

# 演習 1: Vitis ソフトウェア開発ツールの実行

2019.2

## 概要

この演習では、Vitis™ IDE の基本動作について説明します。プロジェクトの作成、アプリケーションへの既存のソースコードの追加、エラーの管理、コンパイルとリンク、ダウンロードなどの概念について説明します。ソフトウェア開発フローの個々の機能については、ほかの演習で詳しく説明します。

## 目的

この演習では次の各項を学習します:

- ワークスペースの作成
- アプリケーション開発に必要な基本的なプロジェクトの作成
- アプリケーションソースコードのナビゲート
- ELF (実行可能なロードフォーマット) ファイルの作成とハードウェアプラットフォームへのダウンロード

## はじめに

Vitis IDE は、強力で複雑なツールです。基本的な機能をナビゲートする方法を知ることによって、ソフトウェア開発がずっと簡単になります。

習得する必要がある最初のコンセプトはワークスペースです。ワークスペースは、すべてのプロジェクト（それぞれが独自のサブディレクトリにある）と、ツールにユーザー設定とプロジェクトの関係を提供する特殊なファイルを含むディレクトリです。通常、ワークスペースには関連プロジェクトが含まれていますが、関連のないプロジェクトも同じワークスペースに保持される場合があります。複数のワークスペースがある場合、ユーザーは既にあるワークスペースを選択するか、新しいワークスペースを指定できます。識別されると、ツールが起動します。

次はプロジェクトの一般的な概念です。プロジェクトとは、特定の種類の事柄について特定の結果を生成するために関連ファイルを集めたものです。この演習で紹介するプロジェクトは、アプリケーション開発で最も一般的なプロジェクトです。より専門的なプロジェクトがありますが、それについてはほかのトピックで説明しています。

プロジェクトを詳しく見ると、アプリケーション、プラットフォーム、システムプロジェクトがあります。「最も低い」レベルは、ハードウェアとブートコンポーネントの定義を含むプラットフォームプロジェクトです。プラットフォームプロジェクトにはドメインがあります。ドメインは、BSP またはオペレーティングシステムのコンテナです。アプリケーションプロジェクトは一つのドメインに基づいており、ソースコードファイルのコレクションが含まれています。ソースコードは、一緒にコンパイルされ、単一の実行可能でリンク可能なフォーマットまたは ELF ファイルを作成します。これらのアプリケーションプロジェクトはシステムプロジェクトにグループ化され、すべてのプロセッサにまたがるプラットフォームで同時に実行されるアプリケーションを定義します。ワークスペース内には複数のシステムプロジェクトが存在できます。ただし、一度にアクティブにできるのは 1 つだけです。

## 演習 1: Vitis ソフトウェア開発ツールの実行

次の図は、アプリケーション開発に必要なプロジェクト間の関係を示しています:

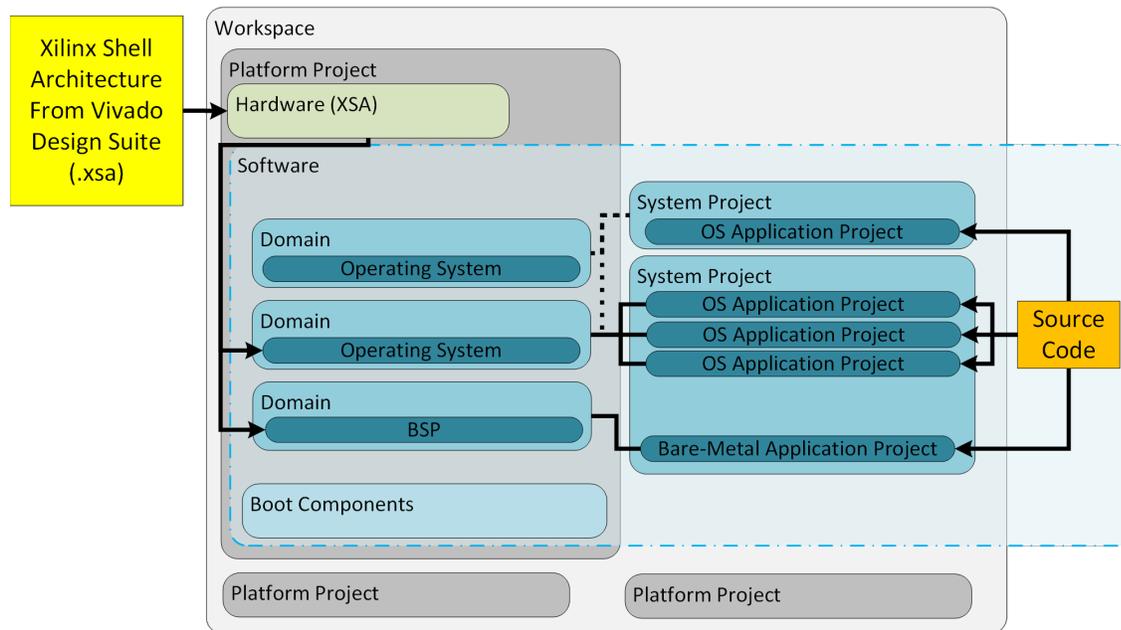


図 1-1: ハイレベルプロジェクトの関係

### 概要:

まず、新しいワークスペースを作成し、そのワークスペース内にプラットフォームプロジェクトを作成します。プラットフォームプロジェクトは、ハードウェア情報、ドメイン、およびハードウェアの適切なブートソフトウェアのコンテナであり、FSBL およびプラットフォーム管理ソフトウェアが含まれる場合があります。ブートソフトウェアに関する詳細は、FSBL および PMU のトピックで説明しています。

まず、Vivado® Design Suite からザイリンクスシェルアーキテクチャ (XSA) フォーマットファイルとしてエクスポートされたハードウェア機能の定義をインポートします。これは、プラットフォームプロジェクトのハードウェアの側面として機能します。ハードウェアのブート要素は自動的に作成されます。

プラットフォームプロジェクトの作成中、ドメインを作成します。通常、ドメインはシングルプロセッサをターゲットとし、ハードウェア記述に基づいてその特定のコアに BSP を提供します。BSP だけが選択肢ではありません。ユーザーは、ドメインにオペレーティングシステムを取り込むことを選択できます。

オペレーティングシステムが実装されると、プロセッサのグループをクラスター化してそのクラスターで OS を実行するなど、より多くのオプションが利用可能になります。さらに、オペレーティングシステムにはスケジューラが含まれているため、物理プロセッサの数よりも多くのアプリケーションプロジェクトを実行できます。BSP を使用するドメインは、ドメインごとに 1 つのアプリケーションプロジェクトに制限されます。

3 つのプロジェクトを作成して、アプリケーションプロジェクトに提供されたソースコードを追加し、コンパイル、リンク、ロードにより ELF ファイルを生成します。ただし、提供されたソースコードにはいくつかのエラーが含まれているので、修正して再コンパイルが必要です。エラーが修正されると、結果の ELF ファイルをボードにダウンロードして実行できます。

### 演習環境の理解

このコースで提供される演習とデモは、提供された Linux プラットフォームで実行するように設計されています。提供されている Linux プラットフォームには、カスタマイズされた演習環境があります。これらのカスタマイズは、学習の機会を簡素化および強化するために作られています。多くの演習とデモは、Windows 環境でも実行できます。PetaLinux、Yocto、QEMU などのトピックは、Windows でサポートされていません。

## 演習 1: Vitis ソフトウェア開発ツールの実行

この演習の手順は Linux 用の表記を使用しており、階層セパレーターとしてスラッシュ「/」が使われています。Windows ユーザーの場合、階層セパレーターとして「¥」を使用してください。

Windows ユーザーは、Oracle の VirtualBox ツールを使用して Linux を実行できます。このツールをダウンロードしてインストールする方法の詳細について、演習セットアップガイドを参照してください。ザイリンクスカスタマーレーニンググループは、事前に構成された仮想マシンイメージを提供し、演習の実行に必要なすべてのツールが含まれています。一部のツールと IP を使用するには、ライセンスが必要です。

演習セットアップガイドで説明されているいくつかの環境変数は、環境を使いやすいようにカスタマイズする方法についても説明しています。多くの環境変数が定義されています。

これらの環境変数は次のとおりです。

環境変数名	説明
XILINX_PATH	これは、ザイリンクスツールのインストールディレクトリを指定します。Vivado Design Suite などのツールはここにあります。
VITIS_PATH	これは、Vitis ツールのインストールディレクトリを指定します。
PETALINUX_PATH	これは、Xilinx PetaLinux ツールのインストールディレクトリを指定します。通常、これは XILINX_PATH 内にありますが、ほかの場所にインストールされている可能性もあります。
TRAINING_PATH	受講者が演習を実行するために割り当てられたディレクトリを指定します。このディレクトリには、演習とデモの事前構築されたイメージと始点が含まれています。
DrivingVitis	これは、TRAINING_PATH ディレクトリ内の特定のディレクトリを指定します。この演習では、\$DrivingVitis を使用してこの演習のディレクトリを示します。これは、\$TRAINING_PATH/DrivingVitis と同等です。

ここで使用される環境変数の表記は、さまざまな環境で常に機能するとは限りません。どの形式を使用できるかについて、説明します。

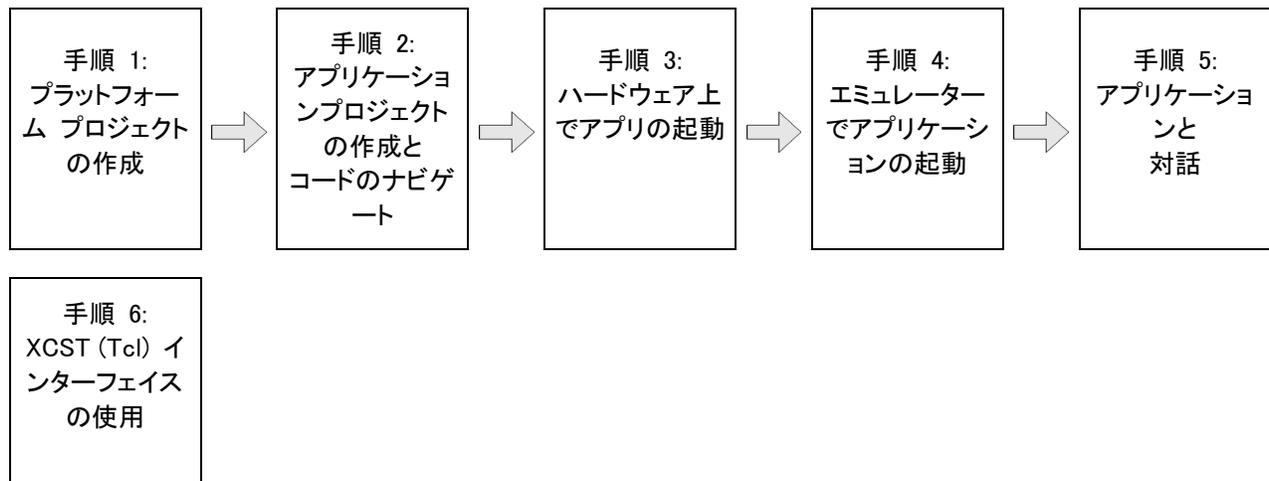
Linux ターミナルウィンドウやシェルスクリプトで作業している時、\$variableName を直接使用できます。Tcl 環境 (Tcl シェル、Tcl スクリプト、またはツール GUI を使用する場合) では、ユーザーは \$::env(variableName) の形式を使用する必要があります。これらの変数は、Vivado Design Suite または Vitis IDE GUI でサポートされていません。これらの状況では、\$variableName を置き換える必要があります。つまり、変数の値を使用します。

