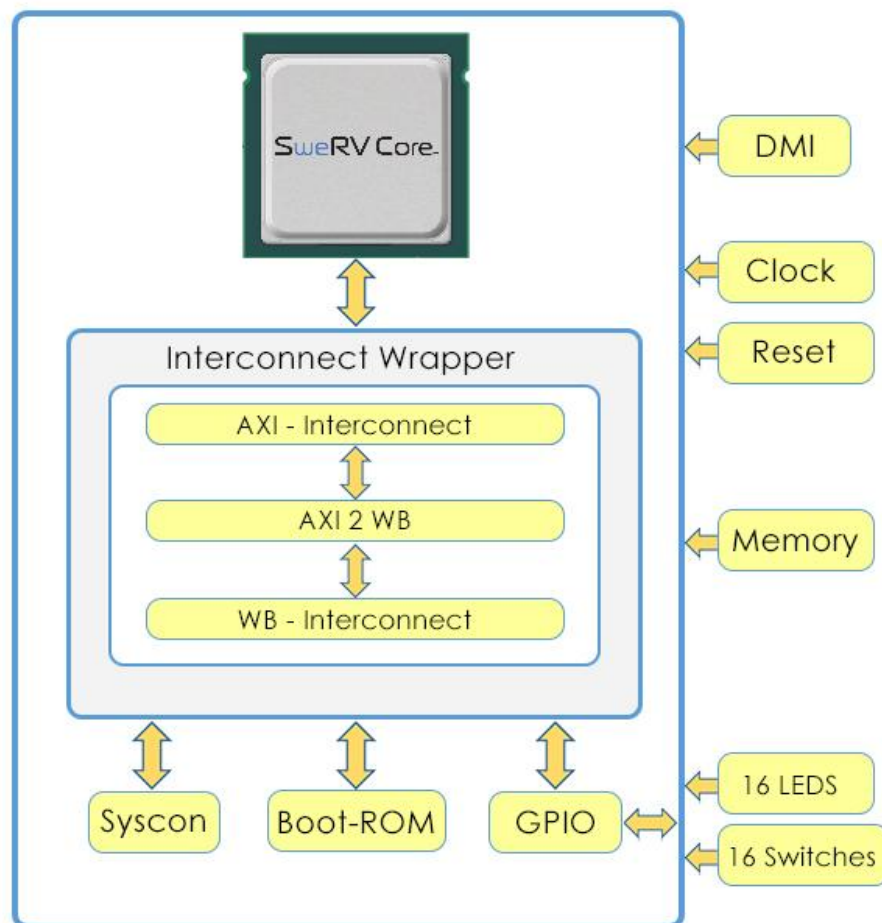


図5 : ラボ1 SoCの高レベルのブロック図

学習および理解を容易にするために、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、およびAXIからWishboneへのブリッジ）を構成する一部の構成要素は1つのインターコネクトラッパモジュールに包含されています。

CPUおよびCPUの内部に焦点を合わせたラボについては、RVfpgaコースを参照してください。RVfpga（RISC-V FPGA）コースは、商用RISC-VプロセッサとSoCをフィールドプログラミング可能なゲートアレイ（FPGA）への対象とする、命令、ツール、ラボが含まれているパッケージであり、これを使用および展開して、コンピュータアーキテクチャ、デジタル設計、組み込みシステム、プログラミングについて学習します。RVfpgaの詳細については、<https://university.imgtec.com/rvfpga/>にアクセスしてください。

## 2. 要件

このラボを完了するには、以下のソフトウェアがインストールされている必要があります。

- Vivado 2019.2 Web Pack （インストールガイド（No.04ページ）を参照）
- Digilentボードファイル （インストールガイド（No.05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てください。ImaginationのUniversity ProgrammeからダウンロードしたRVfpgaSoCフォルダをお使いのマシンにコピーしたことを、確認してください。

### 3. Vivadoプロジェクトの作成

XilinxのVivado Design Suiteを使用して、システムを定義するVerilogファイルであるRTLによってSweRVofFXサブセットを構築します。以下に詳説されているステップに従って、Vivadoプロジェクトを作成します。

#### ステップ1：Vivadoを開く

使用するマシンにVivadoがインストールされていない場合は、今すぐ、RVfpga-SoCインストールガイドに説明されているようにインストールしてください。必ず、ボードファイルもインストールします。

Vivadoを実行します（**Linux**で、ターミナルを開き、**vivado**を入力し、**Windows**でStartメニューからVivadoを開きます）。VivadoのWelcome画面が開きます。Create Projectをクリックします（図6を参照）。

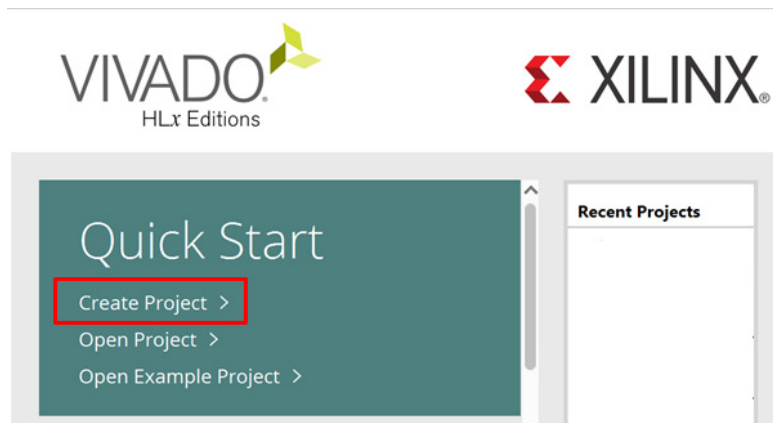


図6：VivadoのWelcome画面：プロジェクトの作成

#### ステップ2：RTLプロジェクトを新規作成する

「Create a New Vivado Project」ウィザードが開きます（図7を参照）。Nextをクリックします。

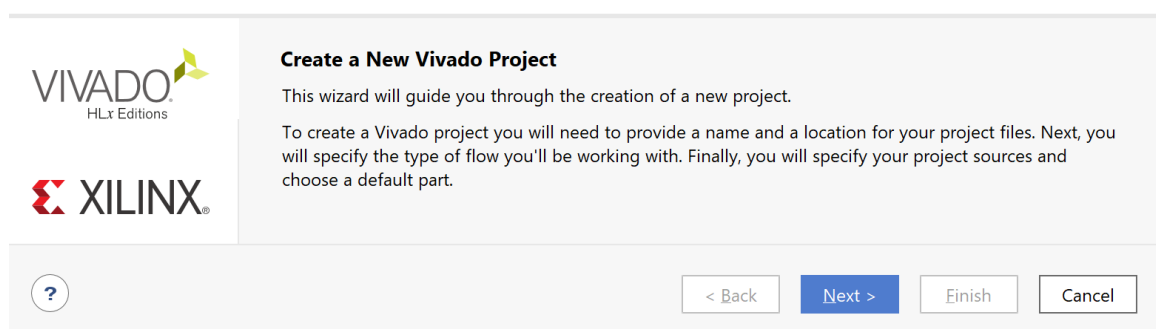


図7：「Create a New Vivado Project」ウィザード

プロジェクト名「**Lab1**」をスペース文字なしで入力します。次にNextをクリックします（図8を参照）。

次記のプロジェクトの場所のパスを選択します：  
[RVfpgaSoCPath] /RVfpgaSoC/Labs/LabProjects/Lab1

「**LabProjects**」フォルダには既に「**Lab1**」と呼ばれるフォルダがあるため、Create project subdirectoryチェックボックスのチェックマークを外します。

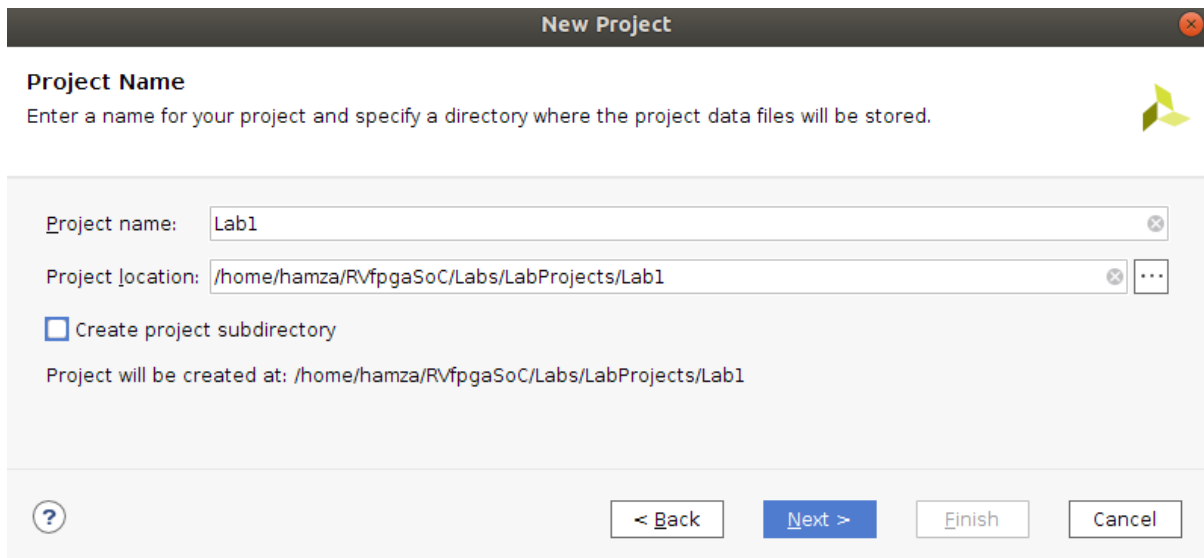


図8：プロジェクト名

プロジェクトタイプにRTL Projectを選択し、Nextをクリックします（図9を参照）。

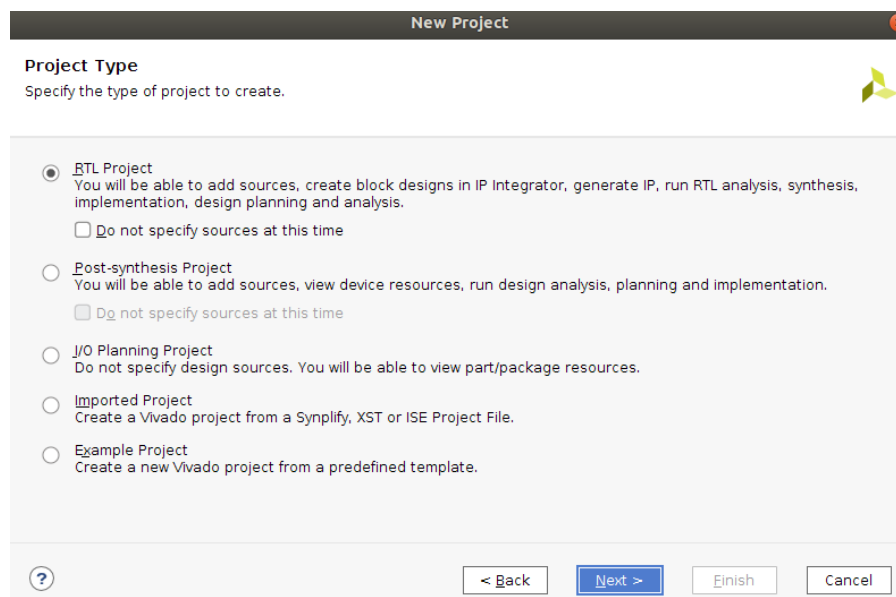


図9：RTLプロジェクト

### ステップ3：RTLソースファイルおよび制約ファイルを追加する

Add Sourcesウィンドウで「**Add Directories**」をクリックします（図10を参照）。



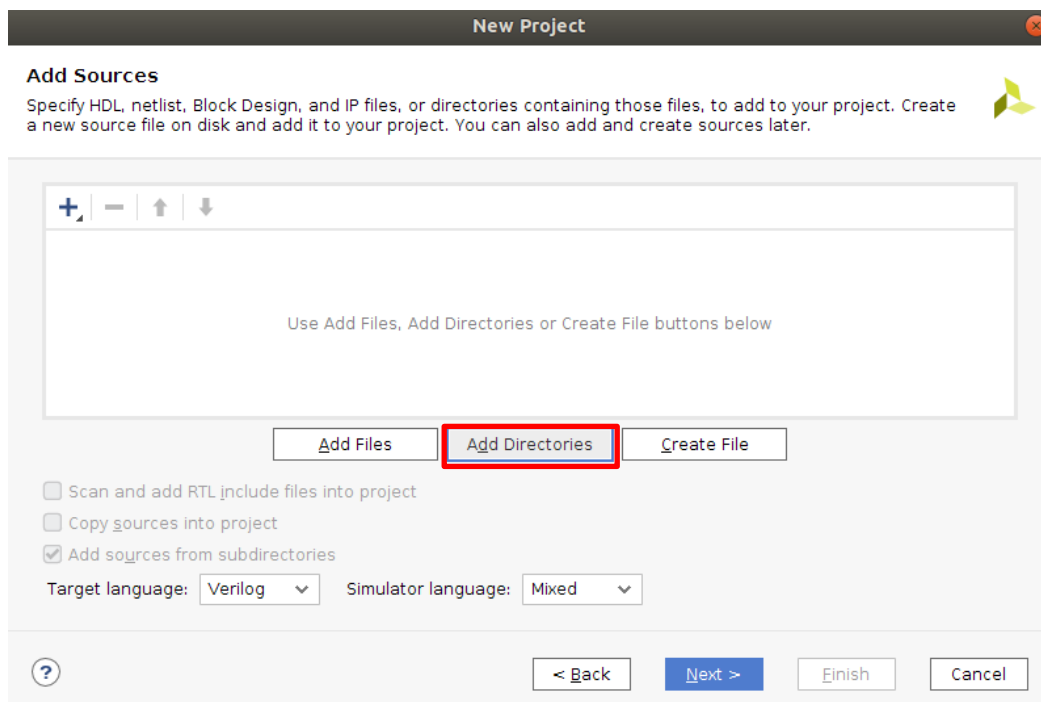


図10 : ソースディレクトリを追加

パス [RVfpgaSoCPath] /RVfpgaSoC/Labs/LabResources/Lab1/src で「src」ディレクトリ選択します（図11を参照）。

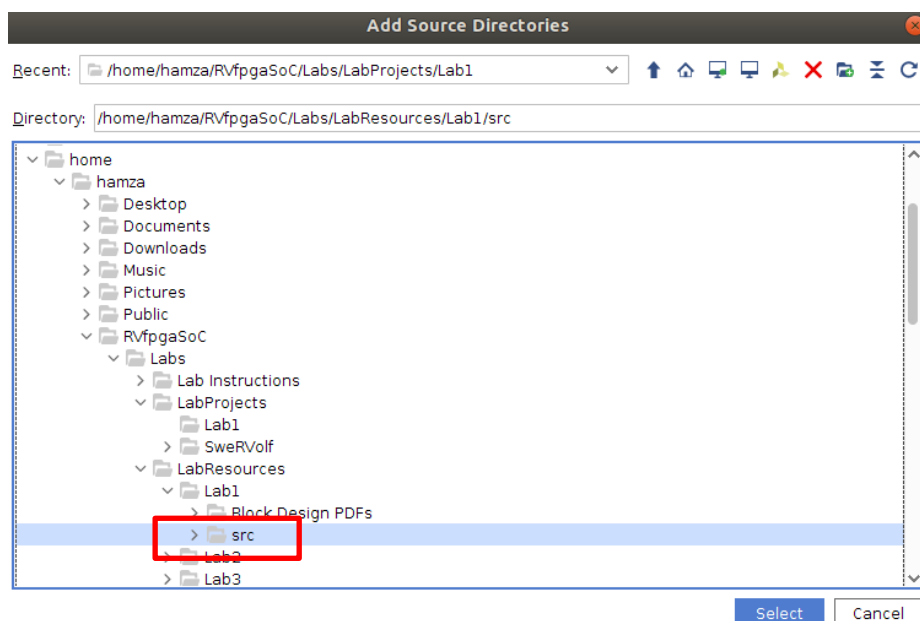


図11 : 「src」ディレクトリを選択

Selectをクリックします。  
次に「Add Files」ボタンをクリックします。



図12 : ファイルを追加

ファイルタイプに「All Files」を選択します。ここで、追加したばかりのsrcディレクトリ内のLiteDRAMディレクトリに移動します。

- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/mem\_1.init
- [RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/LiteDRAM/litedram\_core.init