変数の値の検索方法: ターミナルウィンドウを開き、echo \$variableName と入力します。variableName の値が表示されます。

この演習で提供される手順は、Linux 環境専用です。Windows システムで作業している場合、演習のセットアップガイドを参照して、Oracle VirtualBox アプリケーションを使用して、ザイリンクスカスタマーレーニングの事前構成された Ubuntu Linux 仮想マシンを実行する方法を確認してください。

## 演習 1: Vitis ソフトウェア開発ツールの実行

## 手順



## Vitis IDE の起動とプラットフォームプロジェクトの作成

## 手順1

この演習は、Vitis IDE を開き、プロジェクトを保存するワークスペースを指定することから開始します。次に、ハードウェア仕様（使用可能なプロセッサ、ペリフェラル、メモリなどを記述する）を使用して、プラットフォームプロジェクトを構築します。このハードウェア仕様は、Vivado Design Suite からエクスポートされたものです。

## 1-1. Vitis IDE を起動しワークスペースを設定します。

1-1-1. タスクバーから Vitis IDE (🔴) アイコンをクリックして、ツールを起動します。

Workspace Launcher が開きます。

Vitis IDE は当初、ツールの設定を追跡してツールのログ ファイルを維持するだけの薄い構造を含むワークスペースを作成します。プロジェクトが追加されるにつれ、このワークスペースは更新されて、すべてのタイプのプロジェクトを含むようになります。[File] → [Switch Workspace] を選択すると、ワークスペースがツール内で切り替えられます。

ソフトウェア アプリケーションを別の場所または別のコンピュータに移動しなければならない場合は、インポートとエクスポート機能を使用します。ワークスペース ファイルは絶対パス名を使うように設定されており、手動でファイルをコピーするとツールが不安定になることがあるため、お勧めできません。

**メモ:** Vitis IDE はターミナルウィンドウから起動できますが、事前に環境を構成するスクリプトを実行する必要があります。<Ctrl + Alt + T> をクリックするか、ツールバーのアイコンをクリックして Linux ターミナルを開き、`source $VITIS_PATH/settings64.sh` と入力します。スクリプトの実行が完了後、`vitis` と入力してツールを起動できます。

1-1-2. [Workspace] に `$DrivingVitis/lab` を入力するか、[Browse] ボタンを使用してロケーションを参照します (1)。

**メモ:** Eclipse ベースのツールは環境変数の拡張をサポートしていないため、この拡張を自分で実行する必要があります。`$DrivingVitis` を `DrivingVitis/lab` トレーニングディレクトリへのパスで置き換える必要があります。

トレーニングディレクトリへのパスがわからない場合、Linux ターミナルを開き (<Ctrl + Alt + T>)、`echo $TRAINING_PATH` と入力します。提供されている Ubuntu VM のデフォルトパスは `/home/xilinx/training` です。ただし、マシンにより構成が異なっている可能性があります。

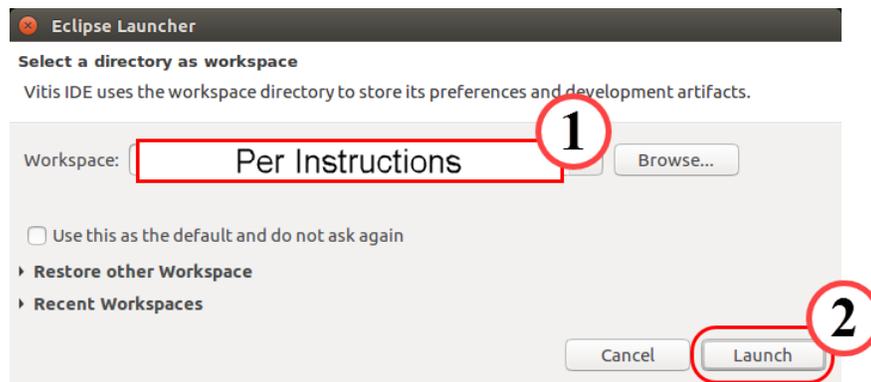


図 1-2: ワークスペースの選択

1-1-3. [Launch] をクリックして新しいワークスペースを開きます (2)。

1-1-4. [Welcome] タブが表示されたら閉じます。

プロジェクトが表示されます。多くの機能を表示するためにウィンドウを最大化するとよいでしょう。

アプリケーションの作業を行う前に、ハードウェア デザインに基づいたプラットフォームプロジェクトを定義する必要があります。このステップは、[Create Application Project] ウィザードの一部として行う場合もありますが、この手順では単独で実行します。

プラットフォームプロジェクトには、ハードウェアデザインの完全な説明が含まれます。たとえば、存在するプロセッサのタイプ、ファブリック内のアクティブなペリフェラル、SoC ベースのシステムの専用シリコン部分、完全なシステムメモリマップなどです。この記述に基づいて、ボードサポート パッケージ (BSP) などのソフトウェアとアプリケーションを、ハードウェアに合わせてカスタマイズできます。

多くの場合、独自のハードウェア記述ファイル (通常は Vivado Design Suite によって生成される) を使用します。ただし、サポートされている SoC デバイスの標準ペリフェラルセットを使用してコードを早期にテストする場合があります。ザイリンクスは、多数のザイリンクスおよびほかのベンダーボード用の事前に定義されたハードウェアテンプレートを提供しています。

次の手順は、カスタムハードウェアを含める方法を示しています:

## 1-2. プラットフォームプロジェクトを作成します。

1-2-1. [Create New] アイコン (  ) の下向きの矢印アイコン (  ) をクリックします (1)。

使用可能なウィザードを示すポップアップウィンドウが表示されます。

演習 1: Vitis ソフトウェア開発ツールの実行

1-2-2. [Platform Project] を選択します (2)。

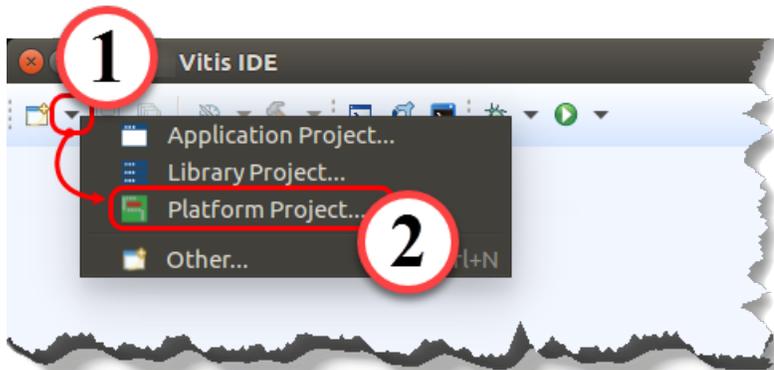


図 1-3: 新しいプラットフォームプロジェクトウィザードを開く

[New Platform Project] ダイアログ ボックスが表示されます。

**メモ:** このウィザードにアクセスする方法は他にもあります。ただし、これが最も簡単です。

1-2-3. [Project name] フィールドに `drvVitis_plat` と入力します (1)。

これは、プラットフォームプロジェクトの名前を定義します。

通常、[Use default location] オプションは選択されたままなので、このプロジェクトはワークスペース内に保持されます (2)。

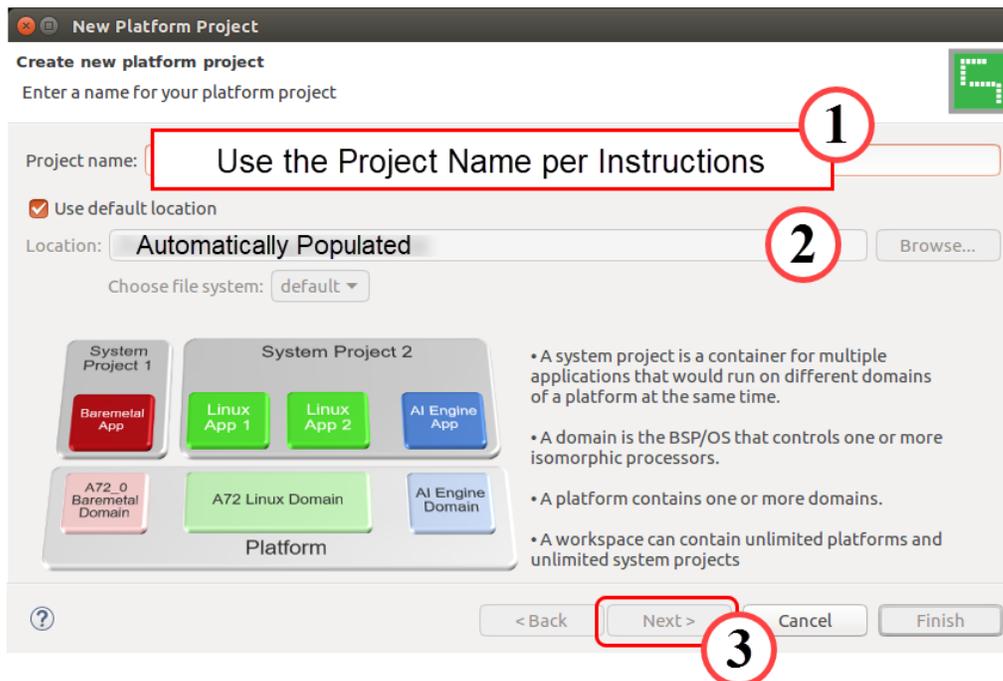


図 1-4: プラットフォームプロジェクト名の入力

1-2-4. [Next] をクリックして続行します (3)。

ここで、既存のプラットフォームに基づいて、またはハードウェア仕様 (Vivado Design Suite で生成) からプラットフォームプロジェクトを作成するように求められます。この一連の指示は、Vitis IDE に XSA ファイルをインポートする方法を示しています。

- 1-2-5. [Create from hardware specification (XSA)] オプションが選択されていることを確認します (1)。

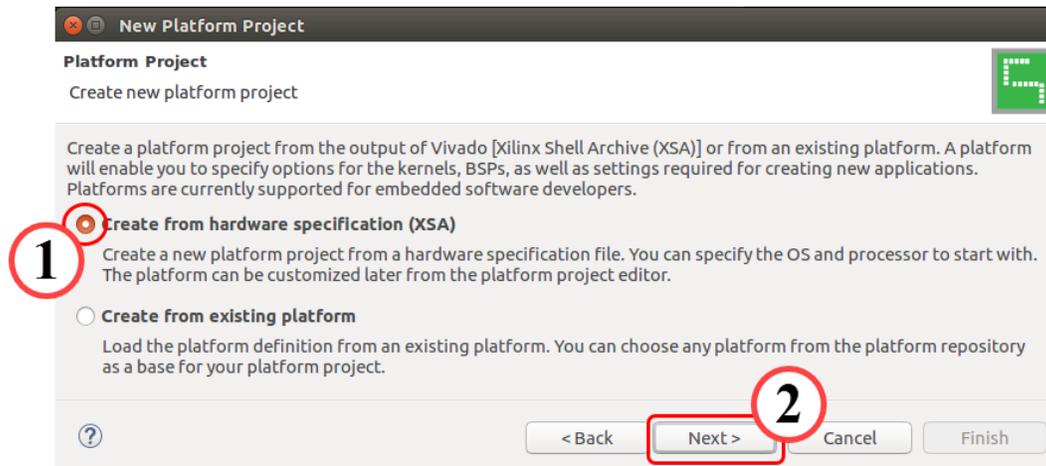


図 1-5: ハードウェア仕様から新しいプラットフォームプロジェクトの作成

- 1-2-6. [Next] をクリックして、XSA ファイルの場所の指定に進みます。

- 1-2-7. XSA ファイルに `$CustEdIP/UED_zcu104.xsa` と入力します (1)。

これを手動で入力するか、[Browse] ボタンを使用して `$CustEdIP` でファイルを検索できます。

Vitis IDE は環境変数を自動的に展開しないことに注意します。入力するか、パスに移動する必要があります。

この環境変数の値がわからない場合、Linux ターミナルウィンドウを開いて `echo $<variable name>` と入力すると、完全なパスが表示されます。