両方の「.init」ファイルを選択し、OKをクリックしてこの両方を追加します（図13を参照）。

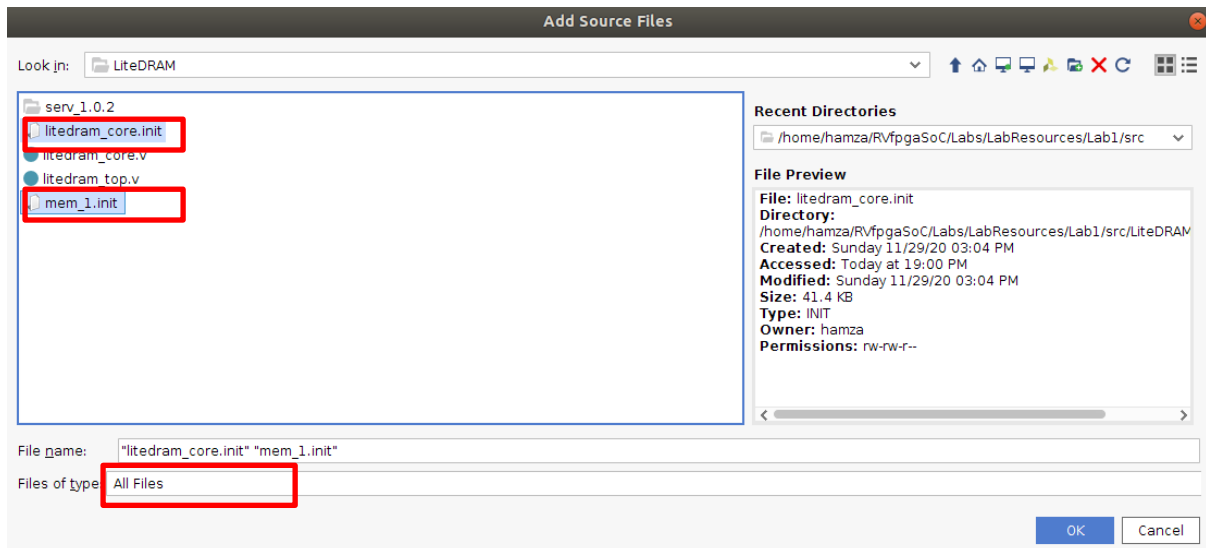


図13 : LiteDramソースファイルを追加

3つのチェックボックスすべてにチェックマークがついていることを確認します（図14を参照）。

「Next」をクリックして、次のステップに進みます。

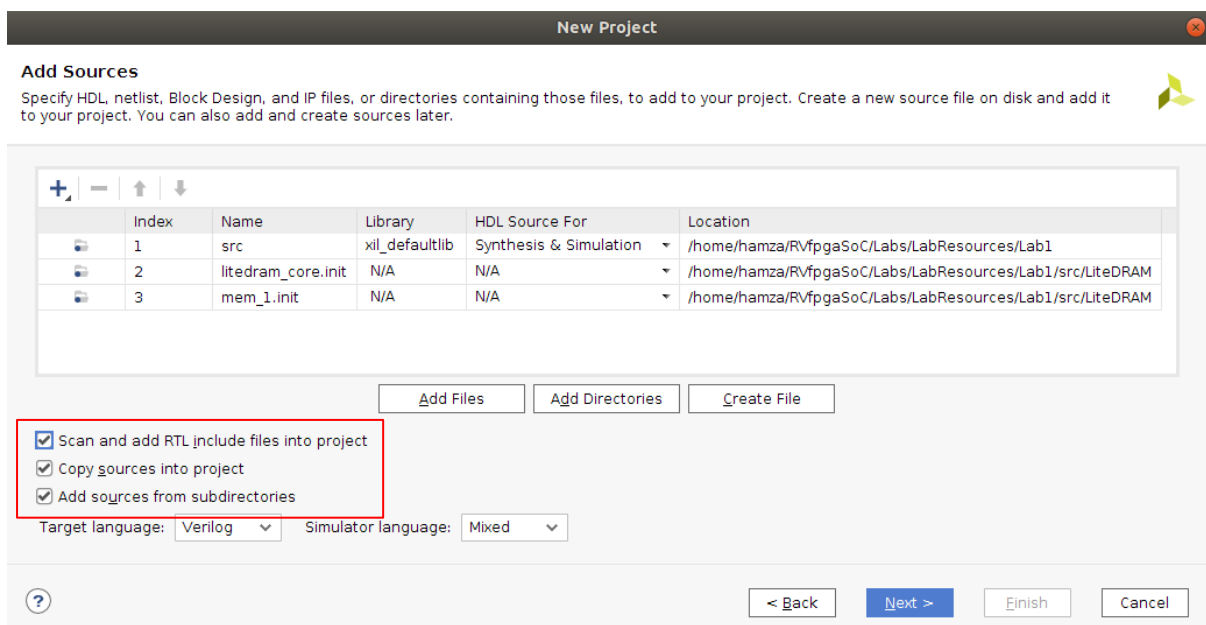


図14 : ソースを追加

これで、システムへの制約を追加します。3つのファイルにより、信号名がボードのピンに割り当てられます。例えば、Nexys A7 FPGAボードのLEDはボードのFPGAピンに、PCBトレースを介して接続されます。Vivadoはこれを知って、RTLの適切な信号名を適切なFPGAピンに割り当てる必要があります。例えば、Xilinx 設計制約ファイルである `[RVfpgaSoCPath]/RVfpgaSoC/src/rvfpga.xdc` ファイルの以下の行に、FPGAピンH17によって最も重要でないLED (`o_led[0]`) に割り当てられ、LVCMOS 3.3V信号送信が使用されることが、示されます。

```
set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports { o_led[0] }]
```

信号名 `o_led` は、Nexys A7ボードのLEDを駆動するためにRVfpgaのVerilogコードで使用される名前であることに、注意してください。

Add Constraintsウィンドウで、「Add Files」をクリックし、以下の2つのファイルを選択します（図15を参照）。

```
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/rvfpga.xdc
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/litedram.xdc
```

次に、**Next**をクリックします。

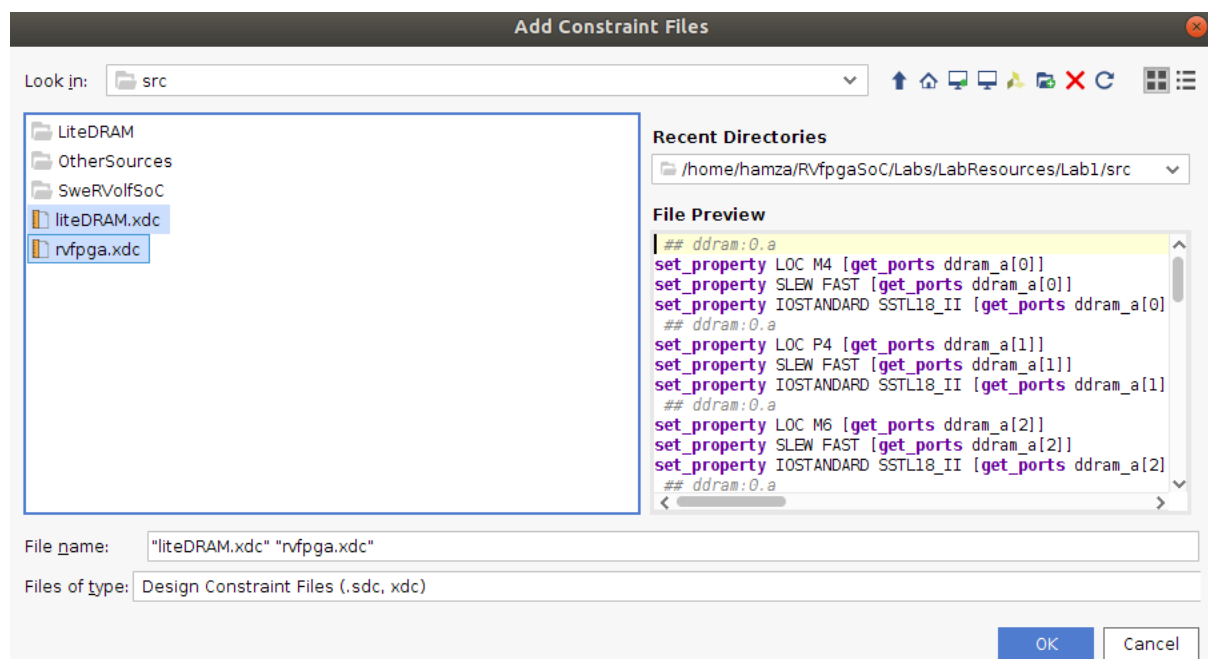


図15：制約を追加

#### ステップ4：Nexys A7を対象のボードとして選択する

Default Partウィンドウで、Boardsをクリックし、Nexys A7-100Tを選択します（図16を参照）。検索ボックスを使用して結果を絞り込むことができます。実際の対象のFPGAの名前が部品列：xc7a100tcs9324-1にリストされていることにも気が付くでしょう。これにより、これが100k等価ゲート、CSG（チップスケールグリッド）パッケージ、324ピン付きXilinx Artix-7 FPGAであることが示されます。

Nextをクリックします。

**Default Part**  
Choose a default Xilinx part or board for your project.

Parts: **Boards**

[Reset All Filters](#) Update Board Repositories

Vendor: All Name: All Board Rev: Latest

Search:  (5 matches)

| Display Name         | Preview | Vendor          | File Version | Part              | I/C |
|----------------------|---------|-----------------|--------------|-------------------|-----|
| <b>Nexys A7-100T</b> |         | digilentinc.com | 1.0          | xc7a100tcsg324-1  | 32  |
| Nexys A7-50T         |         | digilentinc.com | 1.0          | xc7a50ticsg324-1L | 32  |
| Nexys4               |         | digilentinc.com | 1.1          | xc7a100tcsg324-1  | 32  |
| Nexys4 DDR           |         | digilentinc.com | 1.1          | xc7a100tcsg324-1  | 32  |

? < Back Next > Finish Cancel

**図16 : ターゲットボードの選択 : Nexys A7-100T**

New Project Summaryウィンドウで、**Finish**をクリックします（図17を参照）。

**VIVADO**  
HLx Editions

**XILINX**

**New Project Summary**

- A new RTL project named 'Lab1' will be created.
- 208 source files will be added.
- 2 constraints files will be added.
- The default part and product family for the new project:  
 Default Board: Nexys A7-100T  
 Default Part: xc7a100tcsg324-1  
 Product: Artix-7  
 Family: Artix-7  
 Package: csg324  
 Speed Grade: -1

To create the project, click Finish

? < Back Next > Finish Cancel

**図17 : New Project Summaryウィンドウ**

プロジェクトのセットアップが完了すると、ファイルが存在してこれらには構文エラーが含まれている（これは次のステップで修正されます）ことが示されることに、注意してください。

### ステップ5 : rvfpgaを最上位モジュールとして設定する

プロジェクトが初期化されます。RVfpgaモジュールを最上位モジュールとして設定します。Sourcesペインで、Design Sourcesで下方にスクロールし、rvfpgaモジュールを右クリックして、Set as Topを選択します（図18を参照）。示されているように、rvfpgaモジュールの名前を検索ボックスに入力することによって、これを検索することもできます。これによってrvfpgaが階層構造の最高レベルのモジュールおよび合成する対象として設定され、FPGAに実装されます。RVfpgaを最上位モジュールとして設定すると、階層構造が更新されます。

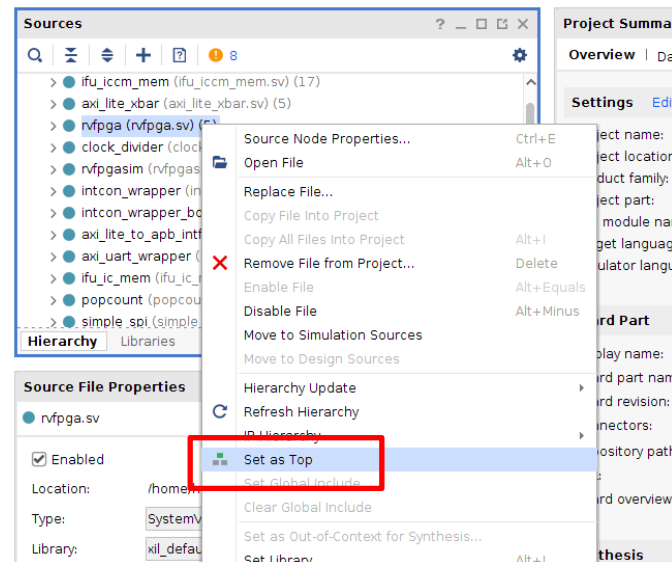


図18 : rvfpgaを最上位モジュールとして設定

### ステップ6 : Verilogヘッダーファイルをグローバルインクルードファイルとして設定する

Design Sourcesの下のSourcesペインの中のみで、Non-modulesファイルグループを展開して、common\_defines.vhをクリックします。すると、ファイルのプロパティが、Sourcesペインのすぐ下のSource File Propertiesペインで開きます。Global Includeをクリックして、そのボックスにチェックを入れます（図19を参照）。階層構造が更新され、そのファイルがDesign Sources/Global Includeに含まれます。

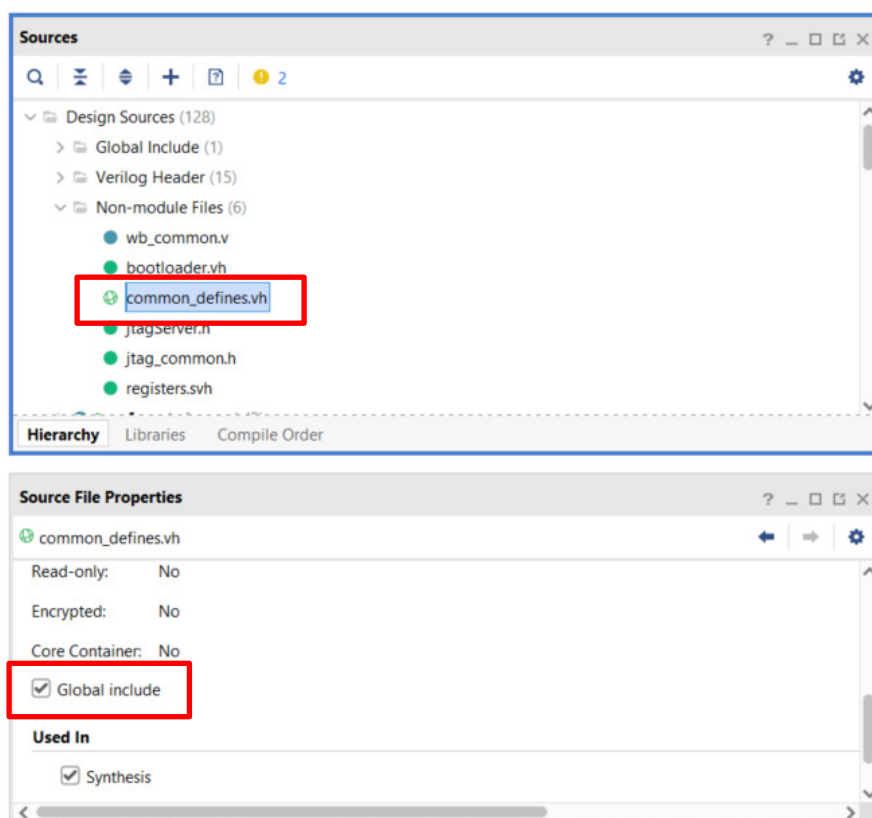


図19 : common\_defines.vhをグローバルインクルードファイルとして設定

同様に、「assign.svh」、「registers.svh」、「typedef.svh」SystemVerilogHeaderファイルをグローバルインクルードファイルとして設定します（図20を参照）。

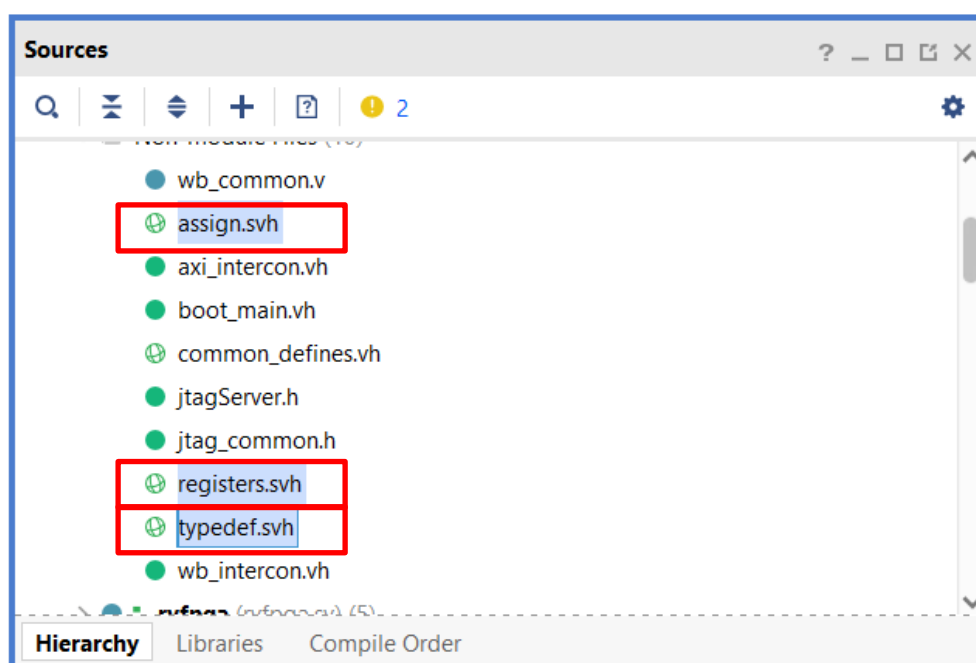


図20 : 「.svh」ファイルをグローバルインクルードファイルとして設定

「unknown」ファイルグループを展開して、「litedram\_core.init」をクリックします。Source File PropertiesパネルのGeneralボタンの横のPropertiesボタンをクリックします。「IS\_Global\_INCLUDE」をクリックして、そのボックスにチェックを入れます（図21を参照）。

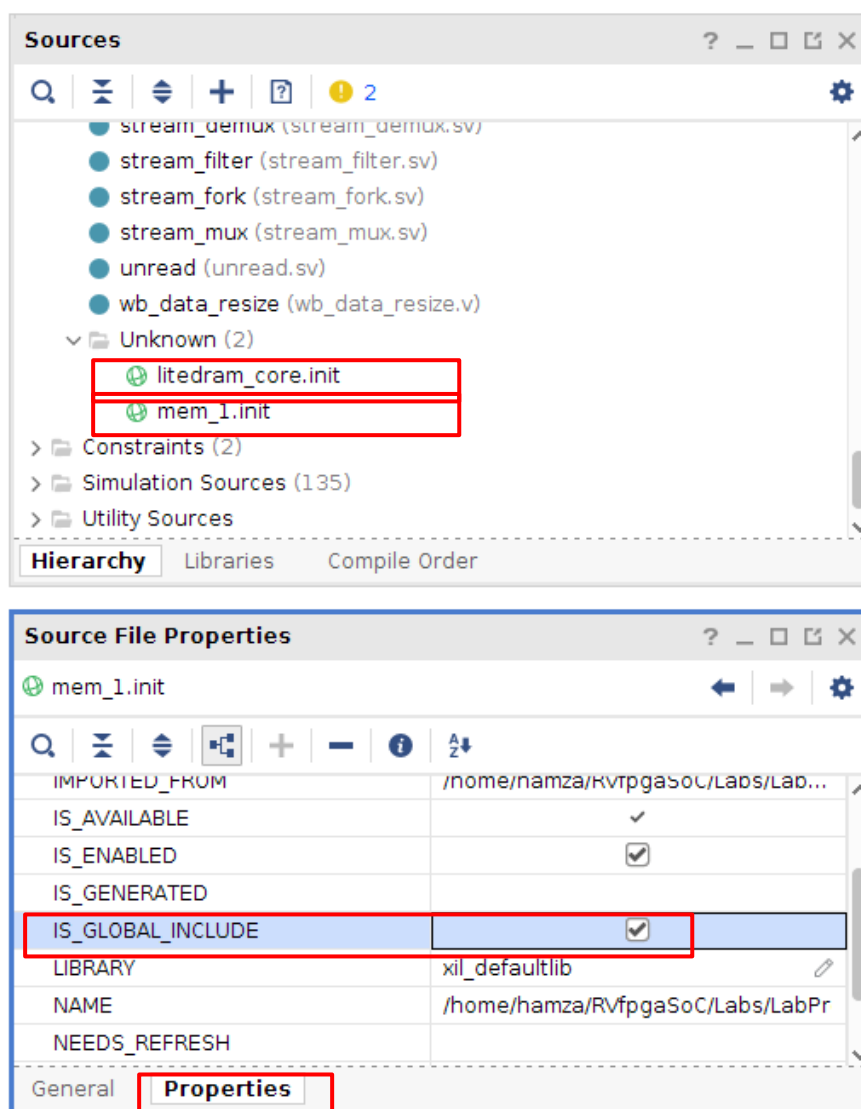


図21 : litedram\_core.initをグローバルインクルードファイルとして設定

「mem\_1.init」ファイルに、「litedram\_core.init」ファイルに実行したのと同様に実行し、このファイルもグローバルインクルードファイルとして設定します。

### ステップ7 : boot\_main.memをプロジェクトに追加する

Flow Navigatorペインで、Add Sourcesをクリックし、デフォルトのオプション（「Add or create design sources」）をそのままにして、Add Filesをクリックします（図22を参照）。[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/src/SweRVolfSoC/BootROM/sw に移動し、boot\_main.memを選択します（図22に示されているように）。階層構造が更新され、設計ソース/メモリファイルにそのファイルが含まれます。