- 1-2-8. まだ選択していない場合、[Operating system] ドロップダウンリストから `standalone` を選択します (2)。

- 1-2-9. 自動的に入力されない場合、[Processor] ドロップダウンリストから `psu_cortexa53_0` を選択します (3)。

- 1-2-10. 特定のデバイスの選択をサポートするために、Vitis IDE に第 1 段階のブートローダー、プラットフォーム管理ファームウェア、およびその他の重要な低レベルソフトウェアを生成するよう指示する、[Generate boot components] オプションが選択されていることを確認します (4)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

プラットフォームプロジェクトと共に、ウィザードは選択したオペレーティングシステムとプロセッサに基づいて、プラットフォームプロジェクト内にドメインを作成します。

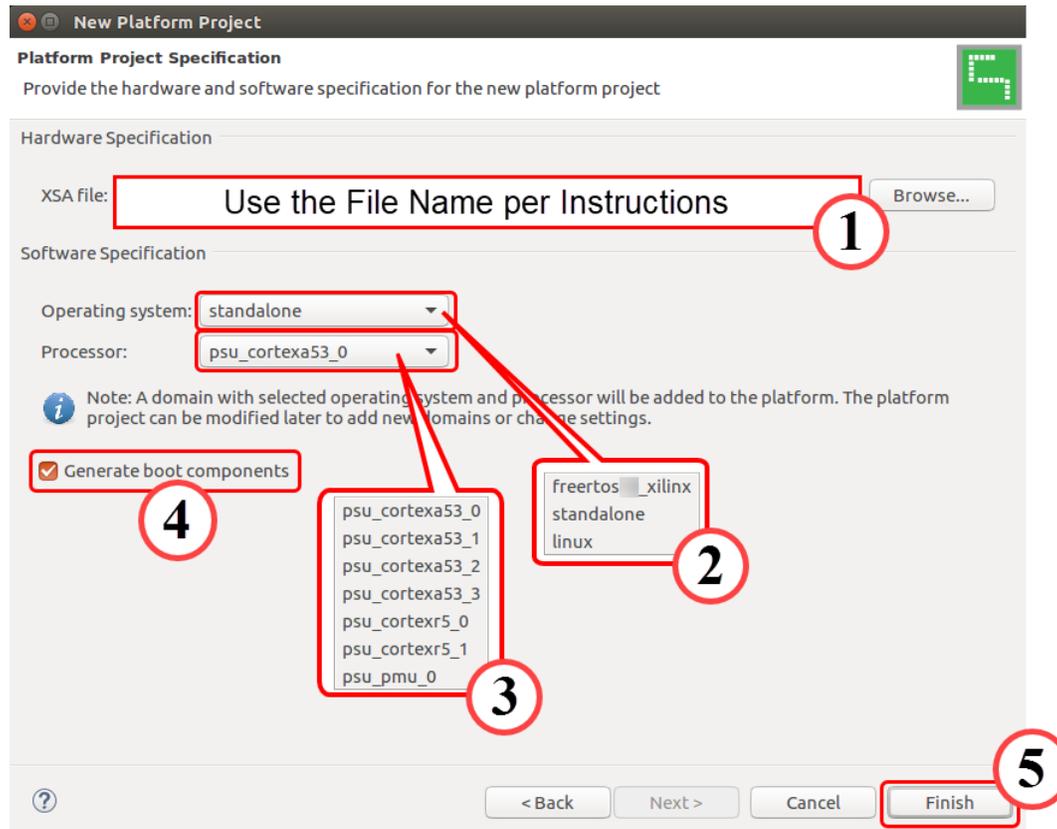


図 1-6: プラットフォームプロジェクトの XSA ファイル、OS、およびターゲットプロセッサの指定

1-2-11. [Finish] をクリックして新しいプラットフォームプロジェクトを作成します (5)。

プラットフォームプロジェクトの作成後、アプリケーションを作成します。

## アプリケーションプロジェクトの作成とコードのナビゲート

## 手順2

ツールを開いてプラットフォームプロジェクトが作成されると、デフォルトの設定では新しいアプリケーションの作成の一環として、ボードサポートパッケージ (BSP) を作成します。BSP はプラットフォームプロジェクトに含まれるドメインの一部になり、アプリケーションはシステムプロジェクトの一部になります。

アプリケーションプロジェクトの作成中に、多数のテンプレートが利用できますが、[Empty Project] テンプレートを使用し、提供されたソースコードファイルをインポートします。

[Application Project] ウィザードを使用すると、システムプロジェクトのアプリケーションを早く作成し、ドメインプロジェクトに関連付けることができます。

ダイアログ ボックスの選択に応じて、前処理、コンパイル、組み立て、リンクに対して適切なツールチェーンが選択されます。

## 2-1. psu\_cortexa53\_0 対象にした新しいスタンドアロンアプリケーションを作成します。

2-1-1. [Create New] アイコンの横にある矢印 (▼) をクリックします (1)。

使用可能なウィザードを示すポップアップウィンドウが表示されます。

2-1-2. [Application Project] を選択して、[Application Project] ウィザードを起動します (2)。

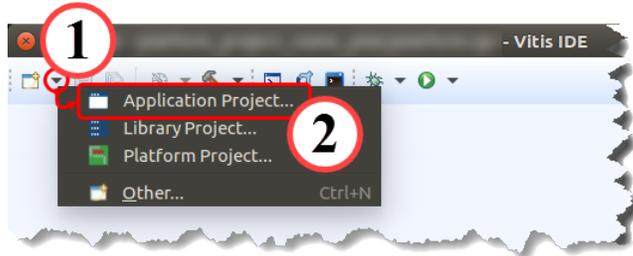


図 1-7: [Application Project] ウィザードを開く

**メモ:** このウィザードにアクセスする方法は他にもあります。ただし、これが最も簡単です。

2-1-3. [Project name] フィールドに `drvVitis_app` と入力します (1)。

これは、アプリケーションプロジェクトの名前を定義します。「app」サフィックスを使用する必要はありませんが、これを使用するとプロジェクトタイプの識別が簡単になります。

**メモ:** PetaLinux ツールで使用されるアプリケーションを構築する場合、名前にアンダースコアを使用しないでください。PetaLinux はそれを正しく処理しません。

[Use default location] オプションは選択されたままにします。このプロジェクトはワークスペース内に保持されます (2)。

これらの一連の手順では、新しいシステムプロジェクトの作成について説明しているので、次のタスクに進みます。このアプリケーションを追加できる既存のシステムが存在する場合、ここに追加されます。

2-1-4. [System project] ドロップダウンリストから [Create New] が選択されていることを確認します (3)。

ウィザードは、[System project name] フィールドを自動的に入力します。

2-1-5. システムプロジェクト名を `drvVitis_sys` に変更します (4)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

アプリケーションプロジェクト名の命名と同様に、「sys」サフィックスは必要ありませんが、これを使用するとプロジェクトタイプの識別が簡単になります。

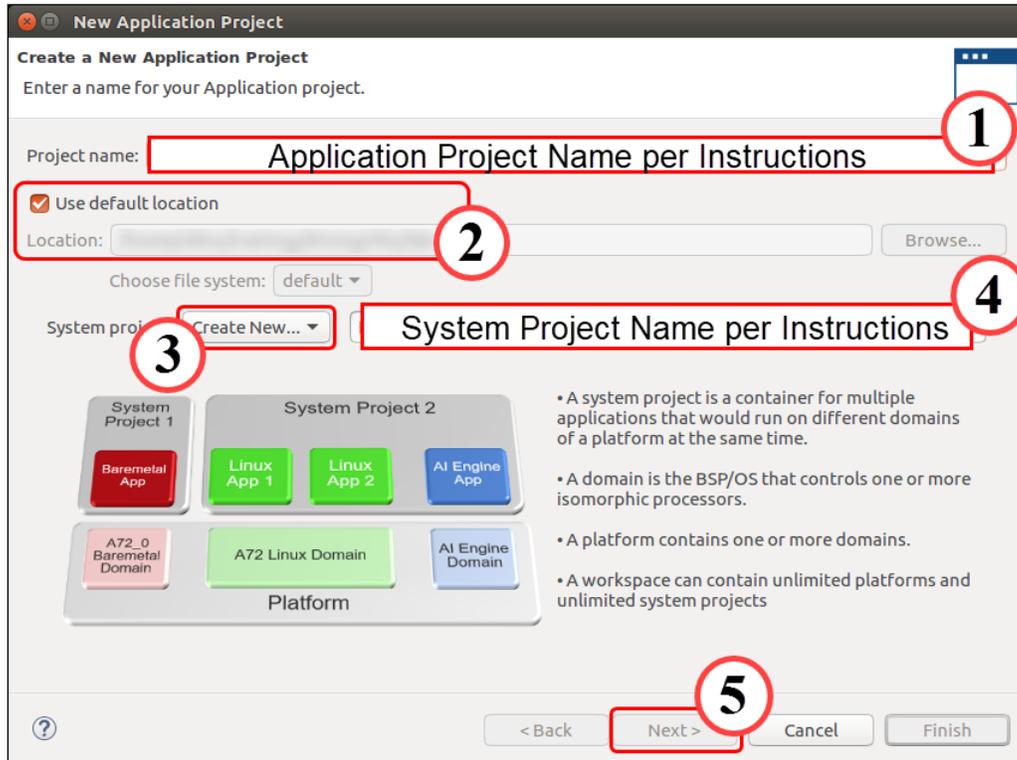


図 1-8: 新しいアプリケーションおよび BSP プロジェクトの作成

2-1-6. [Next] をクリックして、プラットフォーム、システム、およびアプリケーションプロジェクトを関連付けるプラットフォームプロジェクトを指定します (5)。