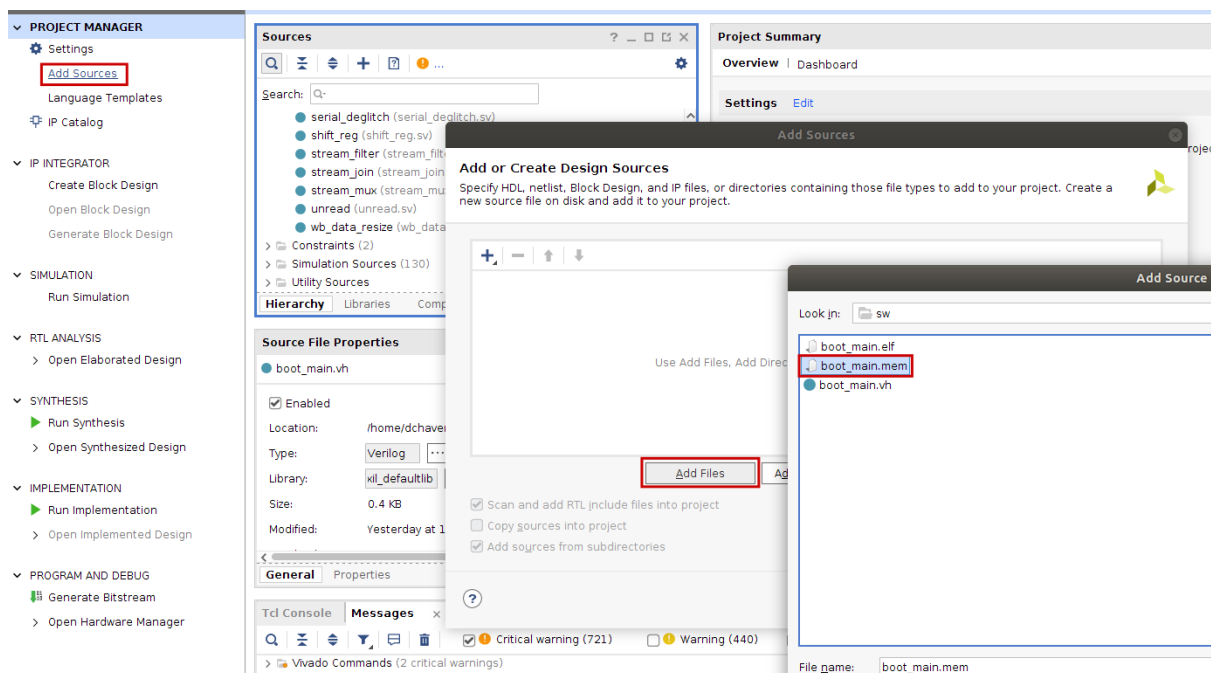


図22：メモリファイルboot\_main.memを追加

設計ソースファイルが追加され、先へ進めてブロック設計を作成できます。

## 4. ブロック設計の作成

Vivadoのブロック設計機能を使用して必要なモジュールを追加し、SweRVolFXサブセットを作成して、モジュールを相互に配線します。

### ステップ1：Create Block Designをクリックする

IPインテグレータの見出しの下でCreate Block Designをクリックして、フローナビゲータでブロック設計を新規作成します（図23を参照）。

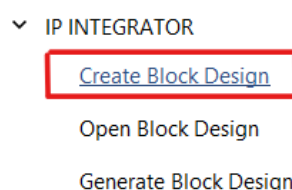


図23：ブロック設計の作成

### ステップ2：ブロック設計の名前を選択する

設計名に「BD」を選択して、後日のラボでの名前付けでの競合を防止します（図24を参照）。

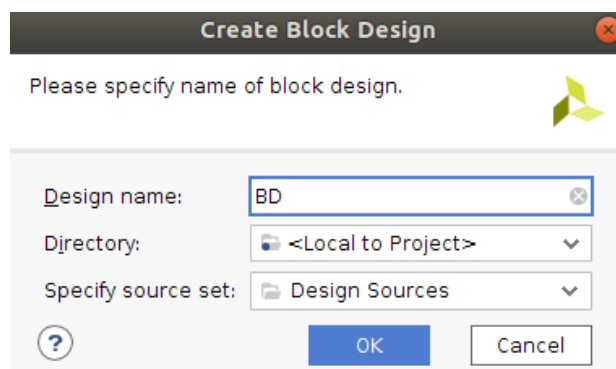


図24 : ブロック設計の名前とディレクトリを選択

ブランクブロック設計のDiagramパネルが表示されます（図25を参照）。

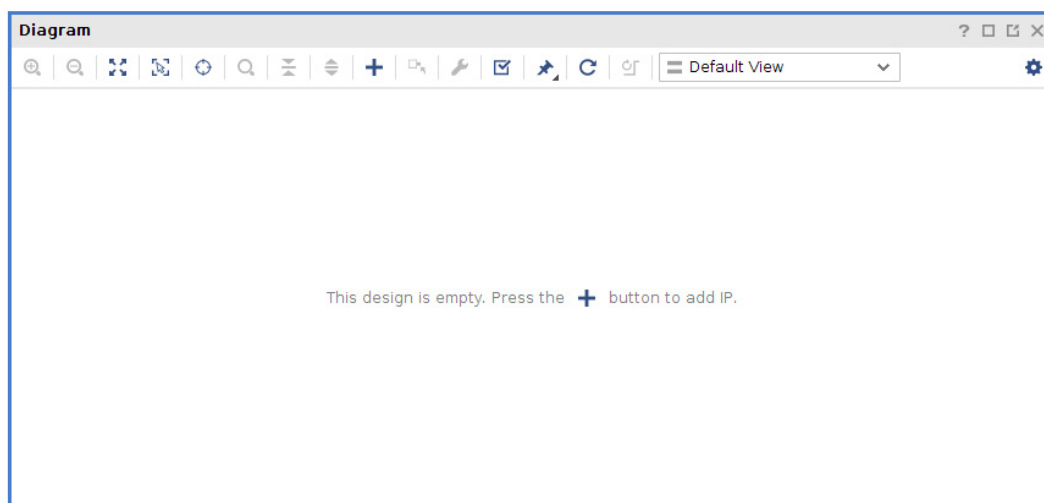


図25 : ブランクブロック設計

### ステップ3 : モジュールをブロック設計に追加する

これで、ブロック設計へのモジュールの追加を開始できます。これをするには、ブランクブロック設計を右クリックし、「Add Module」オプションを選択します（図26を参照）。

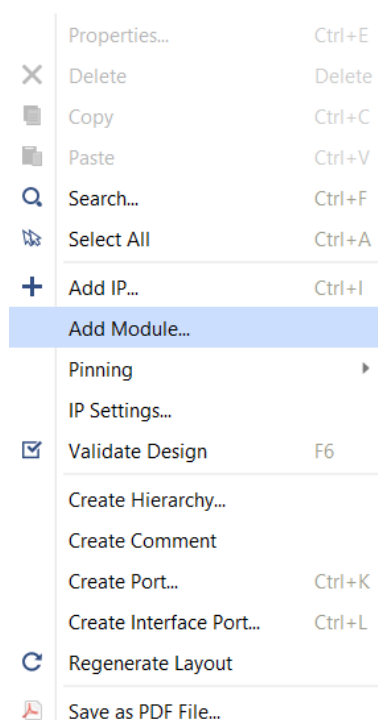


図26 : モジュールを追加

ダイアログボックスが表示され、下方へスクロールするか、追加する必要なモジュールの名前を検索ボックスに入力します。SweRV EH1コア複合体を追加することから開始します。「swerv\_wrapper\_verilog」を選択して、OKをクリックします。

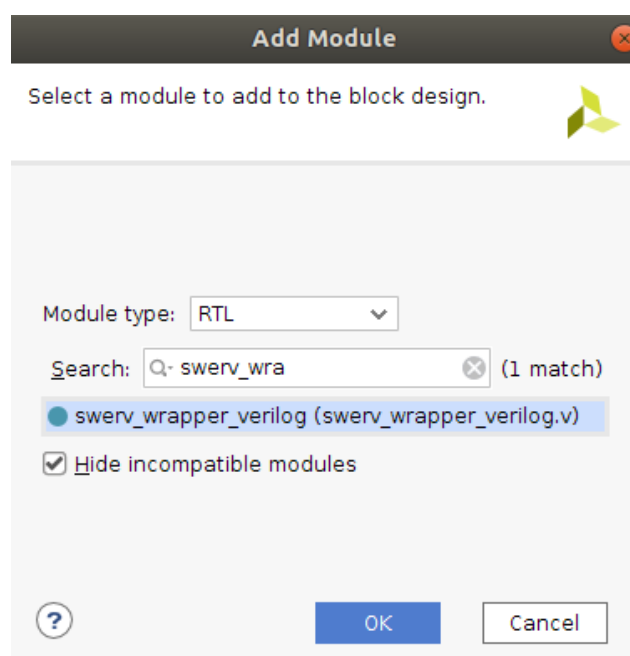


図27 : Swerv\_wrapper\_verilogを追加

非常に重要な警告メッセージがポップアップ表示されます（図28を参照）。OKをクリックして、この警告メッセージを無視します。

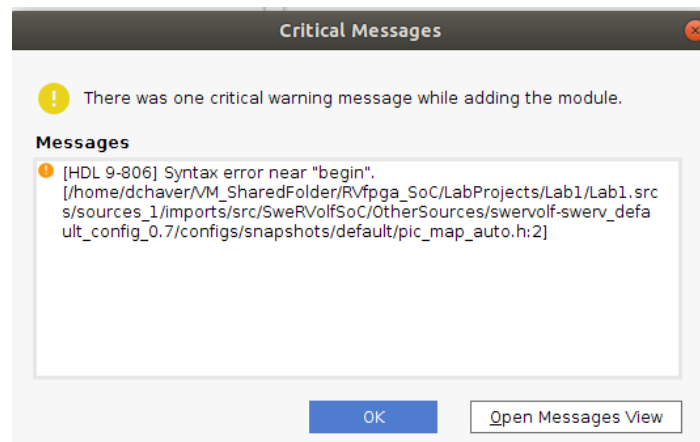


図28：非常に重要な警告メッセージ

モジュールを追加した後、モジュールの「+」アイコンをクリックすることによって、「ifu\_axi」、「lsu\_axi」、または「sb\_axi」のすべてのピンを表示して、これにアクセスできます。

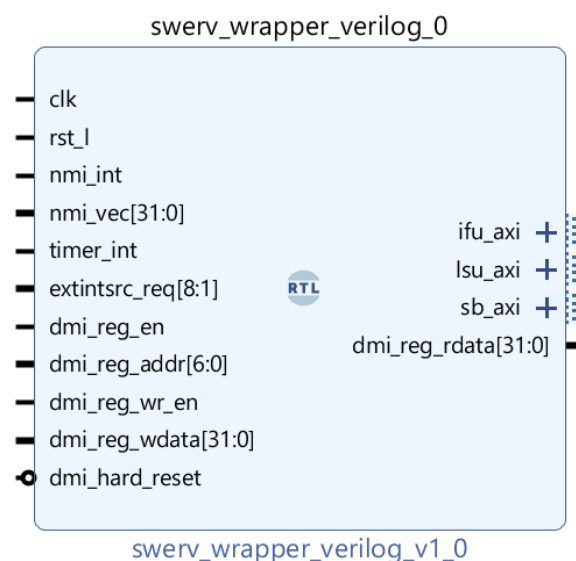


図29：「swerv\_wrapper\_verilog」モジュール

同様に、以下のモジュールを追加します。

- 「intcon\_wrapper\_bd」（相互接続ラッパモジュール）：これは、含まれている3つの相互接続モジュールすべてが含まれているラッパモジュールです。

intcon\_wrapper\_bd (intcon\_wrapper.v)

使用するSoCに必要な以下の周辺機器を追加します。

- 「Bootrom\_wrapper」（ブートROMモジュール）

bootrom\_wrapper (bootrom\_wrapper.v)

- 「gpio\_wrapper」（GPIO最上位モジュール）

gpio\_wrapper (gpio\_wrapper.v)

- 「syscon\_wrapper」 (システムコントローラモジュール)

● syscon\_wrapper (syscon\_wrapper.v)

32の「bidirec」モジュールを追加して、使用するGPIOモジュールを取り付けます。この中の16はLED用であり、16はスイッチ用です。

- 「bidirec」 (双方向GPIOモジュール)

● bidirec (bidir.v)

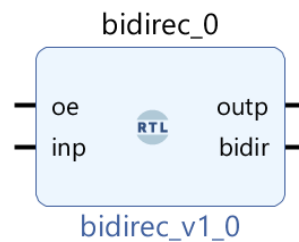


図30 : bidir GPIOモジュール

同様に、これらのモジュールの32をブロック設計に追加します。

これらの32のモジュールを追加する素早い方法は、ブロックをコピーして貼り付けることです。まず1つの「bidirec」ブロックをコピーして貼り付け、次に2つのブロックをコピーして貼り付け、その後32の「bidirec」モジュールをブロック設計に追加するまで、このコピーして貼り付けるプロセスを繰り返します。

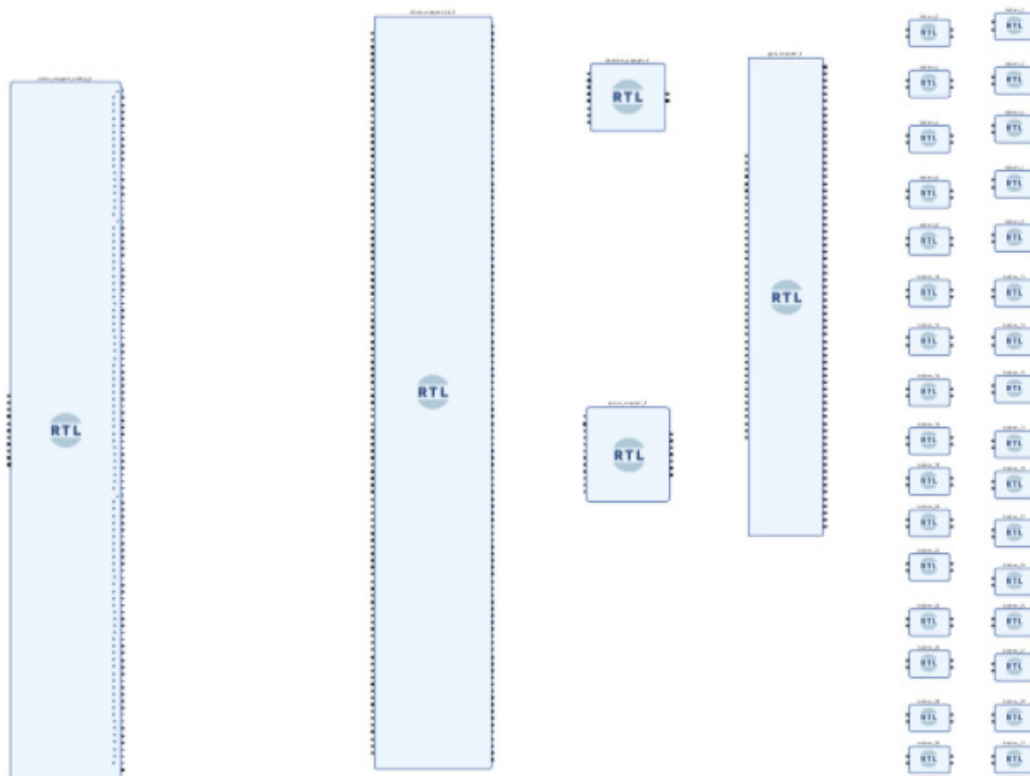


図31 : 必要なモジュールがブロック設計に追加された

(図31を参照) 左側から始めて、**SweRV Core**モジュール (swerv\_wrapper\_verilog\_0) を表示し、右側に**Interconnect Wrapper**モジュール (intcon\_wrapper\_bd\_0) および4つの周辺モジュール (**Boot-ROM** (bootrom\_wrapper\_0) モジュール)、**System Controller** (syscon\_wrapper\_0) モジュール、**GPIO** (gpio\_wrapper\_0) モジュール) を表示します。最も右側に、32の**Bidirec** (bidirec\_x) モジュールが表示されます。

#### ステップ4 : モジュールの配線を接続する

モジュール相互のピンとピンを、場合によってはバスとバスを、配線で接続します。

「swerv\_wrapper\_verilog」の「intcon\_wrapper\_bd」との接続を開始します。コアの以下のサブモジュールに関連する3つのモジュールの間で、異なる3セットのピンを接続する必要があります。

- **IFU** (命令フェッチユニット)
- **LSU** ロードストアユニット)
- **SB** (Store Byte)

まず、IFUに関連するピンを接続することから始めます。「swerv\_wrapper\_verilog」モジュールのピン「ifu\_axi\_arid[2:0]」を「intcon\_wrapper\_bd」の「i\_ifu\_arid[2:0]」ピンに接続します。

同様に、

*ifu\_axi\_araddr[31:0]*を*i\_ifu\_araddr[31:0]*に接続し、

*ifu\_axi\_arlen[7:0]*を*i\_ifu\_arlen[7:0]*に接続し、

*ifu\_axi\_arsize[2:0]*を*i\_ifu\_arsize[2:0]*になど、順次接続します (図32を参照)。

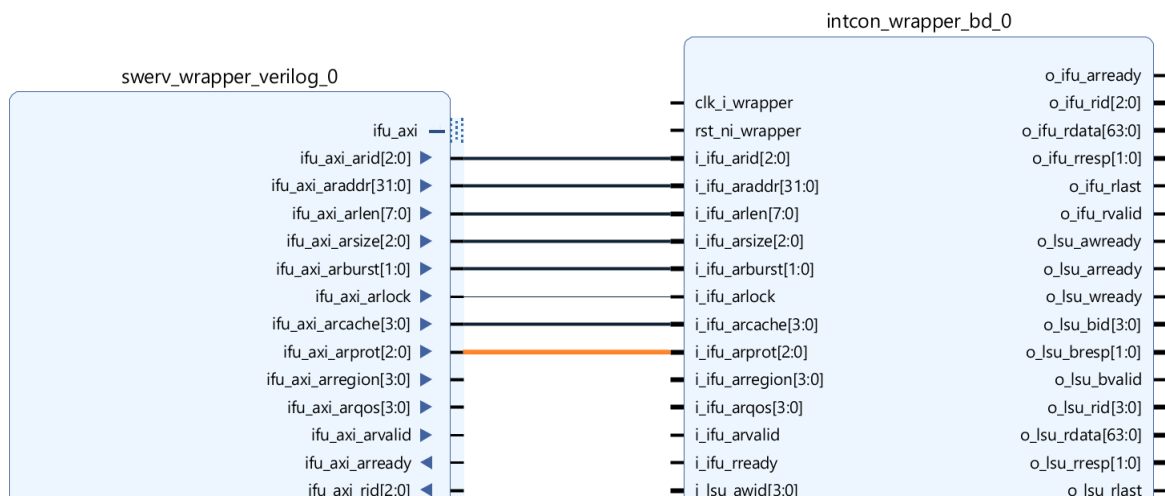


図32 : 関連するピンを接続

同様に、「swerv\_wrapper\_verilog」のすべてのIFU (命令フェッチユニット) ピンを、「intcon\_wrapper\_bd」のIFUのピンと、接続します (図33を参照)。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / InternalConnections / 1\\_SwervW\\_IntconW\\_IFU.pdf](#)