2-1-7. [Select a platform from repository] タブを選択します (1)。

このツールはいくつかの定義済みプラットフォームが持っており、このリストに新しいカスタムプラットフォームプロジェクトが含まれています。

2-1-8. リストから `drvVitis_plat` を選択します (2)。

**メモ:** ハードウェアプラットフォームをインポートした場合、識別しやすくするために、名前に [custom] が追加されます。

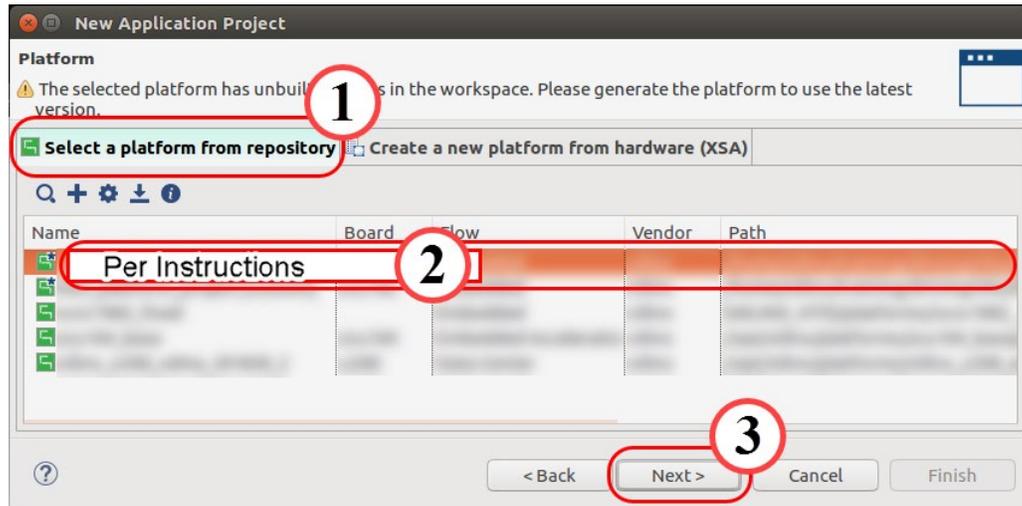


図 1-9: プラットフォームの選択

2-1-9. [Next] をクリックして、このアプリケーションプロジェクトにドメインプロジェクトを関連付けます (3)。

2-1-10. ドメインとして `standalone_domain` を選択します (1)。

2-1-11. 言語として `C` を選択します (2)。

これにより、ツールはこの言語に適切なツールチェーンを使用します。

CPU とオペレーティングシステムは、ウィザードのほかのセクションで構成された値に設定されていることに注意します。

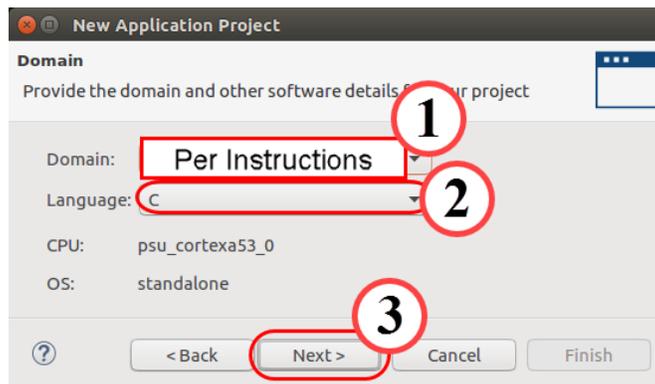


図 1-10: アプリケーションプロジェクト用のドメインと言語の選択

2-1-12. [Next] をクリックしてテンプレートを選択します (3)。

2-1-13. リストから `Empty Application` テンプレートを選択します (1)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

利用可能なテンプレートは、OS の選択とプロセッサのタイプに基づいて異なることに注意します。

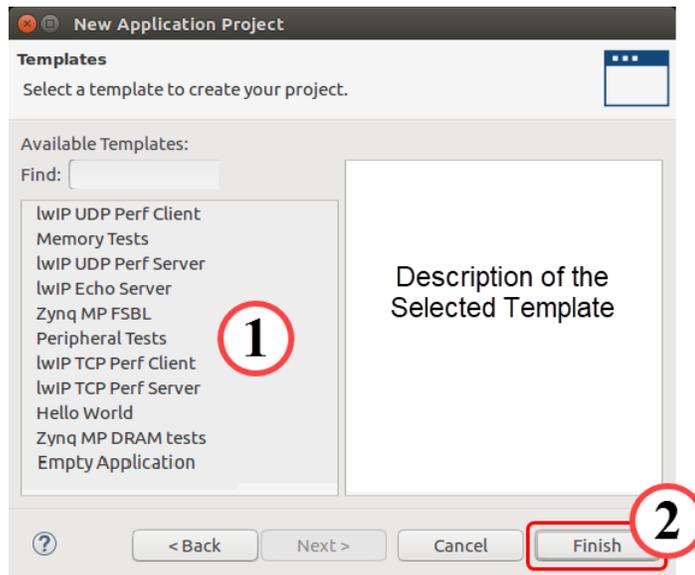


図 1-11: アプリケーション テンプレートの選択

2-1-14. [Finish] をクリックしてアプリケーション プロジェクトを構築します (2)。

**メモ:** これはアプリケーションプロジェクトをコンパイルしません。むしろ、すべてのプロジェクトのピースが組み立てられます。

アプリケーションフレームワークが生成され、提供されたソースファイルがロードされます。

提供されているテンプレートはハードウェアの迅速な検証に役立ちますが、既存のリソースファイル (\*.c、\*.h、\*.cpp、など) をアプリケーションプロジェクトに追加することは一般的な方法です。この操作はインポート機能として実行されます。

2-2. `drivingVitis_main.c`, `hardware_support.c`, `hardware_support.h`, `LED_drivers.c`, `LED_drivers.h`, `platform.c`, `platform.h`, `platform_config.h`, `serialPort_helper.c`, `serialPort_helper.h`, `utils_print.c`, and `utils_print.h` をアプリケーションプロジェクトに追加します。

ソースのインポートで推奨される方法をここに示します。

2-2-1. [Explorer] で、`drvVitis_app` → `src` を展開します。

2-2-2. リソース ファイルを配置する、プロジェクト内の希望のデスティネーション ディレクトリを右クリックします。

一般的には `src` ディレクトリです。

2-2-3. [Import Sources] を選択してインポート ウィザードソースを開きます。

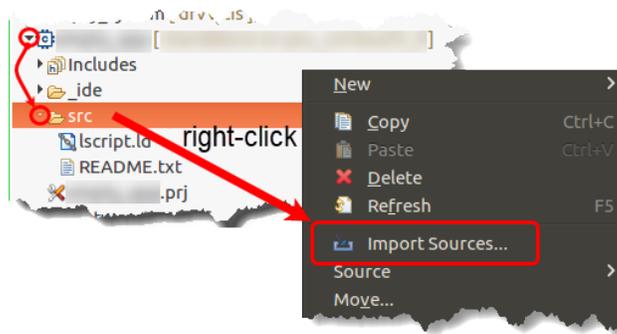


図 1-12: ソースのインポート

[Import Wizard] ダイアログ ボックスが開きます。

2-2-4. ロケーションを入力するか、\$DrivingVitis/support を参照します (1)。

この時点では、ディレクトリを選択しているだけです。次に、実際のファイルを選択します。

**メモ:** 多くのファイルを選択する場合、通常、ディレクトリ全体 (2) を選択してから、不要なファイルを削除する方が簡単です。

2-2-5. **drivingVitis\_main.c, hardware\_support.c, hardware\_support.h, LED\_drivers.c, LED\_drivers.h, platform.c, platform.h, platform\_config.h, serialPort\_helper.c, serialPort\_helper.h, utils\_print.c, and utils\_print.h** をオンにすることによって、これらのファイルを選択します。

必要に応じて、インポートされたソースのロケーションを指定できます。

演習 1: Vitis ソフトウェア開発ツールの実行

2-2-6. [Explorer] からインポートを選択すると、このフィールドは既に入力された状態になります (4)。

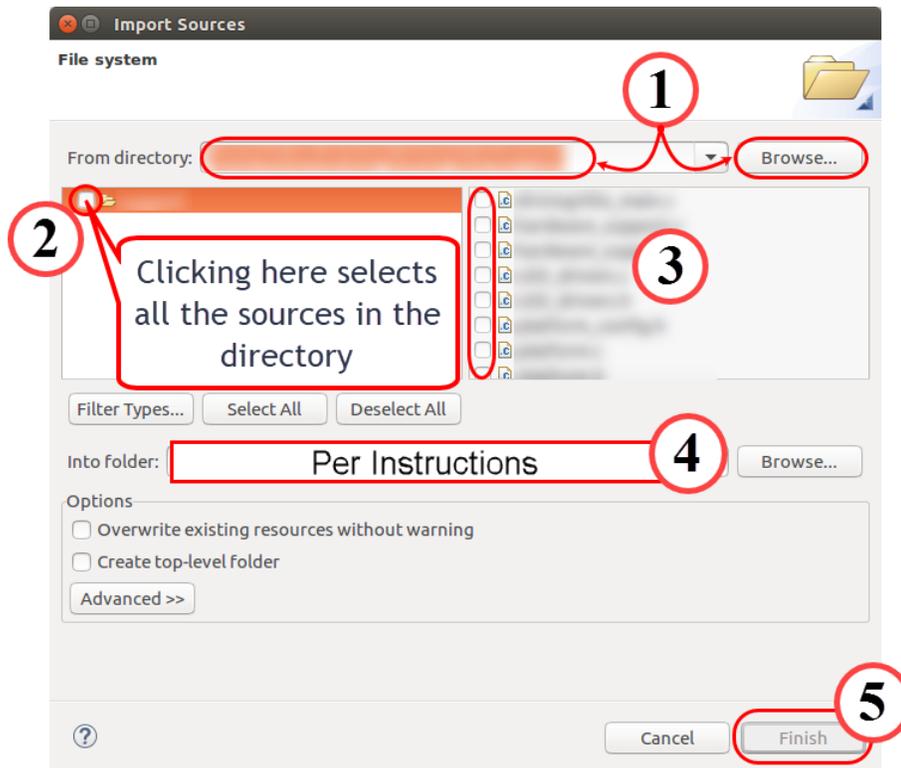


図 1-13: リソースファイルの選択

2-2-7. [Finish] をクリックして、選択したフォルダーにこれらのファイルを追加します (5)。

[Explorer] タブには、このワークスペースにある各種プロジェクトのツリー構造が表示されます。リストの各最上位エントリは、独自のプロジェクトです。プロジェクトのタイプは異なる場合があります。ただし、プロジェクトには、特定のタイプのプロジェクトに必要なすべてのパーツが含まれています。

一般に、エンベデッドアプリケーションの開発には、ハードウェア仕様、必要なブートアプリケーション、およびドメインのコレクションを含むプラットフォームプロジェクトがあります。ドメインプロジェクトは 1 つ以上のプロセッサに関連付けられており、BSP またはオペレーティングシステムの形式でサポートコードが含まれています。

アプリケーションプロジェクトには、ドメインの 1 つでコードを実行するために必要なソースコードと構造が含まれています。

最終プロジェクトはシステムプロジェクトです。1 つのシステムプロジェクトのみがアクティブになります。これには、プラットフォームプロジェクトに定義されているハードウェアで同時に実行される一連のアプリケーションプロジェクトが含まれています。

ワークスペースには、あらゆる種類のプロジェクトが複数存在できます。この演習では、`drvVitis_app` はアプリケーションプロジェクトで、`drvVitis_dom` ドメインにはプラットフォームハードウェアをアプリケーションソフトウェアにブリッジする BSP が含まれ、最後に `drvVitis_plat` プロジェクトにはハードウェアプラットフォームとブートアプリケーションの説明が含まれます。

演習 1: Vitis ソフトウェア開発ツールの実行

ここで、作成された3つのプロジェクトを表示できます。アプリケーションのソースファイルを含む src フォルダを含むさまざまなサブフォルダが表示されるように、アプリケーションプロジェクトを展開します。

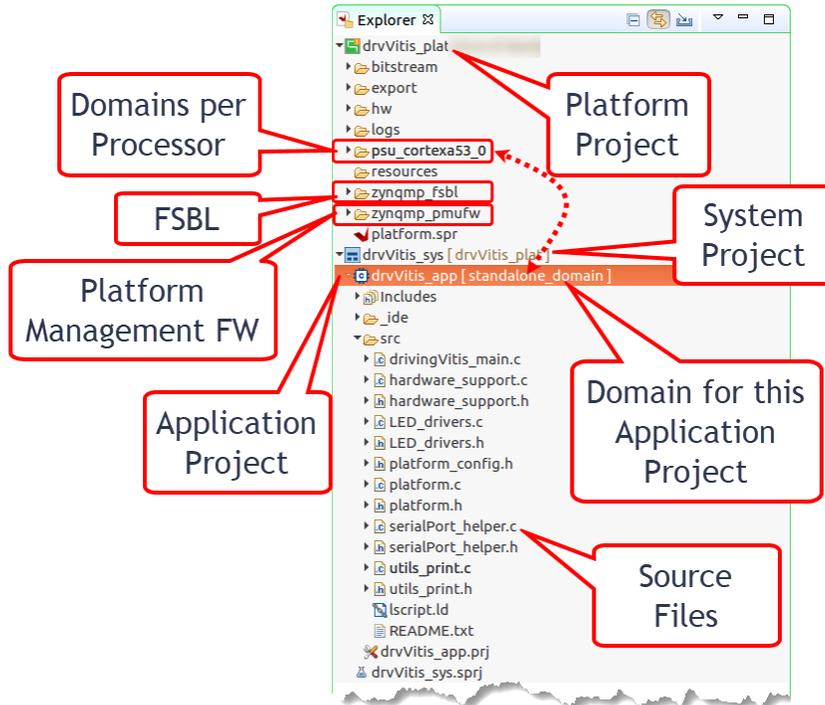


図 1-14: プロジェクトとソースを表示する [Explorer] タブ

現在、Vitis IDE では自動ビルドが有効になっていません。ビルドプロセスを手動で起動する必要があります。

### 2-3. 適切なビルド設定を確認します。

- 2-3-1. [Hammer/Build] アイコン (🔨) の横にある矢印 (▼) をクリックします。
- 2-3-2. **Debug** を選択します。



図 1-15: ビルドのオプション

これにより、今後のすべてのビルド用のデフォルト構成が設定されます。この値はいつでも変更できます。

### 2-4. すべてのプロジェクトを構築します。

- 2-4-1. 任意のプロジェクトを右クリックします (1)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

2-4-2. コンテキストメニューから [Build Project] を選択します (2)。

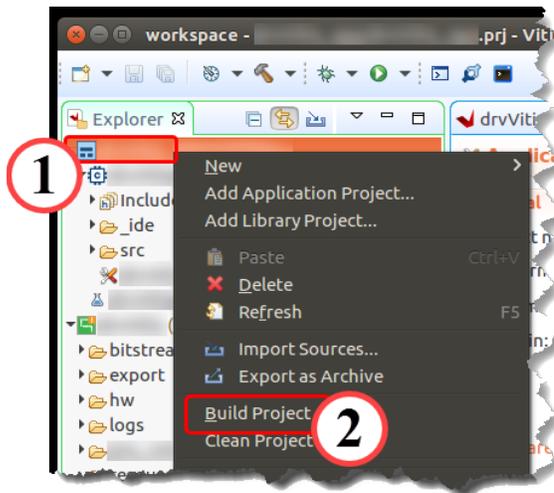


図 1-16: プロジェクトの構築

これにより、ビルドコンフィギュレーションで指定された設定ですべてのプロジェクトが構築されます。

または、[Hammer/Build] アイコン (  ) をクリックします。

インポートしたばかりのソースコードに問題があることに気づくかもしれません。これらのエラーは後で修正します。エラーがどのように呼び出されているかに注目してください。

2-5. このプロジェクトのすべてのソースファイルを表示するように、`drvVitis_app` → `src` を展開します。

2-5-1. `src` の横にある  アイコンまたは記号をクリックして、 アプリケーションプロジェクト内のフィールドを `src` ディレクトリまで展開します。

ヒント: システムプロジェクト、アプリケーションプロジェクト、`src` ディレクトリの順に展開します。

オプションで、`LED_drivers.c` ファイルを展開できます。

それぞれの名前の隣に、ファイルの型を示すアイコンとエラー表示があることに注目します。

演習 1: Vitis ソフトウェア開発ツールの実行

この例では、LED\_drivers.c ファイルにエラーがあることを示しています。このエラーは、src ディレクトリ、アプリケーションディレクトリに通知されます。この方法により、フォルダーが圧縮されている場合でもエラーマーカーが表示されます。さらに、LED\_drivers.c ソースファイルを展開すると、エラーを含む関数が表示されます。

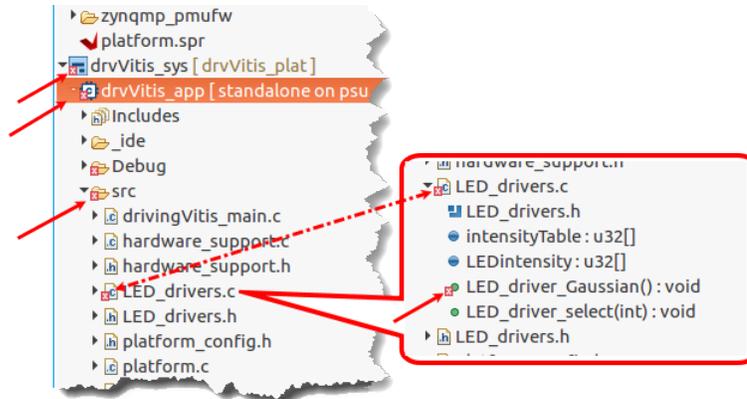


図 1-17: 表示されるエラー

エラーは意図的にソースコードに導入されていて、エラーの特定と修正を練習できます。

## 2-6. 最初のエラーを検索して修正します。

エラーを検索するには、いくつかの方法があります。最初にエラーを検索して修正し、次に別の方法で第 2 のエラーを検索して修正します。

### 2-6-1. [Problems] タブを検索します。

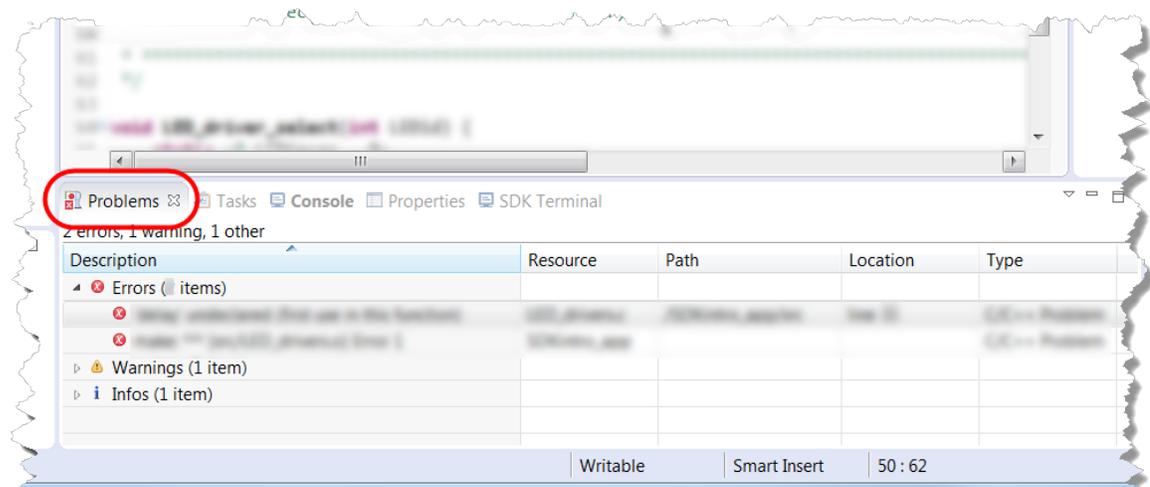


図 1-18: [Problems] タブの位置

このタブは、ワークスペースの下部にあります。

[Errors] エントリを展開する必要があります。

### 2-6-2. 最初のエラー ['delay' undeclared (first use in this function)] をダブルクリックすると、エラーがあるファイルが開きエラーのある行が分かります。

**メモ:** エラーメッセージの横には、問題の特定と修正に役立つヒントがあります。

## 演習 1: Vitis ソフトウェア開発ツールの実行

マージン (🔴 および 🟡) の 2 つの小さなアイコンに注目します。これらは、エラーと警告を示します。

- 2-6-3. 各アイコンの上にマウスを置くと、警告とエラーの内容を説明するポップアップメッセージが表示されます。



図 1-19: 警告とエラー用のポップアップ説明にアクセス可能

**メモ:** この図は複合的に示しています。一度に表示されるエラーまたは警告メッセージは 1 つだけです。

エラーアイコンは、この行に何か問題があることを表示します。[Problems] タブの説明 (‘X’ アイコンの上にマウスを置くと表示される) にエラー原因が示されます。この例では、「delay」が宣言されていないことが原因です。

### 質問 1

この問題を解決するにはどのような手順が必要ですか?

---



---



---

ほかの警告またはエラーメッセージは、エラーまたは警告から伝搬している可能性があります。つまり、警告またはエラーはほかの警告またはエラーと関連し、根本的な原因が修正されると、多くのメッセージが消滅します。

### 質問 2

このエラーを修正する最良の方法は何ですか?

---



---



---

### 質問 3

エディターのデフォルトの設定では行番号が表示されます。必要に応じて、行番号をオフにするにはどうすればよいですか?

---

---

---

警告メッセージは、dely が一度も使用されていないことを示しています。これは警告なので、修正しなくてもコンパイルプロセスを完了できますが、次の行のエラーも考慮すると、dely が入力ミスであり、修正する必要があることが分かります。

2-6-4. エラー行よりも前にある[dely]を[delay]に変更します。

2-6-5. 変更を保存するには、[File] → [Save] (または<Ctrl + S> を押す) をクリックします。

警告メッセージとエラーメッセージが引き続き表示されることに注意します。これは、より一般的な「rebuild-on-save」を使用するのではなく、プロジェクトを手動で構築する必要があるためです。

2-6-6. ツールバーの [Hammer/Build] アイコン (  ) をクリックしてアプリケーションを再構築します。

### 質問 4

ファイルの再コンパイルの結果として何が起こりましたか? ヒント: [Problems]タブを表示します。

---

---

---

2-7. 2 番目のエラーを検索し、修正します。

前の手順で説明した方法と同じやり方もできますが、別の方法でエラー箇所を修正します。どちらの方法も同じくらい効果的です。どちらを使うかは好み입니다。

## 演習 1: Vitis ソフトウェア開発ツールの実行

- 2-7-1. [Explorer] タブで、`drvVitis_app` → `src` を展開します。
- 2-7-2. エラーマーカーでファイルのアイコンを検索します (🔍)。

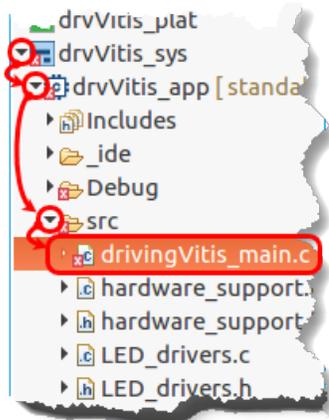


図 1-20: エラーを含むソースファイルの検索

- 2-7-3. ファイル名 (`drivingVitis_main.c`) をダブルクリックし、テキストエディターで開きます。
- 2-7-4. 右マージンを見て、一連の小さな赤と黄色の矩形を確認します。