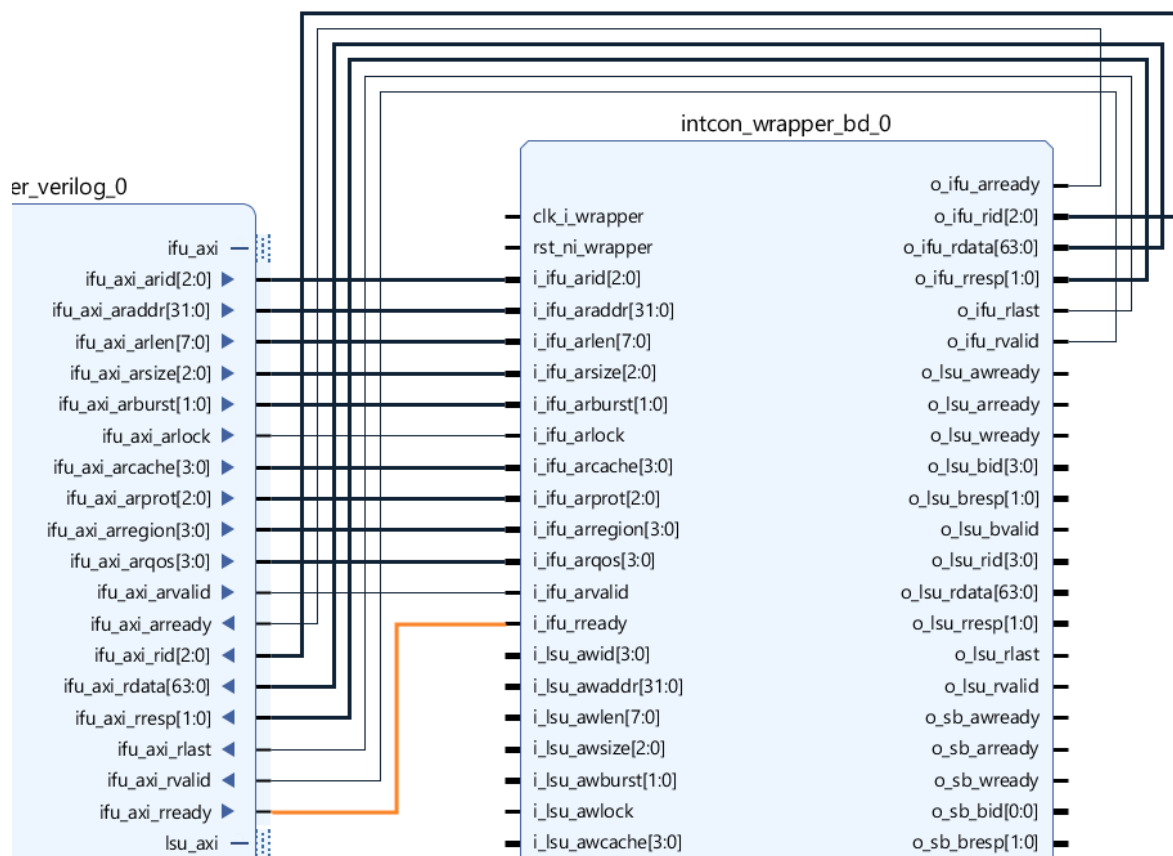


図33 : すべてのIFUピンを接続

移動して、「**swerv\_wrapper\_verilog**」のすべてのLSU（ロードストアユニット）ピンを「**intcon\_wrapper\_bd**」のLSUのピンに接続します。IFUピンに行ったのと同じプロセスを実行して、「**swerv\_wrapper\_verilog**」モジュールの各LSUピンを「**intcon\_wrapper\_bd**」モジュールのそれぞれのピンと接続します（図34を参照）。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / InternalConnections / 2\\_SwervW\\_IntconW\\_LSU.pdf](#)



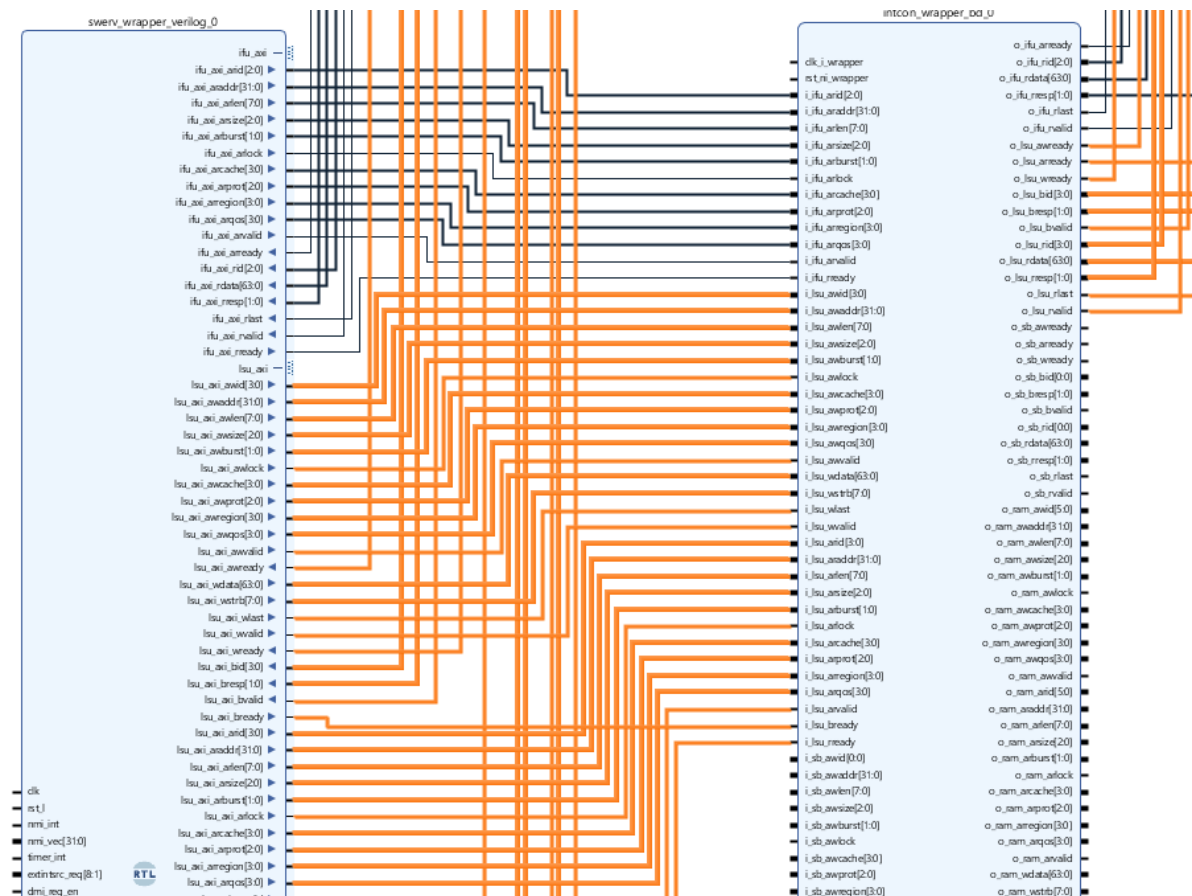


図34 : すべてのLSUピンを接続

続行してSBピンを接続します。同様に、「swerv\_wrapper\_verilog」のすべてのSBピンを、「intcon\_wrapper\_bd」のそれぞれのSBのピンと接続します（図35を参照）。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / InternalConnections / 3\\_SwervW\\_IntconW\\_SB.pdf](#)

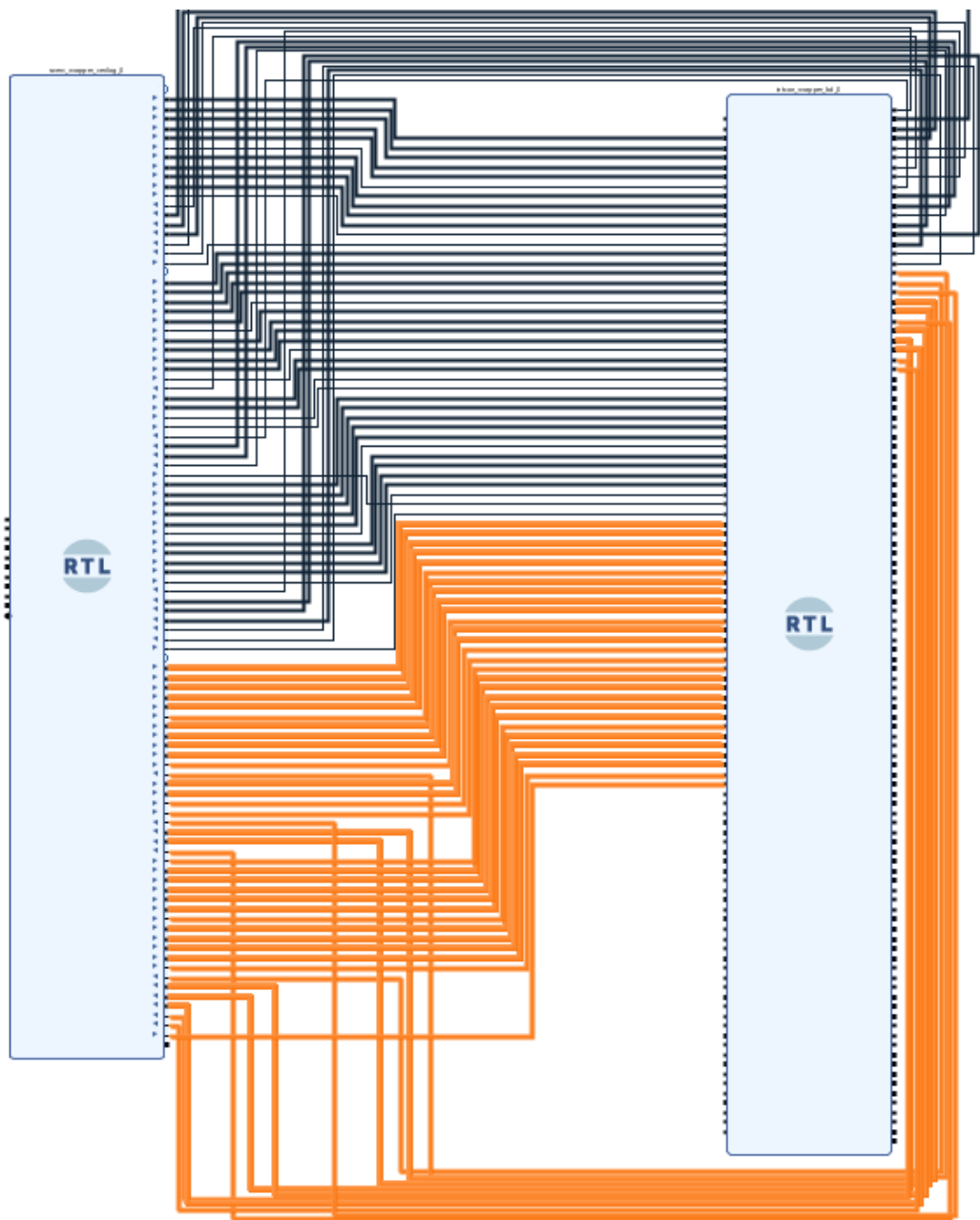


図35 : すべてのSBピンを接続

次に、周辺機器を「Intcon\_wrapper\_bd」と接続します。「Intcon\_wrapper\_bd」の「wb\_rom\_xxx\_x」ワイヤを接続することによって、「bootrom\_wrapper」モジュールから始めます（図36を参照）。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/4\\_BootRomW\\_IntconW.pdf](#)

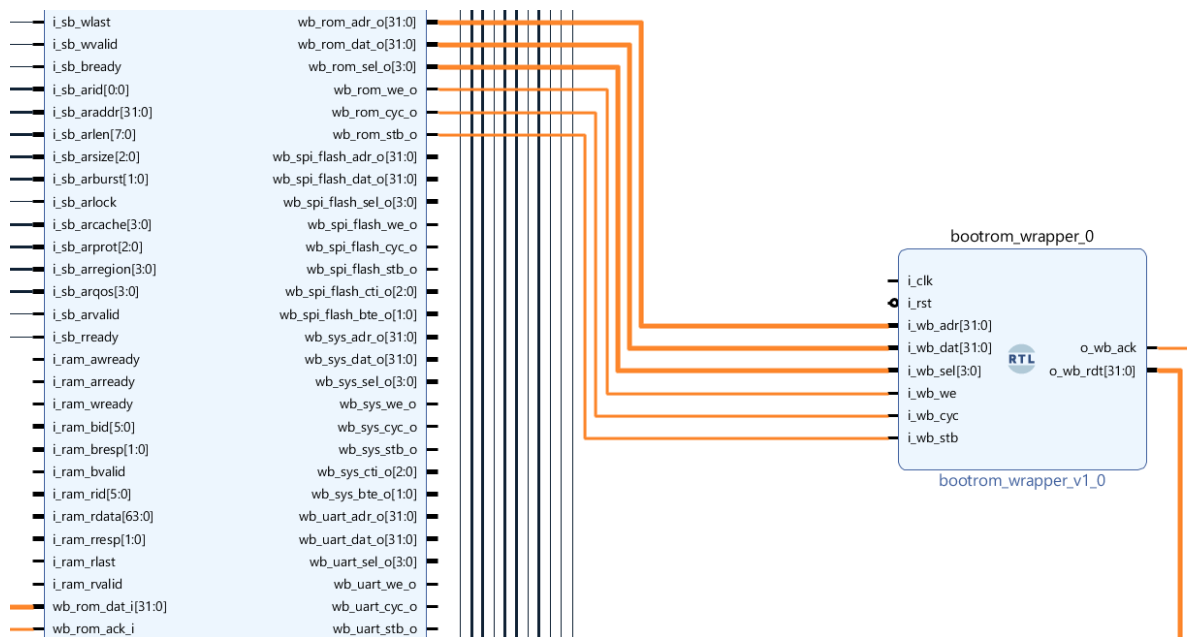


図36 : BootROMモジュールを相互接続ラッパモジュールと接続

「syscon\_wrapper」モジュールを「Intcon\_wrapper\_bd」モジュールと接続します（図37を参照）。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます：  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / InternalConnections / 5\\_SysconW\\_IntconW.pdf](#)

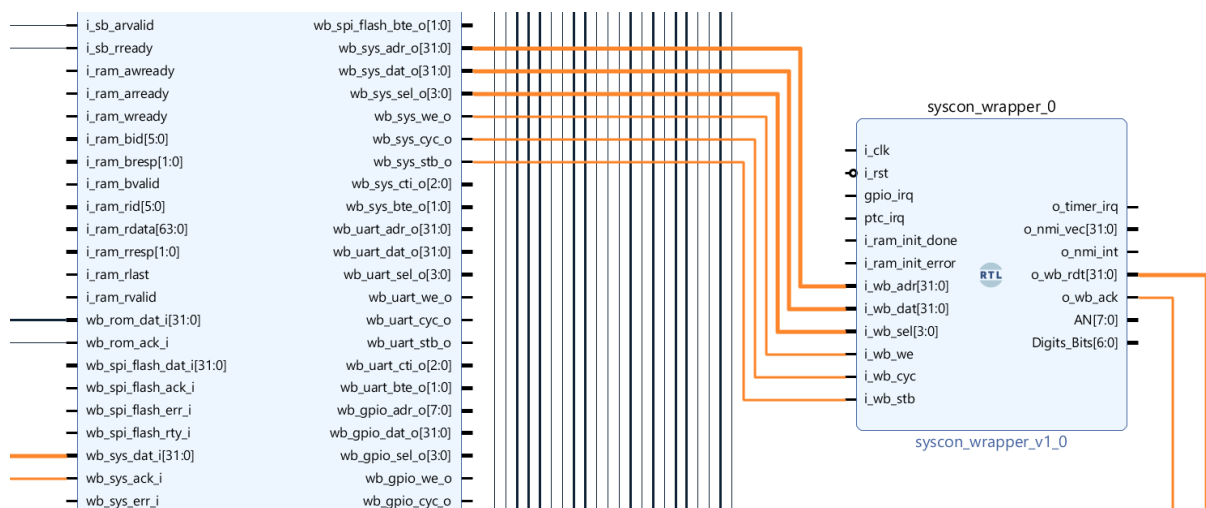


図37 : システムをWB相互接続ピンと接続

「syscon\_wrapper」の以下のピンは、「swerv\_wrapper\_verilog」に接続されます（図38を参照）。

- o\_timer\_irq
- o\_nmi\_vec[31:0]
- o\_nmi\_int

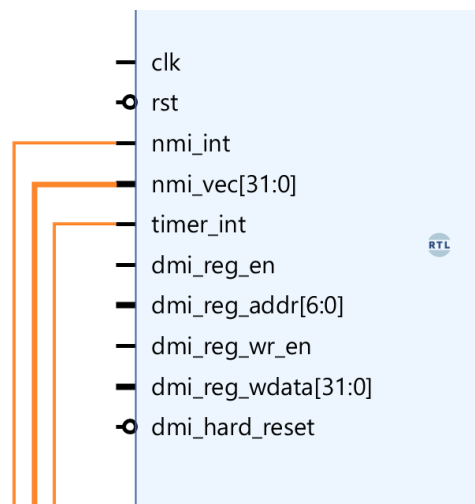


図38 : syscon\_wrapperをswerv\_wrapper\_verilogピンと接続

ここで、「gpio\_wrapper」モジュールを「intcon\_wrapper\_bd」と接続します。

「intcon\_wrapper\_bd」モジュールの「wb\_gpio\_XXX\_X」ピンを「gpio\_wrapper」モジュールのピンと接続します（図39を参照）。

「gpio\_wrapper」モジュールの「wb\_inta\_o」ピンを、「syscon\_wrapper」モジュールの「gpio\_irq」ピンと接続します。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / InternalConnections / 6\\_GpioW\\_IntconW.pdf](#)