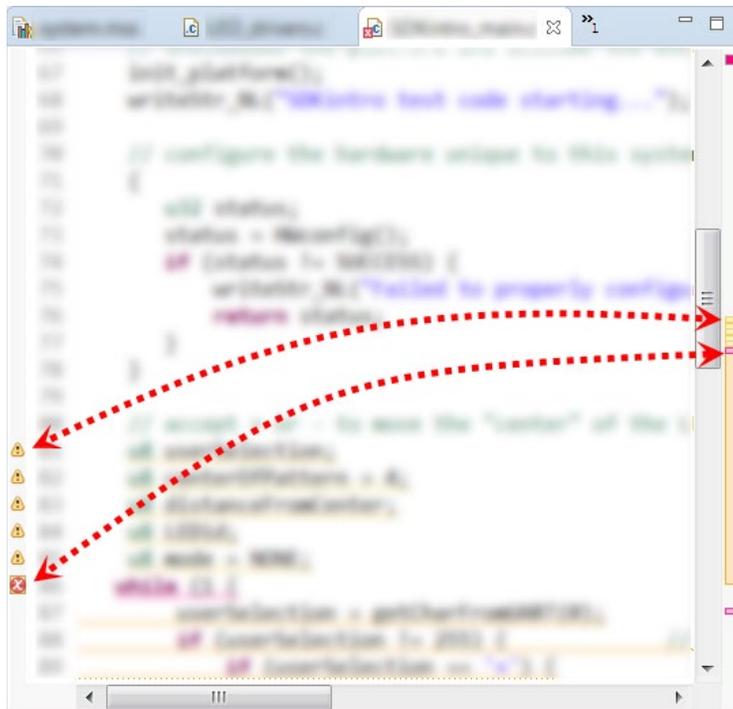


図 1-21: [Error]および[Warning]ナビゲーションバーへの[Error]および[Warning]アイコンのマップ

## 質問 5

これらの小さな赤と黄色の矩形は何を示しますか？

---

---

---

2-7-5. コードの違反行を表示するには、右マージンの一番上の赤い矩形をクリックします。

while ステートメントから右小括弧がない事がすぐに分かります。

2-7-6. 次のように 86 行目に小かっこを挿入します。

```
while (1) {
```

## 質問 6

小かっこを追加するとどうなりますか？

---

---

---

2-7-7. [File] → [Save] (または <Ctrl + S> を押す) をクリックしてコードを保存し、変更を保存します。

2-7-8. ツールバーの [Hammer/Build] アイコン (🔨) をクリックしてアプリケーションを再構築します。

ビルドが成功すると[Problems]タブにエラーが表示されなくなります。[Console]タブの[Build Finished]メッセージの前に、すべてのプログラムセグメントのサイズが表示されます。

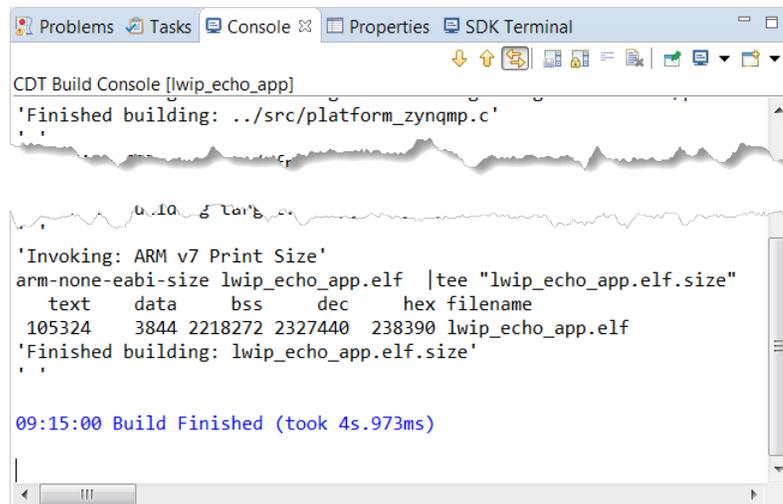


図 1-22: ソフトウェアアプリケーションの構築に成功した場合の表示

## 演習 1: Vitis ソフトウェア開発ツールの実行

[Outline] タブ(ビュー)は、ツールがソースコードをスキャンし、さまざまなエレメントを配置したマップです。次の要素が含まれます:

- #define
- #include
- グローバル変数の宣言
- 関数定義

このリストはフィルタリング可能で、大きくて複雑なソースの検索を簡素化できます。

### 2-8. [Outline] ビューを開きます。

この [Outline] ビューには、開いたファイルの定義、マクロ、関数が表示されます。

2-8-1. [Window] (1) → [Show view] (2) を選択して、開かれているビューのリストを表示します。

2-8-2. [General] エントリを展開します (3)。

2-8-3. [Outline] を選択します (4)。

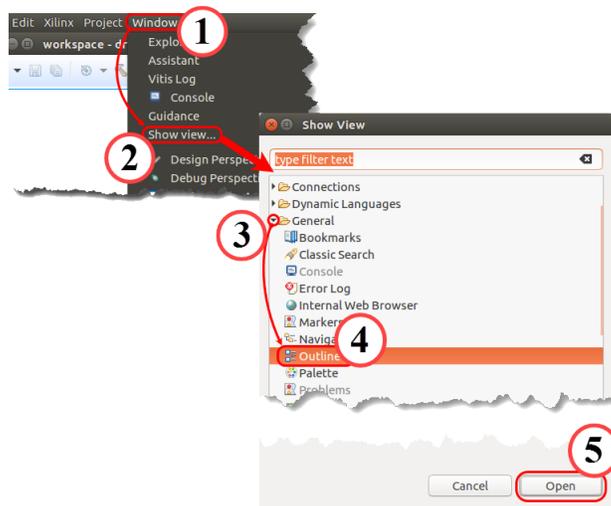


図 1-23: [Outline] ビューを開く

2-8-4. [Open] をクリックして、[Outline] ビューを開きます (5)。

2-9. [Outline] ビューを使用して、以下の問題に答えます。

[Outline] ビューは、アクティブなエディターで使用可能な関数と定数を表示します。

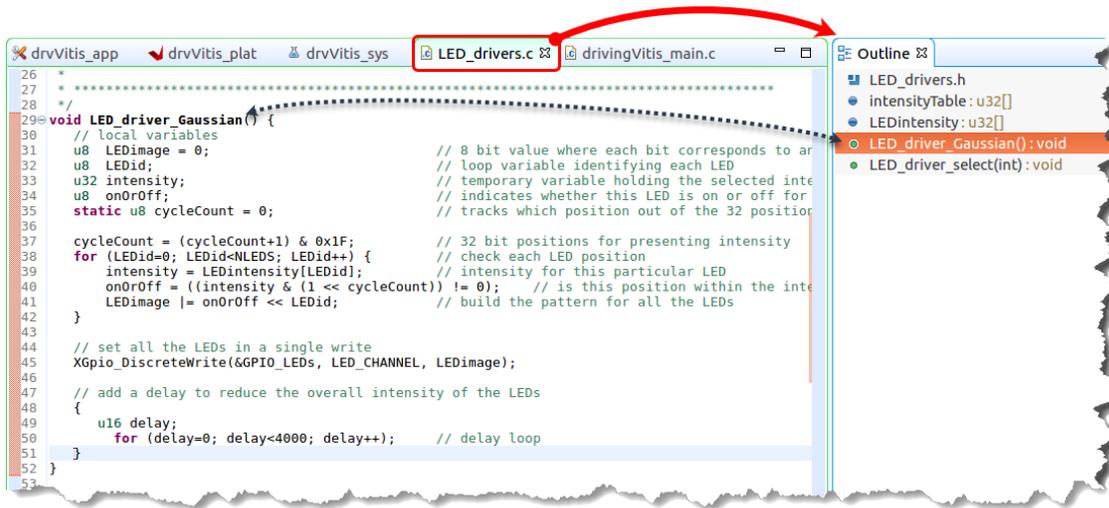


図 1-24: [Outline] ビューへのアクセス

### 質問 7

[Outline] ビューはどのように使用しますか?

---



---



---

### 質問 8

`LED_driver_select(int)` 機能は、何行目から始まりますか? (`LED_drivers.c` にあります)

---



---



---

### 質問 9

別のソースタブをクリックすると、何が起こりますか?

---



---



---

2-9-1. `drivingVitis_main.c` ソースを選択して、アクティブにします。

2-9-2. [Outline] ビューを使用して、`utils_print.h` を検索します。

## 演習 1: Vitis ソフトウェア開発ツールの実行

---

2-9-3. [Outline] ビューで `utils_print.h` を選択して、ソースファイルの「include」を検索します。

### 質問 10

`utils_print.h` を開く簡単な方法は何ですか?

---

---

---

### 質問 11

何がハイパーリンクになりますか?

---

---

---

この演習は、ZCU104 ボードまたはエミュレートモードで実行できます。この情報はコードと通信する必要があります。ボード上で実行されている場合、LED を制御するペリフェラルを構成するために実行する必要があるハードウェア初期化コードと、LED 信号を駆動するコードがあります。

エミュレーションでこの機能をオンにすると、エミュレーターは PS のみをモデル化し PL はモデル化しないため、完了できない AXI トランザクションを待機するため、エミュレーターがロックします。

次に、コードを実行するプラットフォームを指定するコンパイラシンボルを定義します。

- ZCU104 ボード ユーザー: シンボル `HARDWARE` を定義します。
- QEMU エミュレーターのユーザー: シンボル `EMULATION` を定義します。

アプリケーションプロジェクトにシンボルを追加することは、プロジェクト全体に表示される `#DEFINE SYMBOL_NAME` ステートメントを追加することと同じです。条件付きプリプロセッサステートメントを使用してコードを保護して、条件付きでコードをコンパイルするためには、シンボルを使用するのが一般的です (例: `#ifdef SYMBOL_NAME ...#endif`)。

## 2-10. コンパイラ設定にシンボルを追加します。

2-10-1. [Explorer] タブフォルダーで `drvVitis_app` を右クリックして、コンテキスト メニューを開きます (1)。

2-10-2. [C/C++ Build Settings] を選択し、プロジェクトの [edit] ウィンドウを開きます (2)。

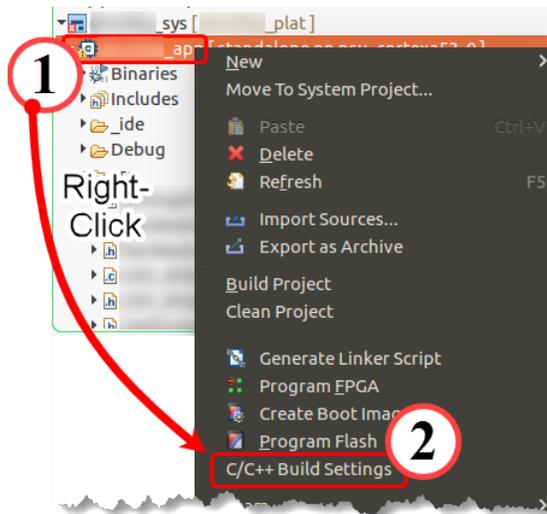


図 1-25: アプリケーションプロジェクト用の [C/C++ Build] 設定へのアクセス

2-10-3. [properties] ウィンドウの左側から **Settings** ] を選択してツール設定を表示します(1)。

必要に応じて、[C/C++ Build] ブランチを展開して、[Settings] エントリを表示します。

2-10-4. まだ選択されていない場合は、 **Tool Settings** タブを選択してください(2)。

2-10-5. 適切なコンパイラエントリの下で、 **Symbols** を選択します (3)。

必要に応じて、コンパイラタイプの下にあるブランチを展開します。

ザイリンクスのソフトウェア開発ツールは複数のプロセッサに対して複数のコンパイラツールチェーンをサポートしているので、使用している特定のプロセッサ用のコンパイラを選択します。

一般的に、これは、Microblaze プロセッサが搭載されているデバイスに関係なく、デバイスの Zynq® ファミリの Arm® gcc コンパイラと MicroBlaze™ プロセッサの GNU になります。重要な点は、コンパイラオプション設定にいますることです。