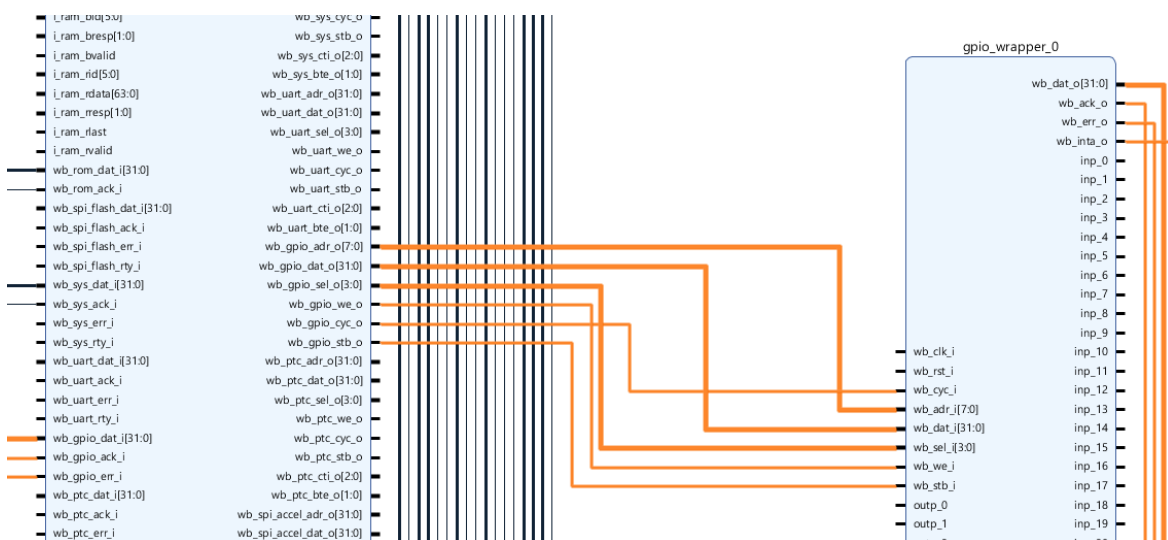


図39 : gpio\_wrapperをintcon\_wrapperピンと接続

32 GPIO「bidirec」モジュールを、既に接続されている「gpio\_wrapper」モジュールと接続します。特に、「gpio\_wrapper」モジュールを「bidirec\_x」モジュールと接続します（ここで、xは1～32の数字です）。接続は以下のように行われます。

「gpio\_wrapper\_0」の「inp\_0」ピンは、「bidirec\_0」の「inp」に接続されます。  
 「gpio\_wrapper\_0」の「inp\_1」ピンは「bidirec\_1」の「inp」に接続され、同様にこれらの接続は最後の「inp」接続まで続行されます。つまり、最後に「gpio\_wrapper」の「inp\_31」は「bidirec\_31」の「inp」に接続されます。

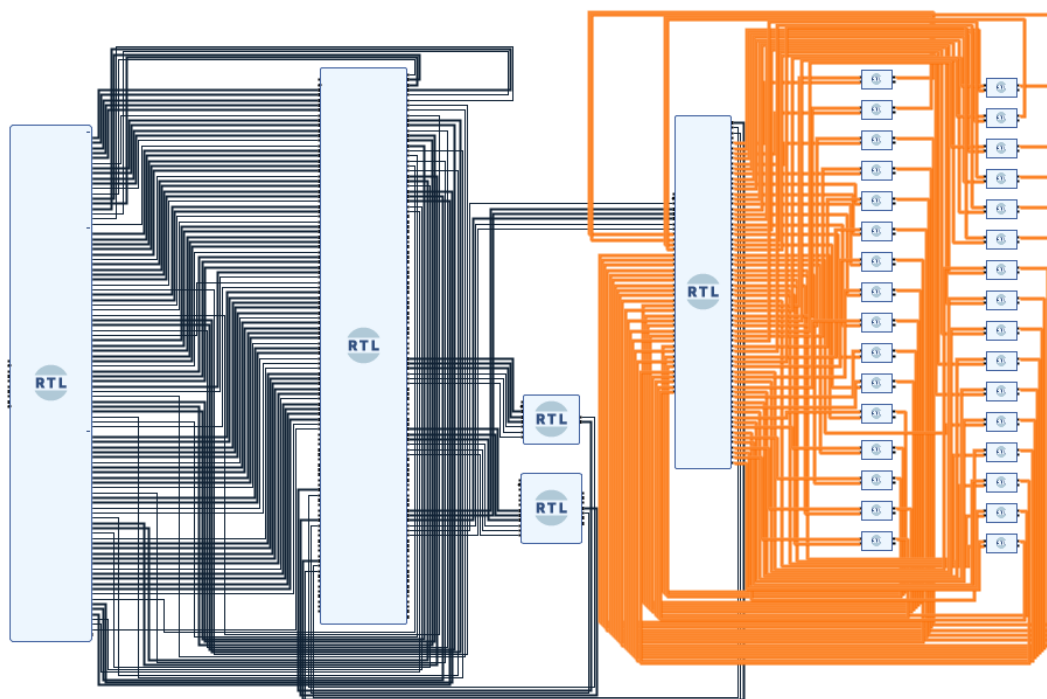
同様に、

「gpio\_wrapper\_0」の「oe\_0」ピンは「bidirec\_0」の「oe」に接続されます。  
 「gpio\_wrapper\_0」の「oe\_1」ピンは「bidirec\_1」の「oe」に接続され、同様にこれらの接続は最後の「oe」接続まで続行されます。つまり、最後に「gpio\_wrapper」の「oe\_31」は「bidirec\_31」の「oe」に接続されます。

同様に再び、

「gpio\_wrapper\_0」の「outp\_0」ピンは「bidirec\_0」の「outp」に接続されます。  
 「gpio\_wrapper\_0」の「outp\_1」ピンは「bidirec\_1」の「outp」に接続され、同様にこれらの接続は最後の「outp」接続まで続行されます。つまり、最後に「gpio\_wrapper」の「outp\_31」は「bidirec\_31」の「outp」に接続されます。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/InternalConnections/7\\_GpioW\\_32xBidirec.pdf](#)



**図40 : gpio\_wrapperモジュールに接続されたすべてのGPIO Bidirecモジュール**

モジュール間のすべての内部接続を完了したので、外部接続をします。

#### ステップ5 : I/Oピンの外部接続を実行する

使用するブロック設計に入力として受け取る、または使用するブロック設計から出力として出ていくピンを接続しましょう。これらのピンを、外部ピン/ポートとして接続します。こ

これらのピンには、**RAM**（DDR）、**CLK**（クロック）、**RST**（リセット）、**DMI**（デバッグモジュールインターフェース）のピンが含まれます。

「**clk**」ピンの接続から始めます。「**swerv\_wrapper\_verilog**」モジュールに移動し、「**clk**」ピンを右クリックすると、ドロップダウンが表示されます（図41を参照）。すべてのドロップダウンオプションからMake Externalオプションを選択します。ピンを左クリックしてショートカットキー「CTRL + T」を使用して、ピンをExternalにすることもできます。

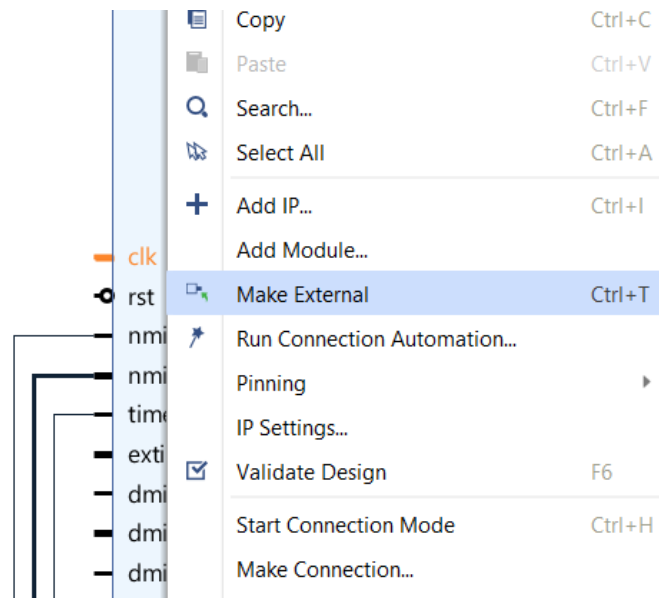


図41 : 「clk」を外部接続にする

外部ピン「**clk\_0**」に接続された「**swerv\_wrapper\_verilog**」の「**clk**」ピンが表示されます（図42を参照）。

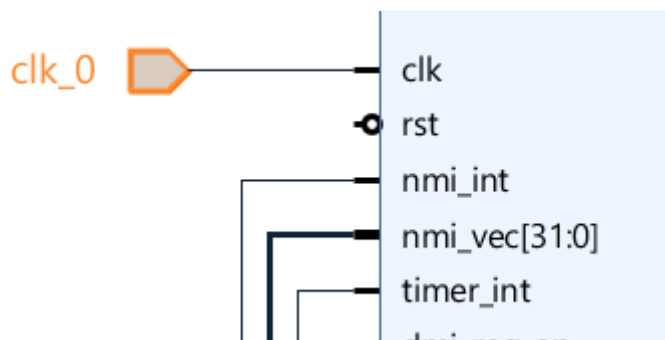


図42 : 「clk」は外部接続になる

「**clk**」外部ピンを、**intcon\_wrapper\_bd**、**syscon\_wrapper**、**bootrom\_wrapper**、**gpio\_wrapper**などの残りのモジュールと接続できます。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / ExternalConnections / 1\\_Clock.pdf](#)

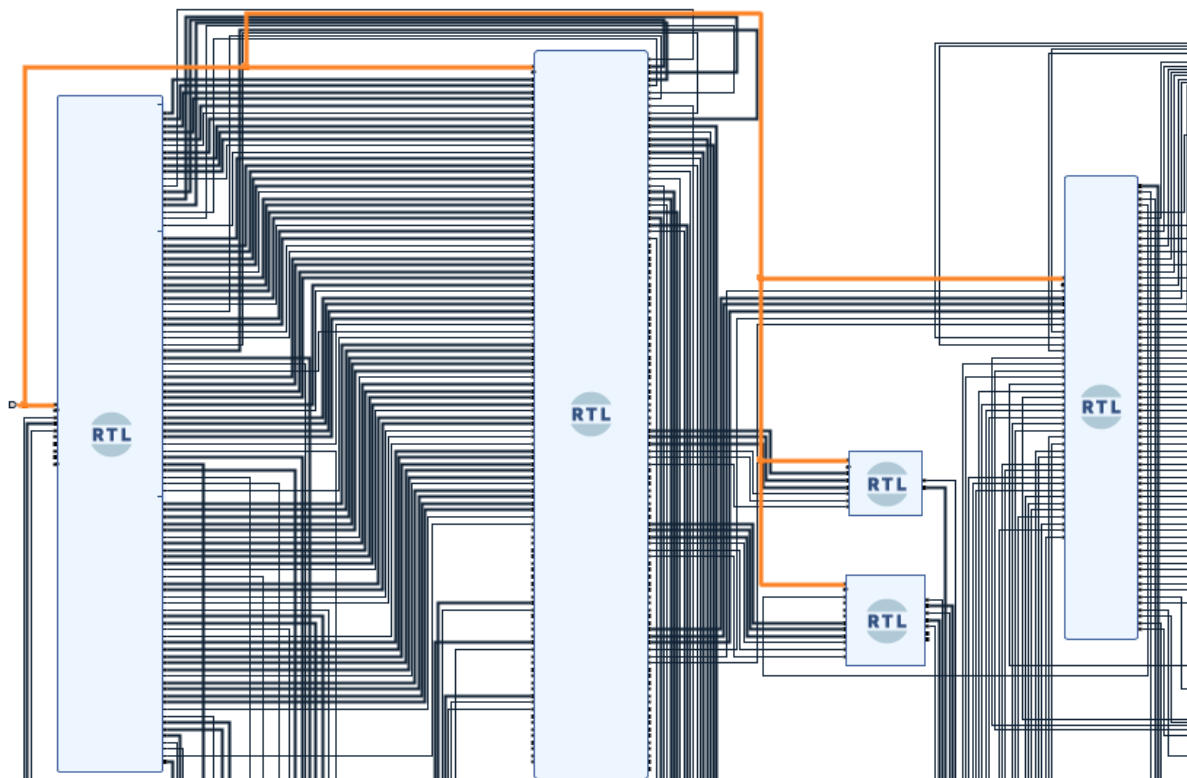


図43：すべてのモジュールに接続されている信号clk

同様に、「rst」ピンをすべてのモジュールに接続できます。

「clk」用に作成した外部ピンのように、「rst」用の外部ピンを作成します。再び「swerv\_wrapper\_verilog」モジュールに移動して、「rst」ピンを右クリックし、これを外部ピンにします。

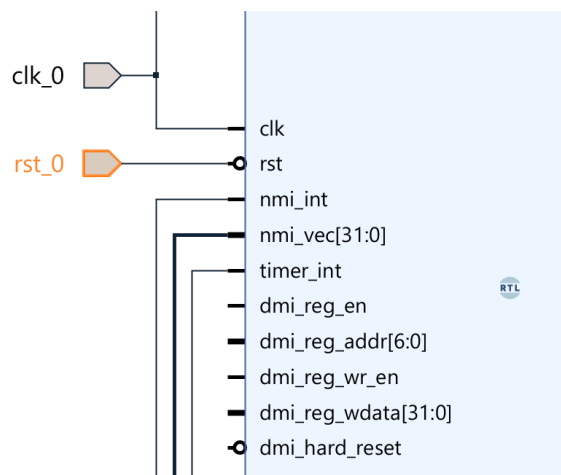


図44：rst\_Iを外部ピンにする

「rst\_0」外部ピンをintcon\_wrapper、syscon\_wrapper、bootrom\_wrapper、gpio\_wrapperなどの残りのモジュールと接続します。



**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます：  
[\[RVfpgaSoCPath\] /RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs /ExternalConnections/2\\_Reset.pdf](#)

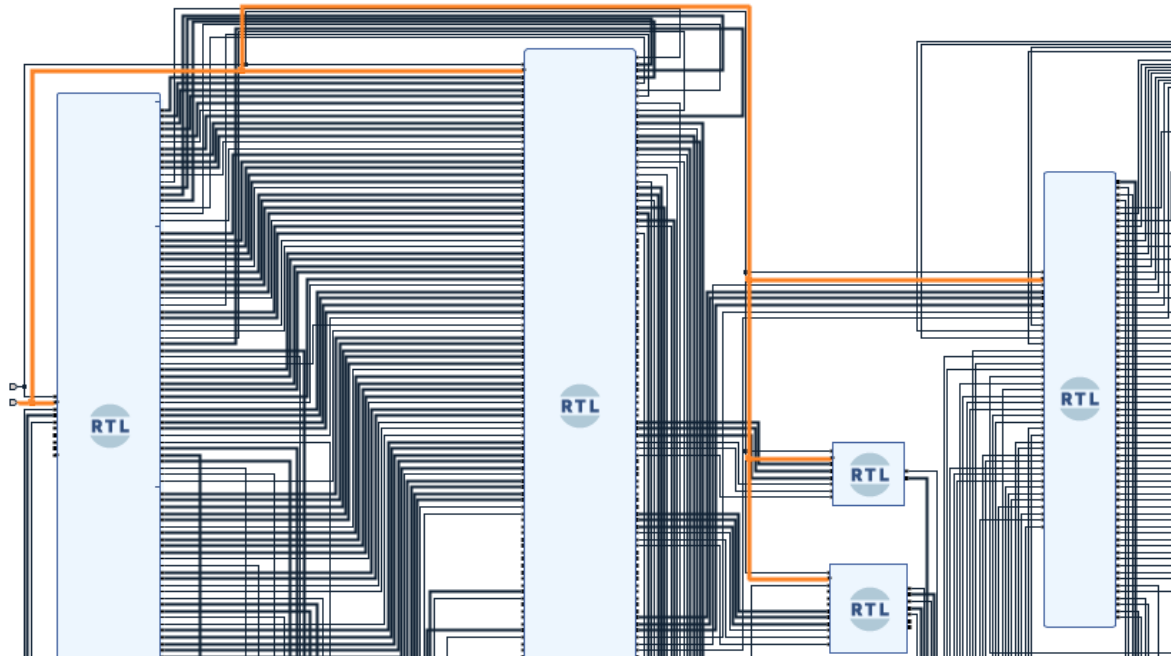


図45 : 反転した「rst\_0」ピンを残りのモジュールに接続

以下のステップを実行して、「Intcon\_wrapper\_bd」モジュールのすべてのRAM（DDR）ピンを外部RAMピンに接続します。

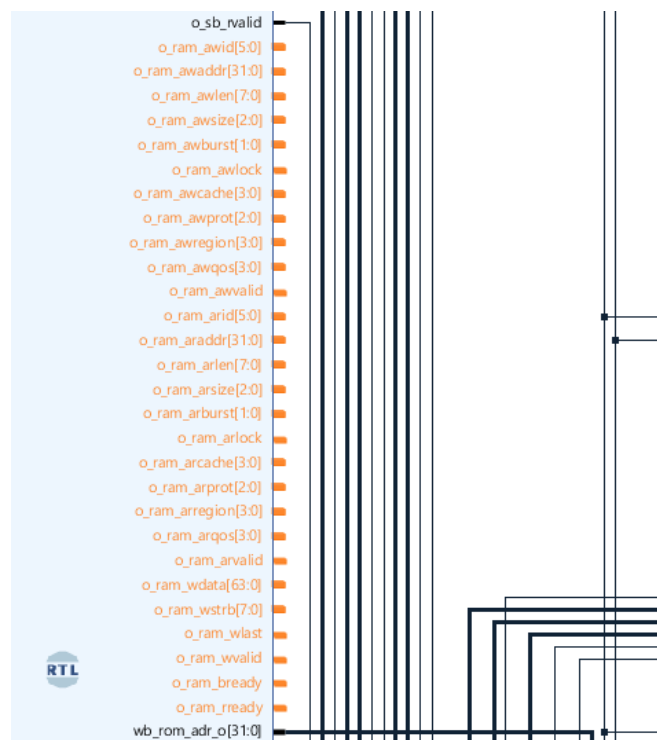


図46 : Intconラッパの右側のRAMピン

「Intcon\_wrapper\_bd」モジュールの右側のすべてのRAMピンを、外部ピンにします（図47を参照）。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます：  
[\[RVfpgaSoCPath\]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs/ExternalConnections/3\\_RAM\\_R.pdf](#)

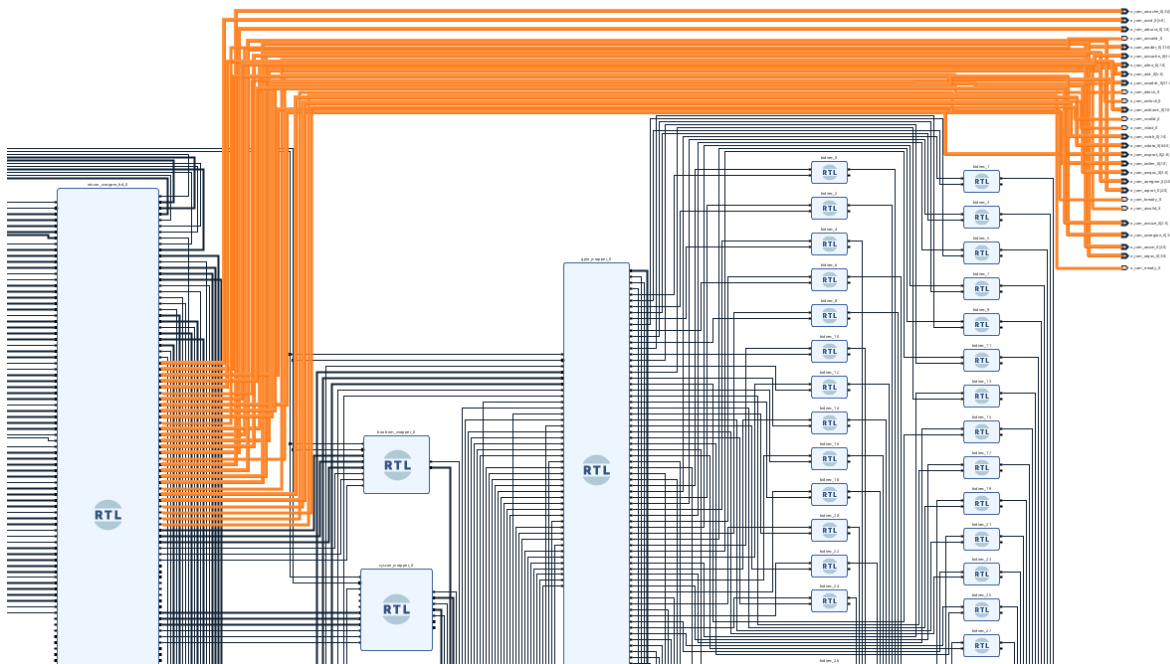


図47：右側のすべてのRAMピンを外部ピンにする

「Intcon\_wrapper\_bd」の左側のRAMピンを外部ピンにします。

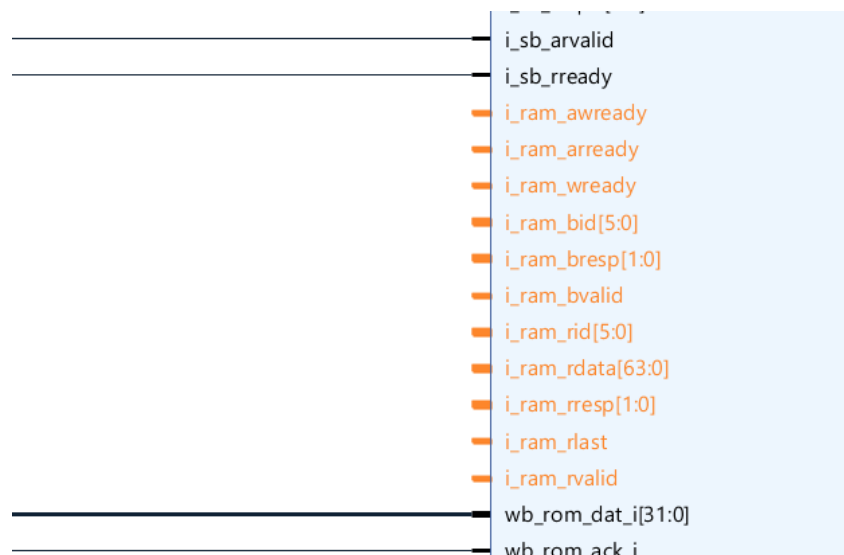


図48：相互接続ラッパの左側RAMピン

これらのRAMピンすべてを外部ピンにします（図49を参照）。

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs  
/ExternalConnections/4\_RAM\_L.pdf

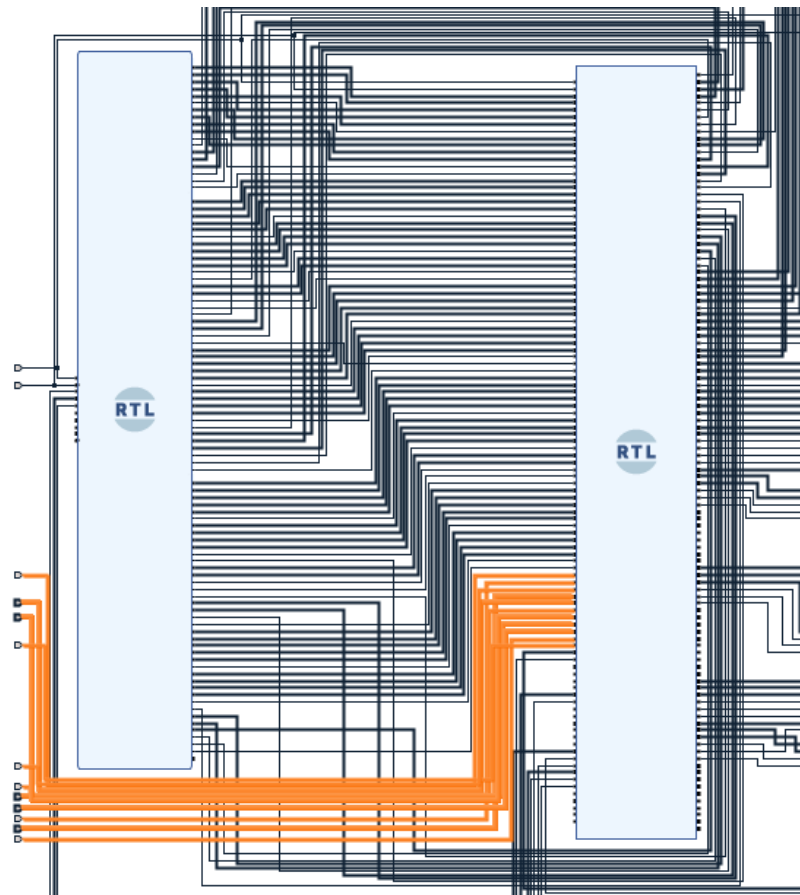


図49 : 左側のすべてのRAMピンを外部ピンにする

「swerv\_wrapper\_verilog」モジュールのDMIピンを外部ピンに接続します。

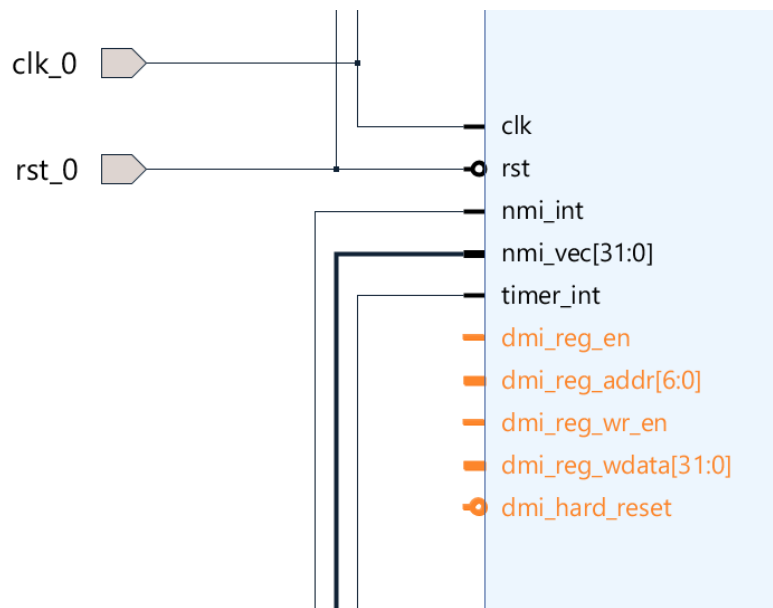


図50 : dmiピンop swerv\_wrapper\_verilog (左側)

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs / ExternalConnections/5\\_DMI.pdf](#)

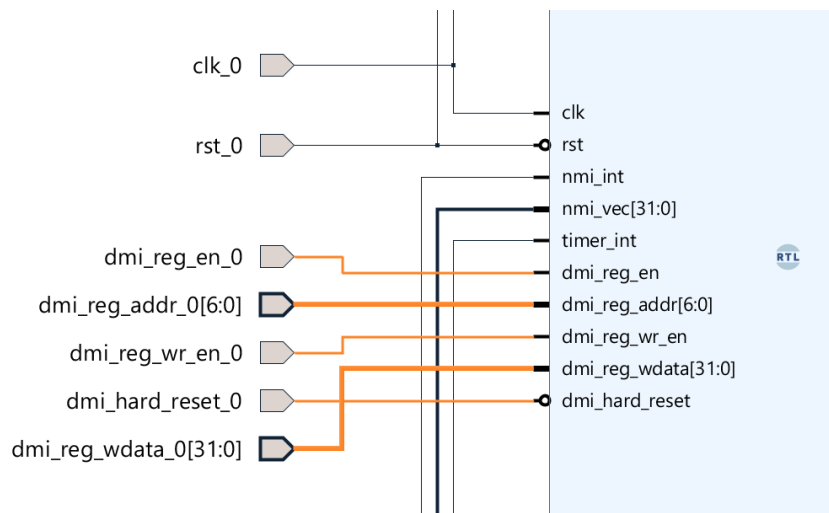


図51 : Dmiピンを外部ピンにする

もう1つのピンを、「swerv\_wrapper\_verilog」モジュールの右下側の外部ピンに接続します。このピンは「dmi\_reg\_rdata[31:0]」です。

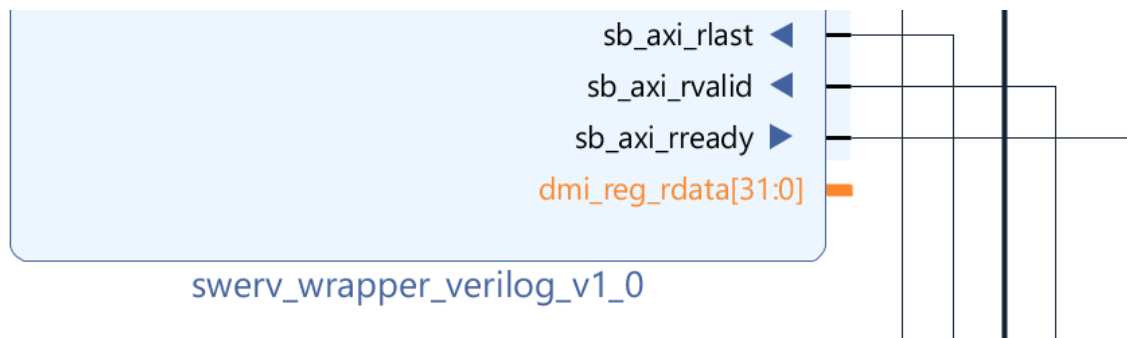


図52 : 「dmi\_reg\_rdata[31:0]」ピン (swerv\_wrapper\_verilogの右側)

「dmi\_reg\_rdata[31:0]」も外部ピンにします。

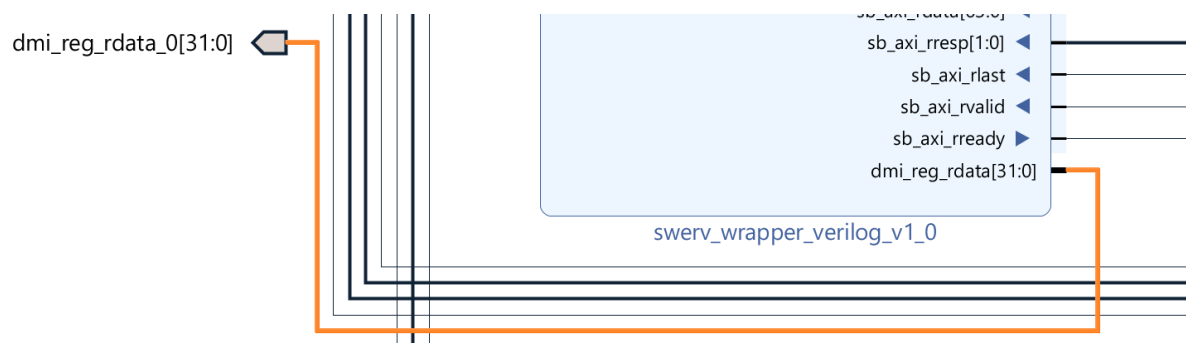


図53 : 「dmi\_reg\_rdata[31:0]」ピンを外部ピンにする

ここで「syscon\_wrapper」モジュールの以下のピンを外部ピンにします。

- i\_ram\_init\_done
- i\_ram\_init\_error
- AN[7:0]
- Digital\_Bits[6:0]

**PDF :** 配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます :  
[\[RVfpgaSoCPath\] / RVfpgaSoC / Labs / LabResources / Lab1 / BlockDesignPDFs / ExternalConnections / 6\\_SysconW\\_External.pdf](#)

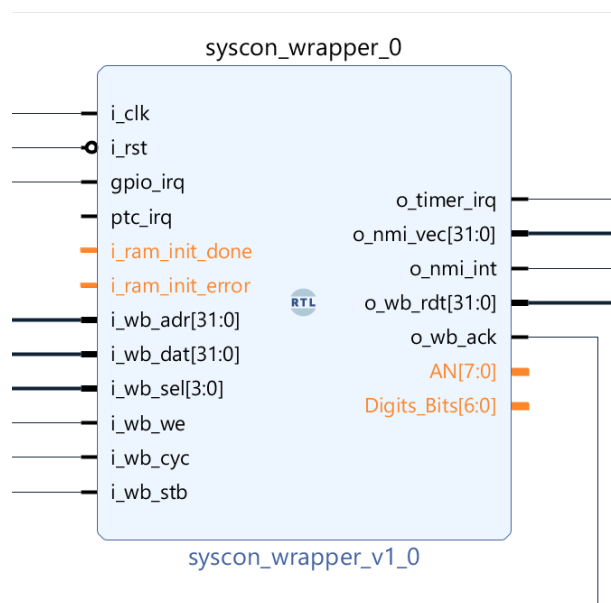


図54 : syscon\_wrapperの外部ピン

残っている最後の接続は、すべての「bidirec」モジュールのすべての「bidir」ピンを外部ピンにすることです。

**注意：**これらの接続を、「bidirec\_0」モジュールから始めて1つずつ外部接続にします。つまり、「bidirec\_0」モジュールの「bidir」ピンが外部ピン「bidir\_0」に接続されます。次に「bidirec\_1」の「bidir」ピンに移動し、以後同様に実行します。

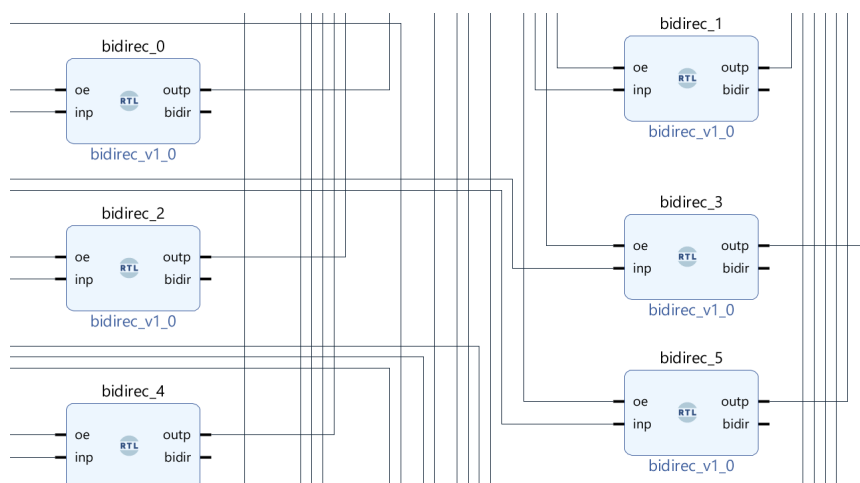
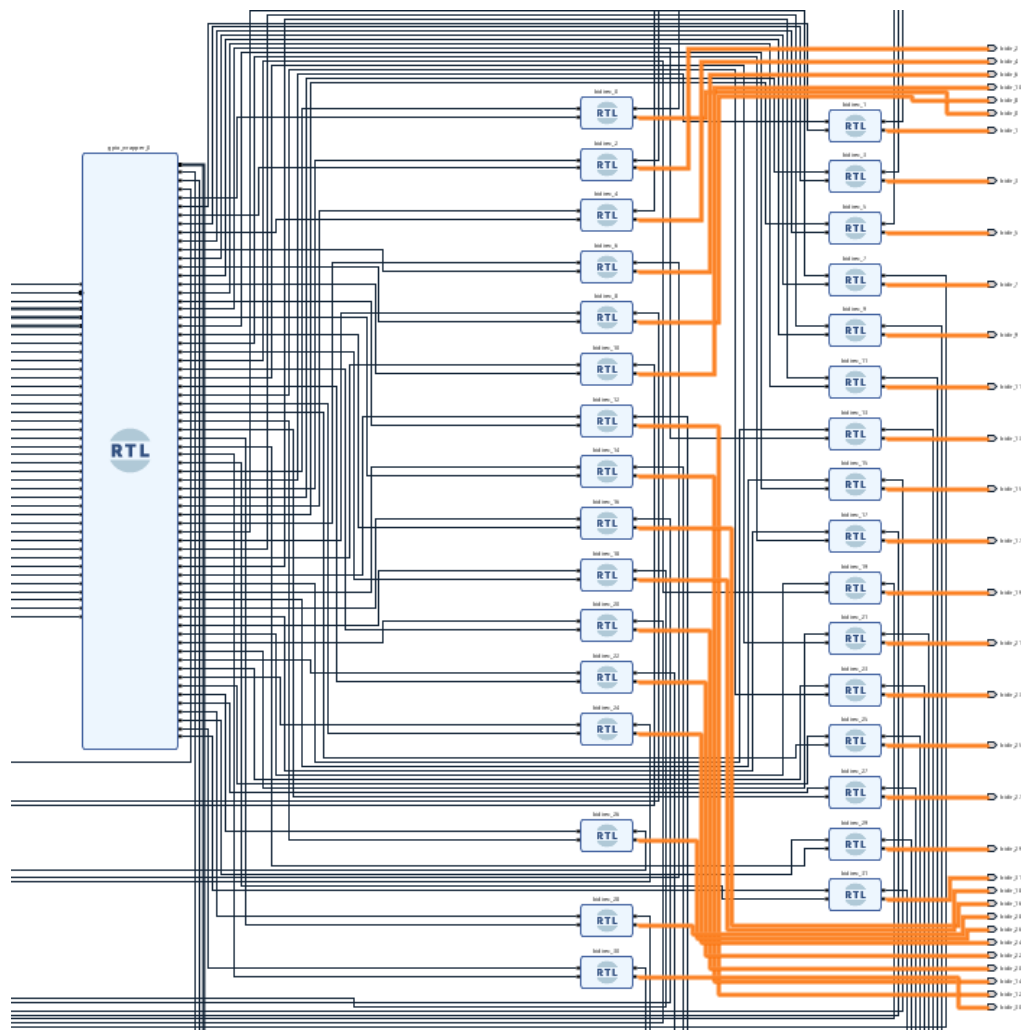


図55 : GPIO Bidirecモジュールの「bidir」ピンを外部接続にする

**PDF：**配線のクローズアップ詳細を示すブロック設計の高品質PDFは、次記で入手できます：  
[\[RVfpgaSoCPath\] / RVfpgaSoC/Labs/LabResources/Lab1/BlockDesignPDFs / ExternalConnections/7\\_Bidir.pdf](#)



**図56 : 「bidir」を外部接続にする**

ブロック設計SoCの内部接続と外部接続の両方のすべてを、完了しました。「**Ctrl + S**」を押して、ブロック設計を保存します。

SweRVofX SoCを完了した後に形作られたブロック設計には、以下の接続されたモジュールが含まれています。

- 1つのSweRVコア (swerv\_wrapper\_verilog)
- 1つの相互接続ラッパ (intcon\_wrapper\_bd)
- 1つのブートROM (bootrom\_wrapper)
- 1つのGPIO最上位モジュール (gpio\_wrapper)
- 1つのシステムコントローラ (syscon\_wrapper)
- 32のBidirec Gpioモジュール (bidirec)