シンボルは 1 つずつ入力します。このリストの元素ごとに、手順の 6~8 を実行します:  
**HARDWARE or EMULATION。**

2-10-6. 定義済みシンボル領域の下で、 **Add** (📄) アイコンをクリックしてダイアログボックスを開き、新しいシンボル名を入力します(4)。

2-10-7. [Enter Value] ダイアログボックスにシンボル名を入力します (5)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

2-10-8. [OK] をクリックして、新しいシンボルを一つずつ追加します (6)。

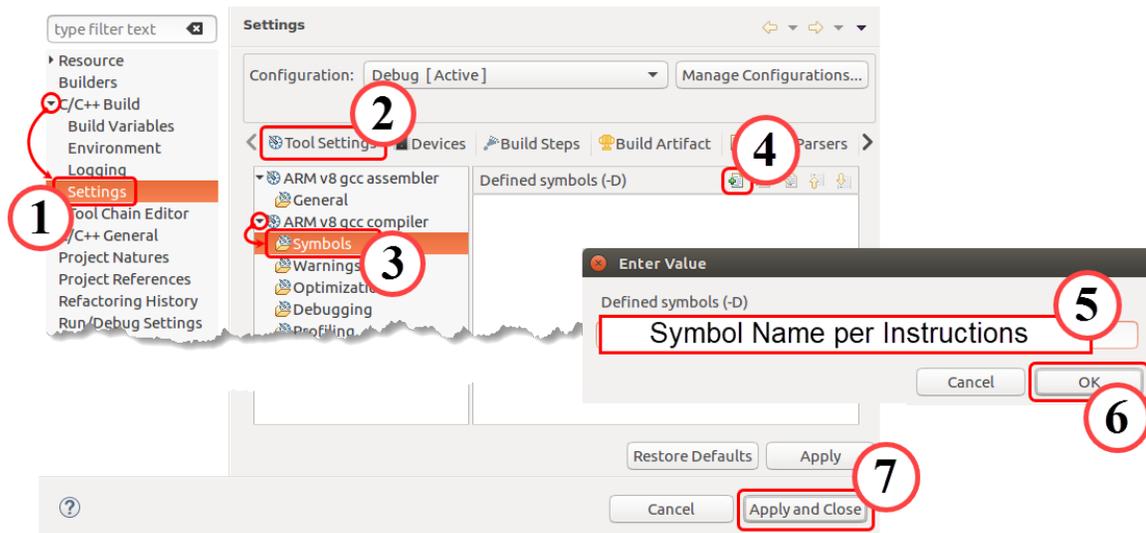


図 1-26: 新しいコンパイラシンボルの定義

2-10-9. すべてのシンボルを追加したら、[Apply and Close] をクリックして変更を適用し、[Properties] ダイアログボックスを閉じます (7)。

## ハードウェア上でアプリケーションの起動

### 手順3

この手順には、ZCU104 ボードを使用するユーザー向けの情報が含まれています。ZCU104 ボードがない場合、「エミュレーターでアプリケーションの起動」の手順にスキップできます。

提供されるデザインでは、LED とシリアルポートメッセージを使用します。QEMU は PS のエミュレーターであり、LED 出力と同等のものを持っていませんが、シリアルポート経由で必要なメッセージを生成します。

ボードの電源を入れ、シリアル通信セッションを開始します。

### 3-1. ハードウェアターゲットで作業を開始する前、ZCU104 ボードを立ち上げます。

3-1-1. JTAG/UART への USB ケーブルをボードからホストマシンに接続します。

3-1-2. 次の図に示すように SW6 スイッチを設定し、ボードが JTAG から起動するようにします。

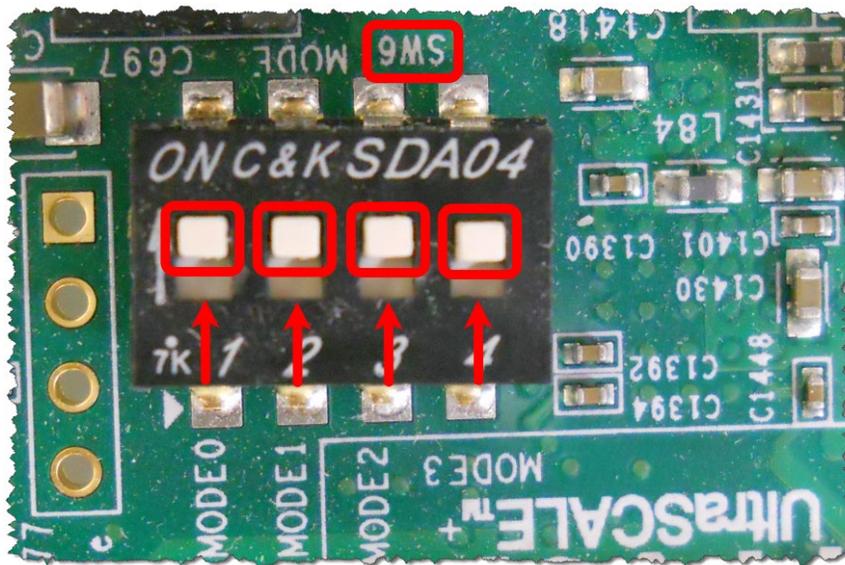


図 1-27: JTAG ブートの設定 (ZCU104 ボード)

3-1-3. ボードに電源を投入するには、電源スイッチを「ON」にスライドさせます。

### 3-2. 開発ボードの USB シリアルポートに接続する COM ポートを決定します。

3-2-1. ホストプラットフォームと開発ボードの間に USB ケーブルが接続されていることを確認します。

VM を使用している場合、[Devices] → USB → [Xilinx JTAG+3Serial] を選択して、VM が USB 接続にアクセスできるようにします。

**メモ:** VM から USB ポートにアクセスするには、VirtualBox 拡張パックが必要です。商用利用にはライセンスの購入が必要です。

## 演習 1: Vitis ソフトウェア開発ツールの実行

ヒント: これは、VM 自体ではなく、仮想マシンシェルから実行されます。VM ビューの上部にあるメニューを探します。

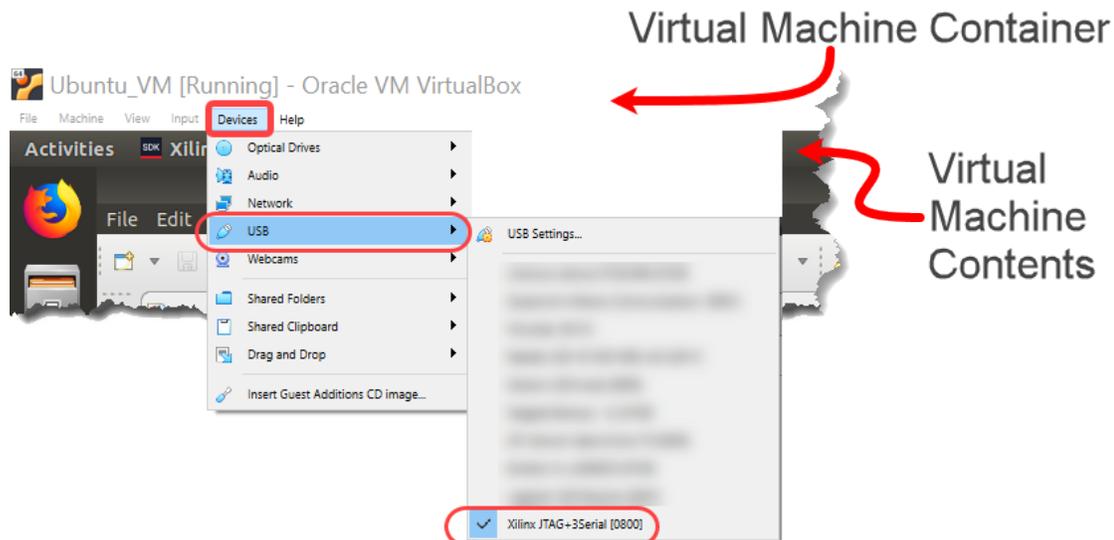


図 1-28: Determining the COM Port in the VirtualBox Environment

提供されている VM を使用していない場合、オペレーティングシステムに基づいて、使用する適切なシリアルポートを識別する必要があります。

メモ: 提供されている VM を使用する場合、`/dev/ttyUSB1` に接続されているシリアルポートを使用します。

**2019.2 VM ユーザー:** 2019.2 VM を使用していて、Vitis シリアルターミナルが VM の USB ブリッジに正しく接続されていない問題が起こる場合、代わりに `gtkTerm` アプリケーションを使用してボードと通信します。

Vitis シリアルターミナルはシリアルポート/UART 通信をサポートするインターフェイスです。より一般的なターミナル タブは、SSH や Telnet など他の形式をサポートすることができます。

### 3-3. [Vitis Serial Terminal] タブを検索します (通常、このタブはデフォルトでは開いていません)。

3-3-1. [Vitis Serial Terminal] タブが現在表示されていない場合、[Window] (1) → [Show view] (2) を選択して、一般的に開かれているビューのリストを表示します。

3-3-2. [Xilinx] (3) を展開し、[Vitis Serial Terminal] (4) を選択します。

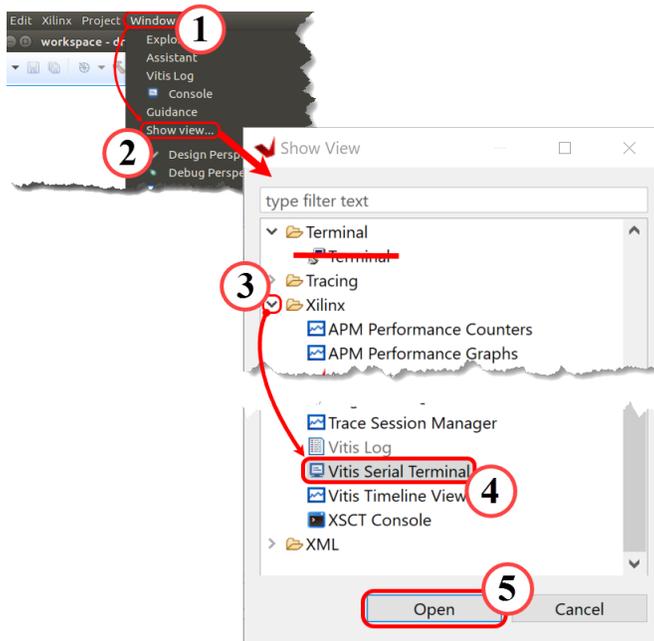


図 1-29: Vitis シリアルターミナルを開く

3-3-3. [Open] をクリックして、Vitis シリアルターミナルにアクセスします (5)。

通常、このタブは [Console] タブの横に開きます。このターミナルをこの位置のままにするか、タブを別のウィンドウ領域にドラッグできます。

### 3-4. Vitis シリアルターミナルを構成します。

3-4-1. 緑色の「+」記号をクリックして、[Connect to Serial Port] ダイアログボックスを開きます。

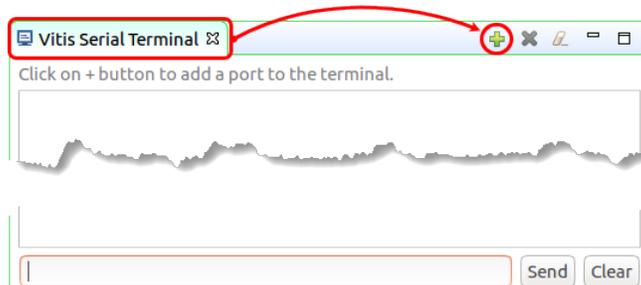


図 1-30: ターミナルへのポートの追加または関連付け

3-4-2. 通信するデバイスに接続されているシリアルポートを選択します (1)。

これは、ボードへのシリアルポート/USB 接続に関連するポート番号です。このポートを表示するには、ボードの電源をオンにする必要があります。

**メモ:** CustEd VM を使用している場合、これは /dev/ttyUSB1 になります。これには、拡張パックがインストールされている必要があります (EULA を参照します)。

3-4-3. ボーレートを **115200** に設定します (2)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