(図57を参照)。



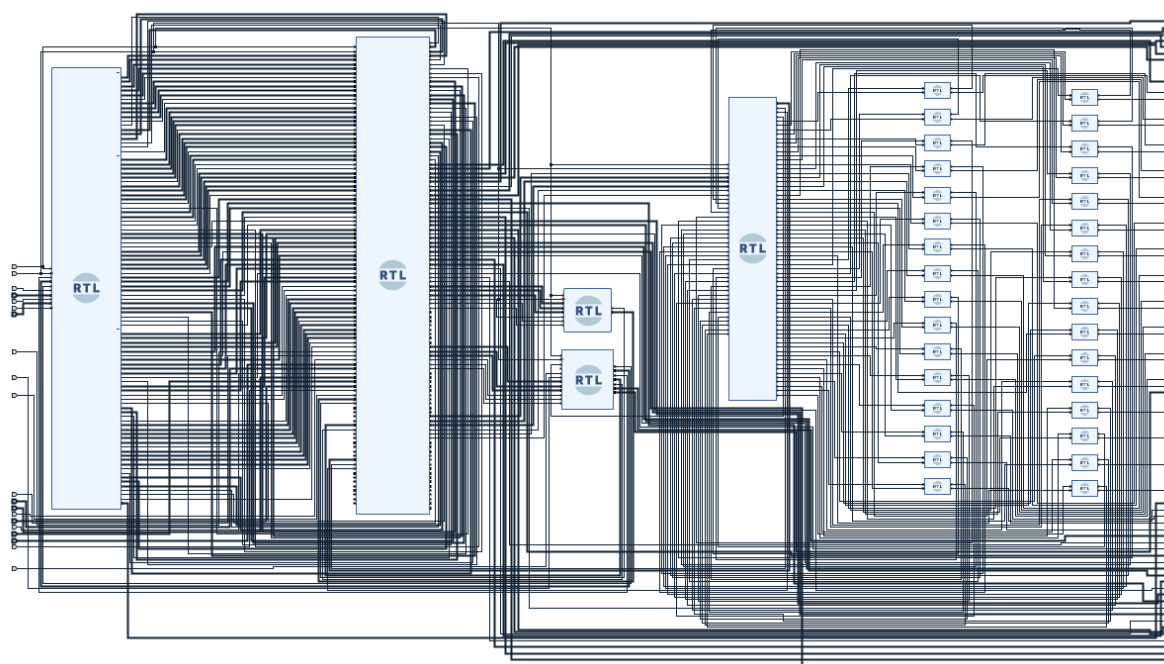


図57：完成したブロック設計SoC

## 5. ブロック設計モジュールVerilogファイルの生成

それでは、作成したブロック設計のVerilogモジュールファイルを生成します。

**ステップ1：**ソースパネルに移動し、作成したばかりのブロック設計モジュール「BD」を見つけます。

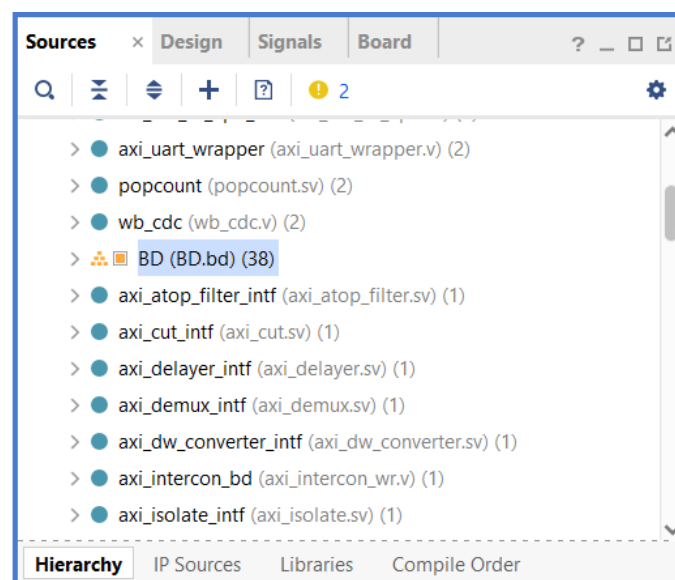


図58：ソースで「BD」を見つける

**ステップ2：**ブロック設計（BD）で右クリックして、「Create HDL Wrapper」を選択します（図59を参照）。

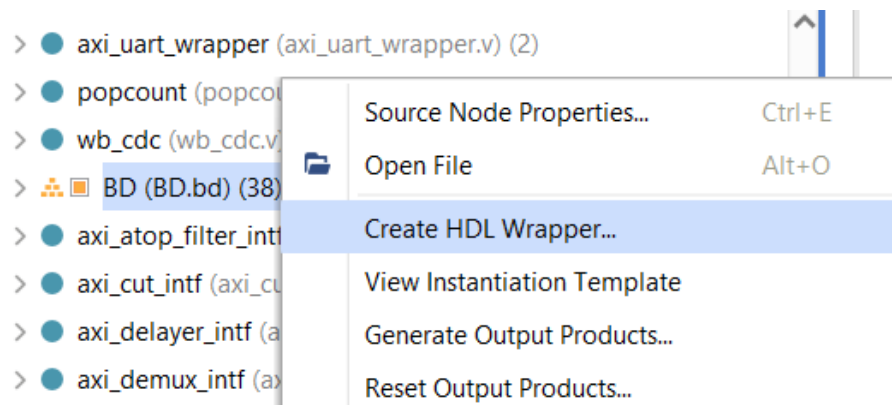


図59 : HDLラッパを作成

ステップ3 : 「Let Vivado manage wrapper and auto-update」 オプションを選択し、OK をクリックして続行します。

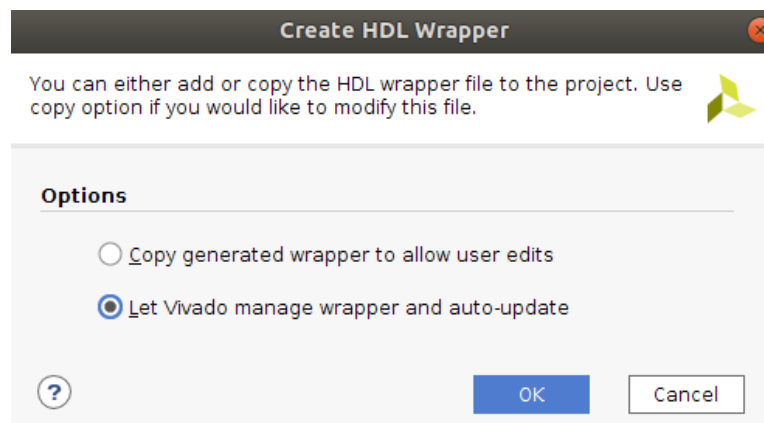


図60 : 第2のオプションを選択

使用するブロック設計にいくつかのピンが未接続で残っているため、これらのピンが「0」（接地）に自動的に接続されるように、非常に重要な警告のポップアップが表示されます。

OKをクリックします。

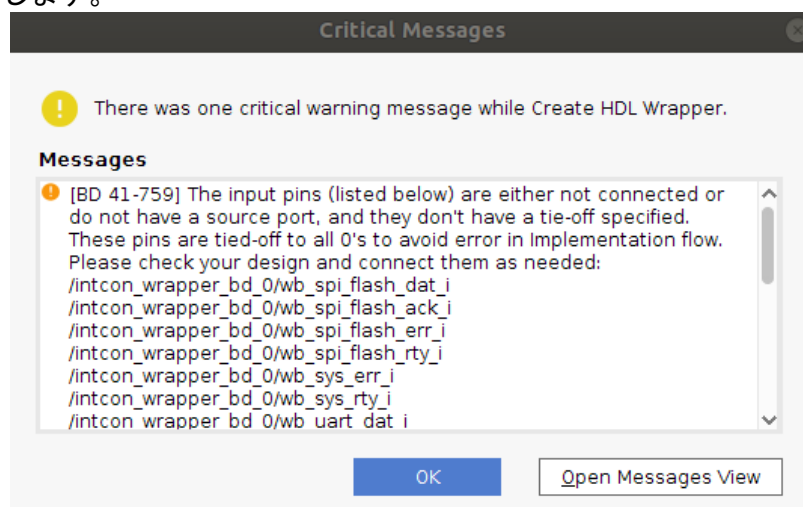
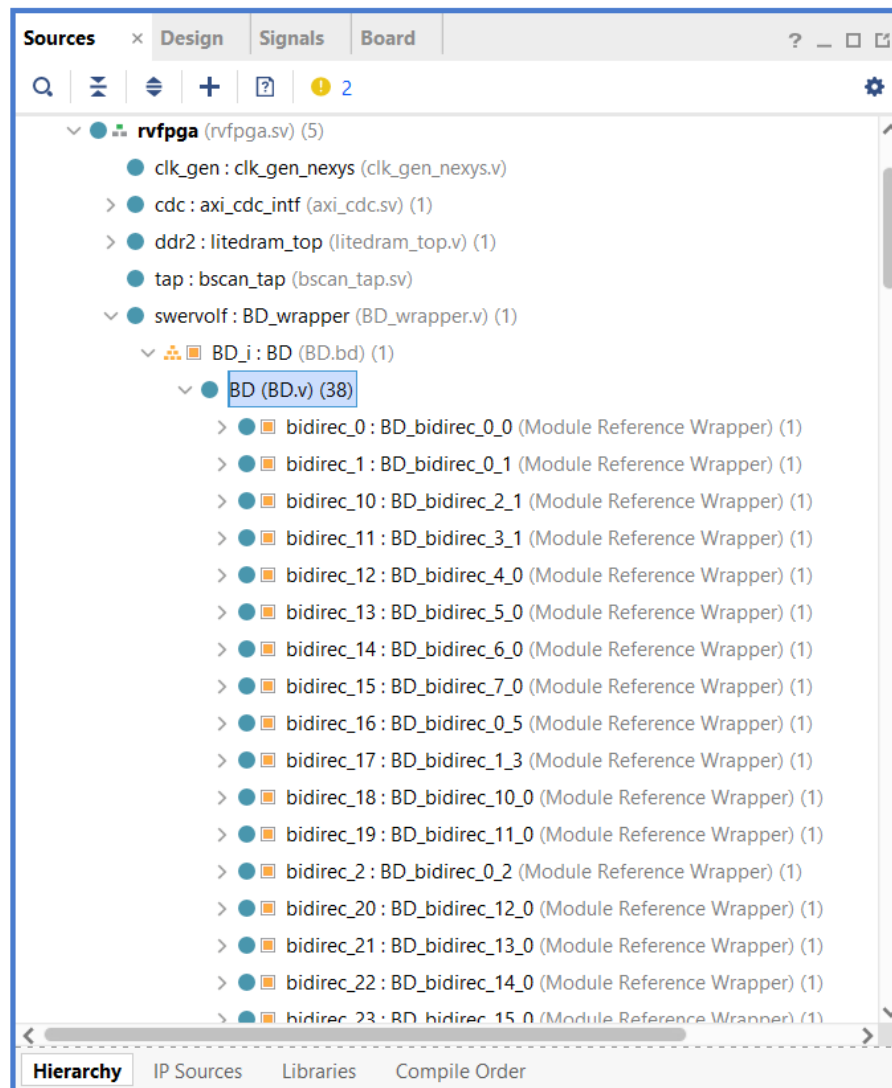


図61 : 警告のポップアップ

これで、ブロック設計のHDLラッパが作成されました。Sourcesパネルに移動して、「BD\_wrapper」が表示されるまで、下方にスクロールします。その横のドロップダウンアイコンをクリックして、再度「BD\_i」をクリックします。

「BD (Bd.v)」ファイルをダブルクリックして、これを開きます（図62を参照）。



**図62 : ソースパネルで「BD.v」を見つける**

Vivadoのブロック設計ツールを使用して作成した「BD.v」 Verilogファイルが表示されます。

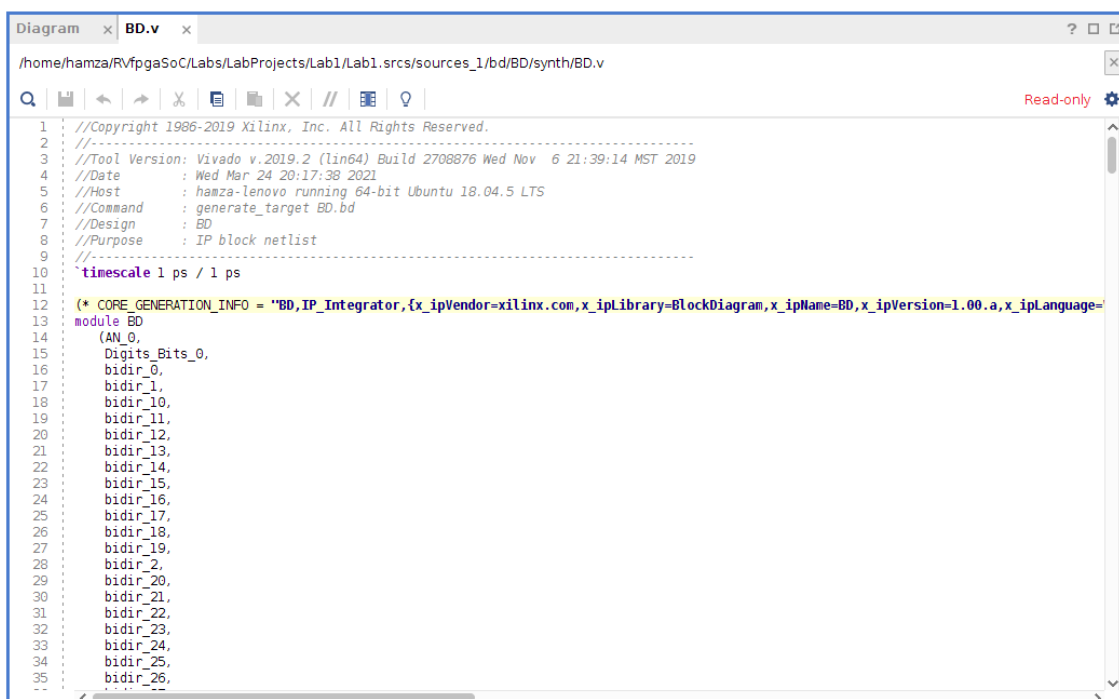


図63 : 「BD.v」

この新規作成したファイルのパスをファイルの最上部に表示しています。次のラボで、このパスを使用してこの「BD.v」ファイルにアクセスします。

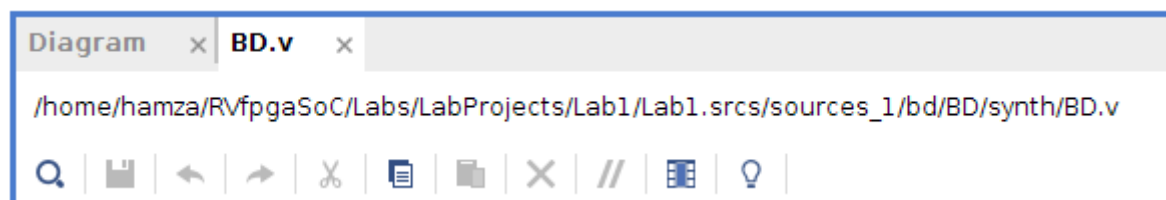


図64 : 「BD.v」 ファイルのパス

## 6. ビットストリームの生成

Vivadoのブロック設計ツールを使用してSweRVolfXサブセットが作成され、Verilogラッパが生成されており、FPGAを構成するビットストリームを生成する準備が完了しています。ビットストリームを生成するには、まずVivadoで以下のステップを実行して、一部の設定を調整する必要があります。

### ステップ1 : 設定に移動する

Vivadoのナビゲーションバーの左上の「Tools」に移動して、オプションから「Settings」を選択します。

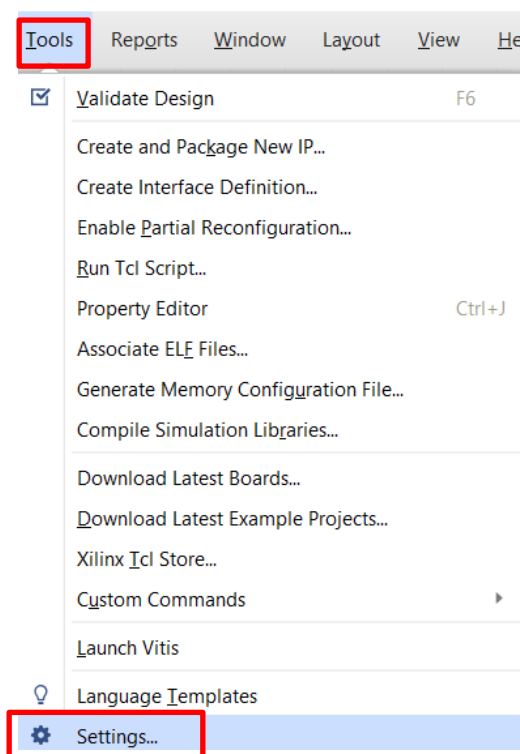


図65：設定に移動

## ステップ2：Generalタブに移動する

「General」タブに移動して、言語オプションセクションで「Verilog options」を選択します。

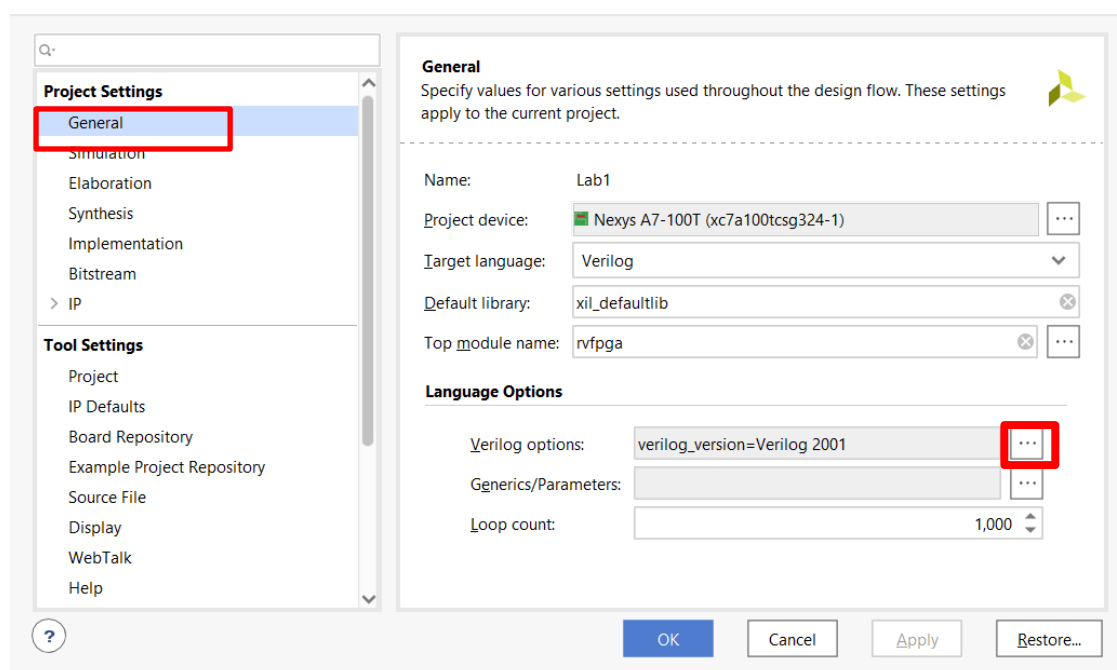


図66：一般的設定

## ステップ3：インクルードファイルへのパスを追加する

「+」ボタンをクリックして、Verilog検索パスにInclude Filesを追加します。

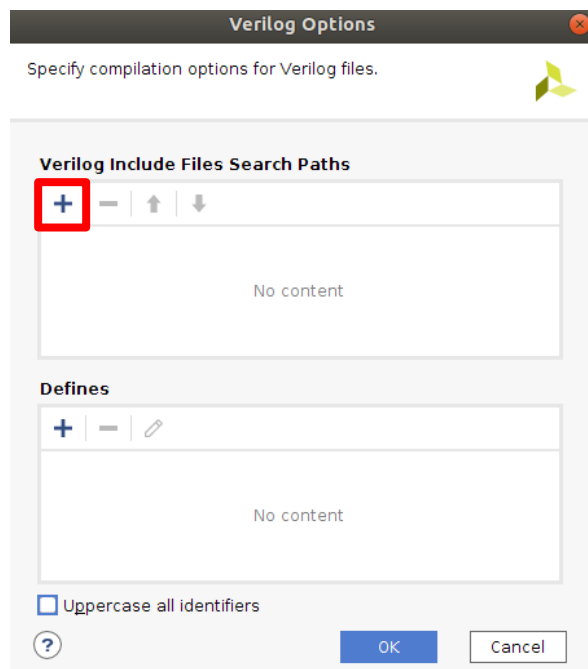


図67 : Verilogオプション

以下の3つのパスを追加します。

- [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srscs/sources\_1/imports/src/SweRVolfSoC/Interconnect/AxiInterconnect/pulp-platform.org\_\_axi\_0.25.0/include
- [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srscs/sources\_1/imports/src/OtherSources/pulp-platform.org\_\_common\_cells\_1.20.0/include
- [RVfpgaSoCPath]/RvfpgaSoC/Labs/LabProjects/Lab1/Lab1.srscs/sources\_1/imports/src/SweRVolfSoC/SweRVeh1CoreComplex/include

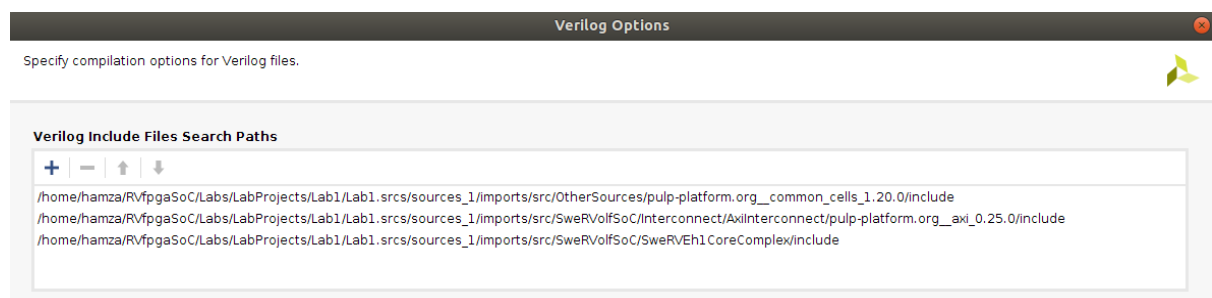


図68 : Verilogのインクルードファイルのパス

OKをクリックします。

#### ステップ4 : Bitstreamタブに移動する

「Bitstream」タブに移動して、「tcl.pre」ボタンをクリックします。

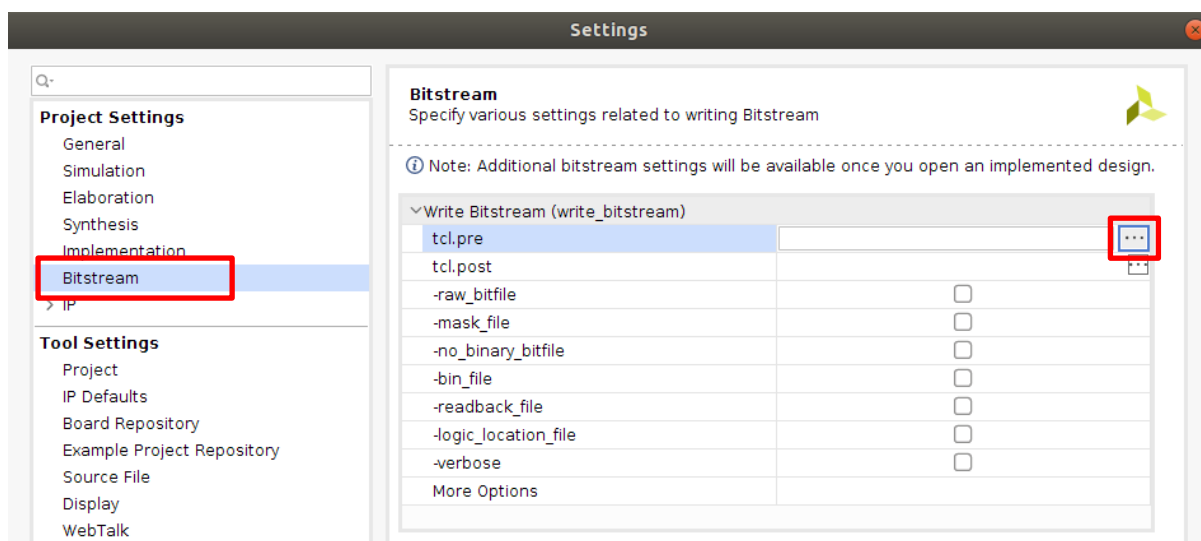


図69 : ビットストリーム設定

「New script」オプションを選択します。

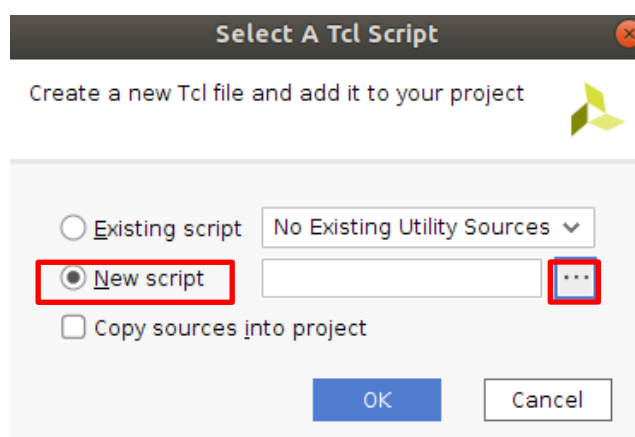


図70 : 新しいTclスクリプト

以下のパスに移動し、「script.tcl」ファイルを選択します（図71を参照）。  
[RVfpgaSoCPath]/RVfpgaSoC/Labs/LabResources/Lab1/script.tcl

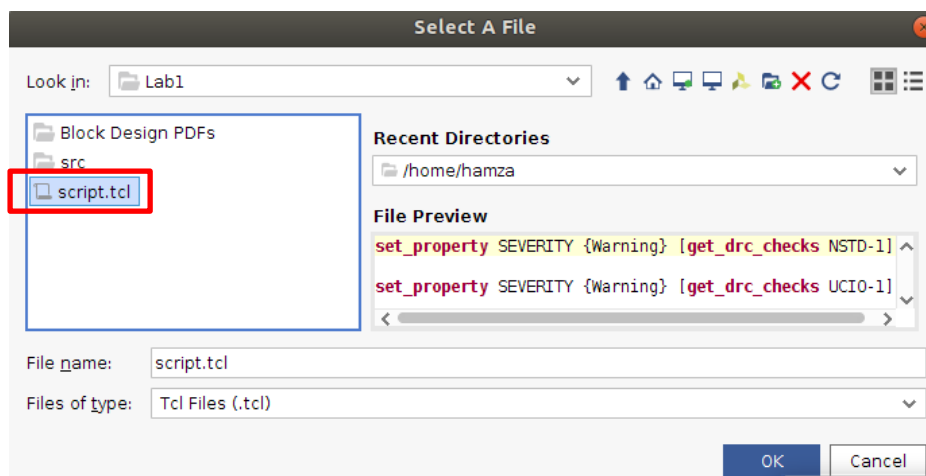


図72 : 「script.tcl」ファイルをインポート

OKをクリックして、変更を適用します。

#### ステップ4：ビットストリームを生成する

図73に示されているように、Flow → Generate Bitstreamをクリックします。

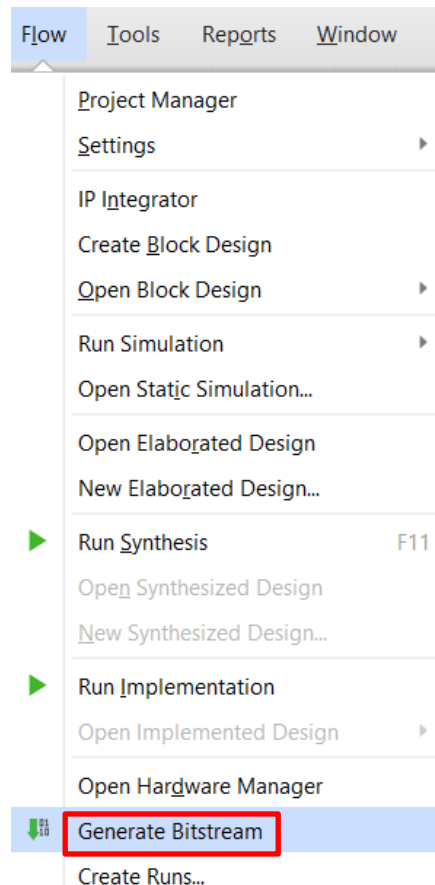


図73：ビットストリームの生成

ウィンドウがポップアップ表示されて、使用可能な実装結果がないことをが示され、合成および実装を起動するように求められます。

Yesをクリックします（図74を参照）。

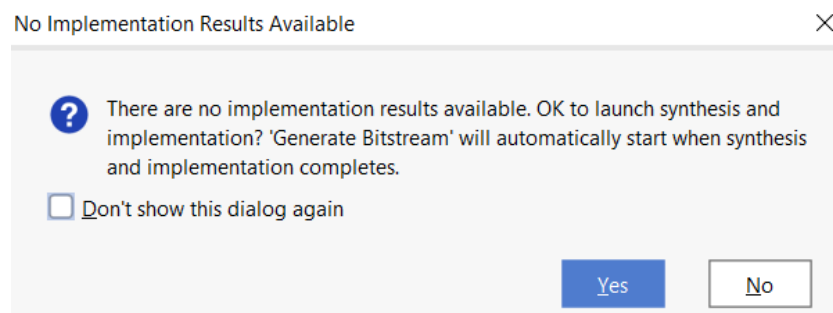


図74：合成および実装のウィンドウを起動

**Launch Runs**ウィンドウが画面にポップアップ表示されます（図75を参照）。OKをクリックします。



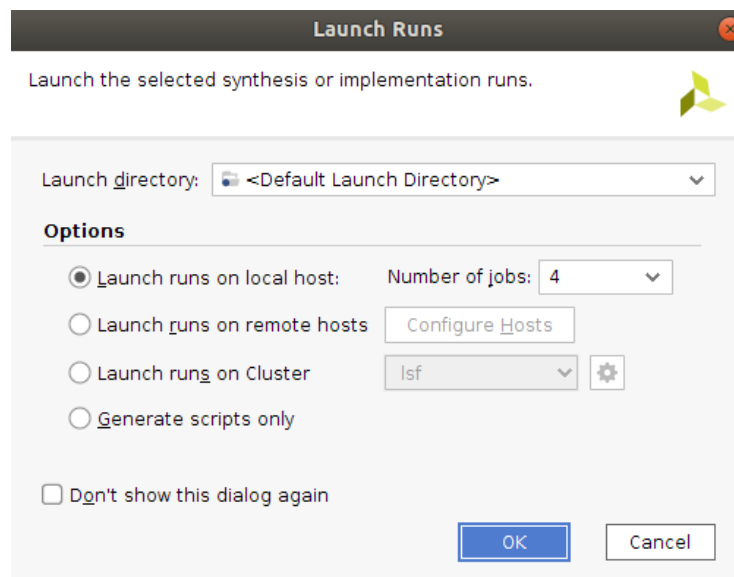


図75 : 実行を起動

未接続のままのピンが自動的に「0」に接続されることについて通知する警告のリストが、表示されます。OKをクリックします（図76を参照）。

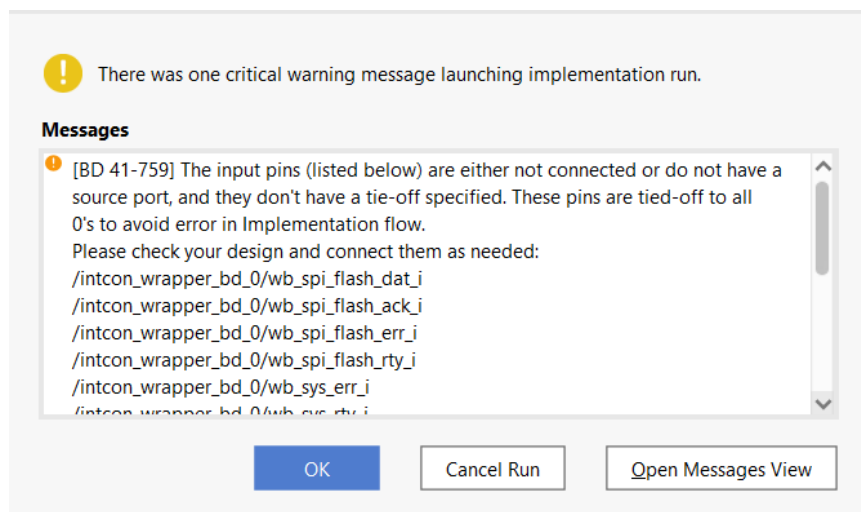


図76 : 実行警告メッセージを起動

このステップにより、**RVfpgaNexys**が合成され（このプロジェクトでVerilogおよびSystemVerilogファイルによって定義されるように）、FPGAにマッピングされ、ビットストリームが作成されます。

| 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 | Start           | Elapsed  |
| synth_1 (active)                               | constrs_1    | Queued...               |     |     |     |     |      |             |               |     |    |      |      |     |                 | 00:00:00 |
| impl_1   | constrs_1    | Queued...               |     |     |     |     |      |             |               |     |    |      |      |     |                 | 00:00:00 |
| Out-of-Context Module Runs                     |              |                         |     |     |     |     |      |             |               |     |    |      |      |     |                 |          |
| BD   |              | Running Submodule Runs  |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_7                                   | BD_bidirec_7 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_5                                   | BD_bidirec_5 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_8                                   | BD_bidirec_8 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_4                                   | BD_bidirec_4 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_2                                   | BD_bidirec_2 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_3                                   | BD_bidirec_3 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_9                                   | BD_bidirec_9 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_5                                   | BD_bidirec_5 | Running synth_design... |     |     |     |     |      |             |               |     |    |      |      |     | 3/3/21, 2:55 PM | 00:01:50 |
| BD_bidirec_7                                   | BD_bidirec_7 | Queued...               |     |     |     |     |      |             |               |     |    |      |      |     |                 | 00:00:00 |
| BD_bidirec_0                                   | BD_bidirec_0 | Queued...               |     |     |     |     |      |             |               |     |    |      |      |     |                 | 00:00:00 |
| BD_bidirec_3                                   | BD_bidirec_3 | Queued...               |     |     |     |     |      |             |               |     |    |      |      |     |                 | 00:00:00 |
| BD_bidirec_2                                   | BD_bidirec_2 | Queued...               |     |     |     |     |      |             |               |     |    |      |      |     |                 | 00:00:00 |

図77：実行を設計

**注意：** 以下のようなエラーが発生した場合、Gtk-Message: Failed to load module "canberra-gtk-module"

以下のコマンドによってパッケージをインストールして、問題を解決します。

```
sudo apt install libcanberra-gtk-module libcanberra-gtk3-module
```

VMを使用している場合、合成中に、RAMの低割り当てにより、Vivadoがクラッシュすることがあります。Vivadoがクラッシュする場合は、より多くのRAMをVMに割り当てることを、推奨します。

使用するコンピュータの速度によって異なりますが、このプロセスには数分かかることがあります。

| 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 | Start           | Elapsed  |
| synth_1 (active)                               | constrs_1   | synth_design Complete!    |       |       |       |       |       |             |               | 3368  | 3331  | 13.0 | 0    | 0   | 5/4/21, 3:22 PM | 00:02:13 |
| impl_1   | constrs_1   | write_bitstream Complete! | 0.327 | 0.000 | 0.050 | 0.000 | 0.000 | 0.934       | 0             | 33637 | 18546 | 44.0 | 0    | 4   | 5/4/21, 3:25 PM | 00:12:32 |
| Out-of-Context Module Runs                     |             |                           |       |       |       |       |       |             |               |       |       |      |      |     |                 |          |
| BD   |             | Submodule Runs Complete   |       |       |       |       |       |             |               |       |       |      |      |     | 5/4/21, 3:03 PM | 00:19:17 |

図78：Verilogのインクルードファイルのパス

ビットストリームが生成されると、図79に示されるように、ウィンドウがポップアップ表示されます。右上隅のXボタンをクリックして、ウィンドウを閉じます。

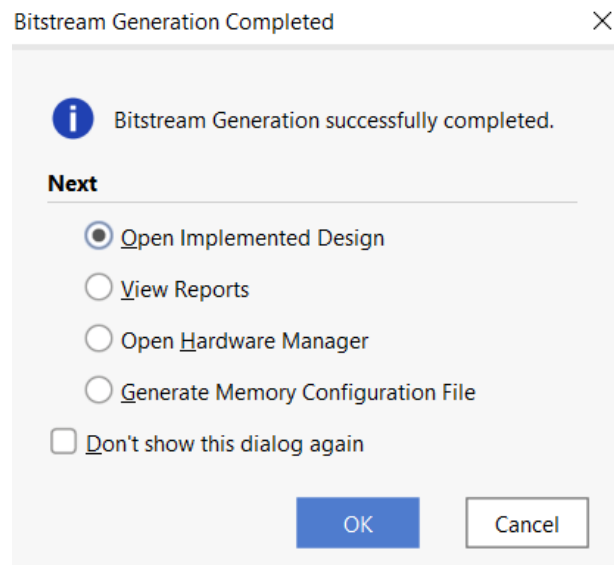


図79 : ビットストリーム生成完了

ビットストリームの作成が完了しました。次のラボでは、このビットストリームを PlatformIO を介して Nexys A7 にアップロードする方法を示し、このラボで構築したばかりの SweRVolfX サブセットでプログラム例を実行する方法を、示します。