3-4-4. 他の設定はデフォルトのままにしておきます。

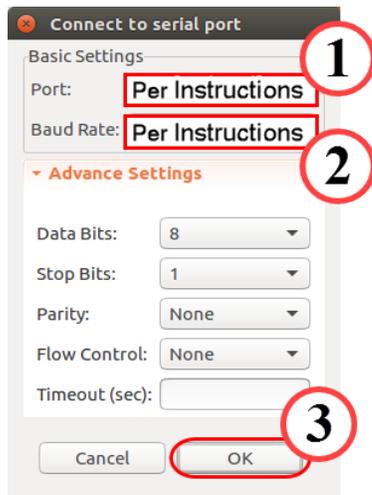


図 1-31: ターミナルを設定する

3-4-5. [OK] をクリックして、これらの設定を保存し、ターミナル セッションを開始します (3)。

この演習で使用されるハードウェアデザインには、プログラマブルロジック (PL) にロードする必要があるデザインの一部が含まれています。つまり、ソフトウェアを実行する前に、ビットストリームをファブリックにロードする必要があります。アプリケーションを適切なプロセッサにロードするだけでなく、PL ビットストリームもプログラムされるランコンフィギュレーションを作成します。

Run コンフィギュレーションは、アプリケーションの実行時にシステムがどのように動作するかを定義します。多くのスイッチとオプションがありますが、最も一般的なものは次の通りです。

### 3-5. peripheral\_test の Run コンフィギュレーションを設定します。

3-5-1. [Explorer] ウィンドウで `drvVitis_app` を右クリックします (1)。

3-5-2. コンテキストメニューから [Run As] を選択します (2)。

3-5-3. Run Configurations を選択します (3)。

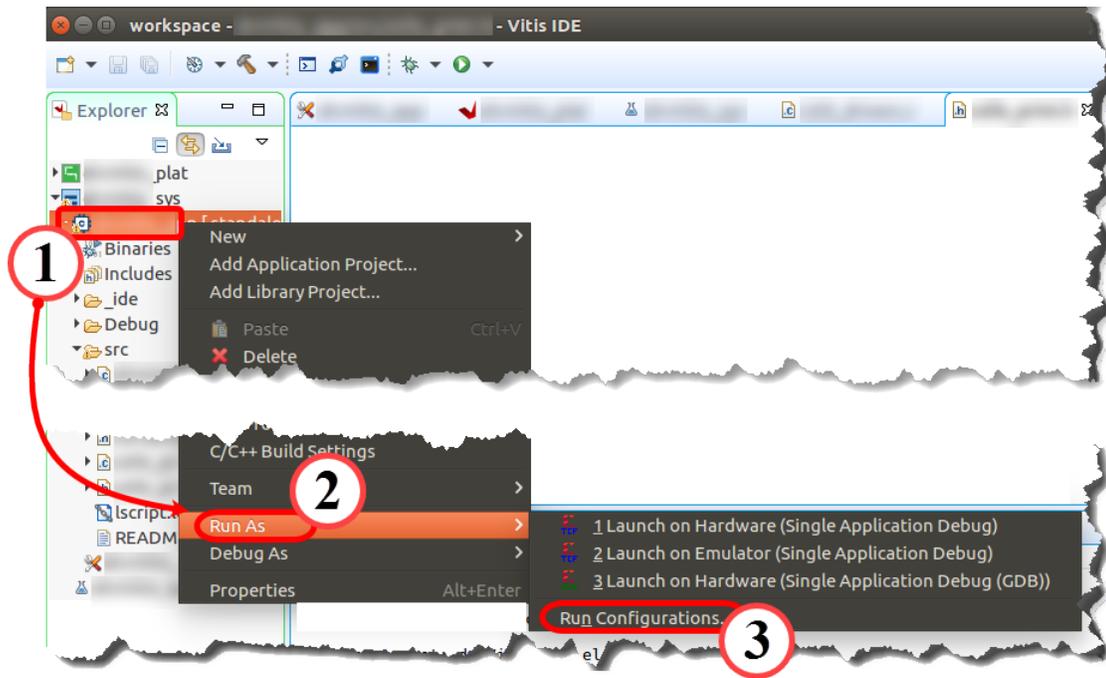


図 1-32: アプリケーション用の Run コンフィギュレーションの作成

[Run Configurations] ダイアログ ボックスが開きます。

3-5-4. [Single Application Debug] をダブルクリックして、起動するコンフィギュレーションを作成します (1)。

または、[Single Application Debug] を選択して、[New Launch Configuration] アイコン (📄) をクリックします (2)。

## 演習 1: Vitis ソフトウェア開発ツールの実行

これがプロジェクト用に作成される最初のデバッグ構成の場合、さまざまなオプションの説明が左側のペインに表示されます。既に 1 つまたは複数のコンフィギュレーションが存在する場合は、最後に開いたコンフィギュレーションが表示されます。いずれにしても、新しいコンフィギュレーションは常に追加できます。既存のコンフィギュレーションは左側のパネルに表示されており、編集用に選択できます。

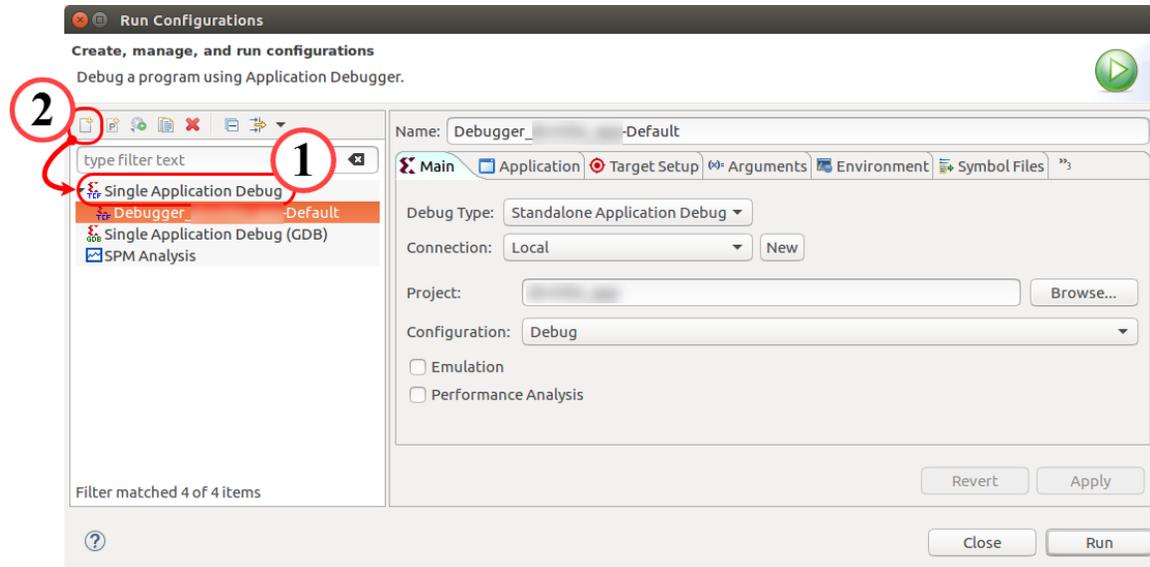


図 1-33: 新しいランコンフィギュレーションの作成

[System Debugger using Debug\_drvVitis\_app.elf on Local] という名前の新しいランコンフィギュレーションが作成され、構成情報が右側のペインに表示されます。

デフォルトでは、[Main] タブが選択されており、構成に関する一般情報が表示されます。

- 3-5-5. [Target Setup] タブを選択して、デバイスの起動方法を指定できます (1)。
- 3-5-6. ビットストリームファイルが存在することを確認します (2)。

通常、PL にデザインコンポーネントがない場合、またはビットストリームが XSA ファイルの一部としてエクスポートされていない場合、ビットストリームはありません。通常、XSA にビットストリームが含まれている場合、このフィールドは自動的に入力されます。

演習 1: Vitis ソフトウェア開発ツールの実行

3-5-7. [Program FPGA] オプションが選択されていることを確認します (3)。

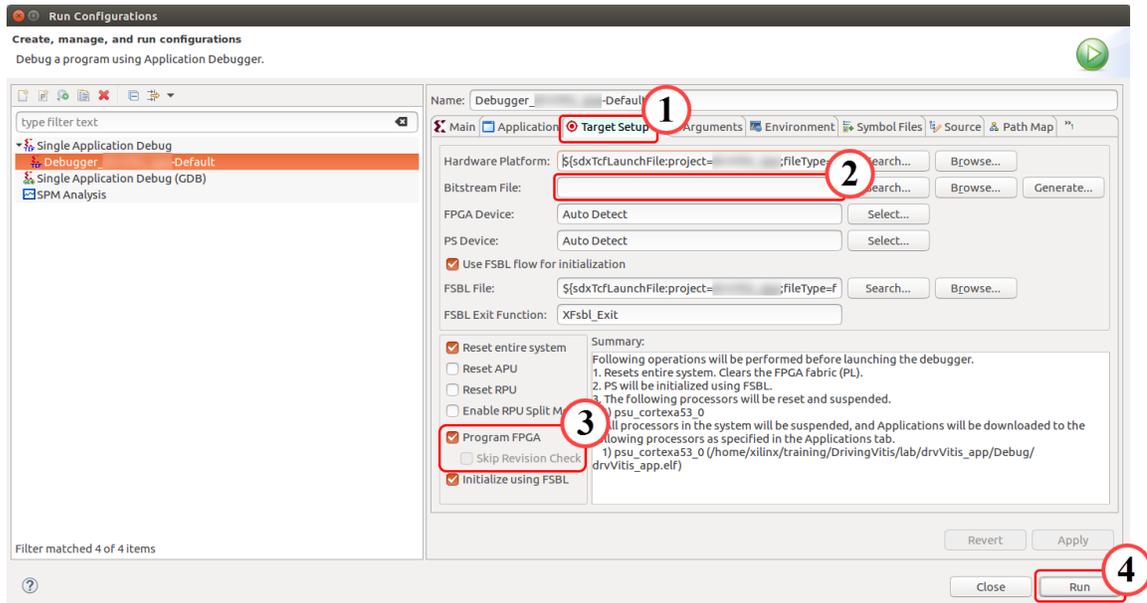


図 1-34: [Run Configurations] ダイアログ ボックス

他の既存のコンフィギュレーションと共に新しいコンフィギュレーションが表示され、使用しているアプリケーションの名前が付いています。

また、アプリケーションの名前を使用して、多くのフィールドが自動的に入力されることに注意します。つまり、アプリケーションの名前が XYZ の場合、[Name] に XYZ Debug が入力されます。ほかの設定はデフォルトのままにしておきます。

3-5-8. [Run]をクリックして、ダイアログボックスを閉じ、ハードウェア評価ボードでソフトウェアアプリケーションを起動します (4)。

3-5-9. 「アプリケーションと対話」の手順にスキップします。

## エミュレーターでアプリケーションの起動

## 手順4

この手順は、CloudShare ユーザーおよび、VM に拡張パックがインストールされていないユーザーなど、ZCU104 ボードを使用していないユーザー向けです。

提供されるデザインでは、LED とシリアルポートメッセージを使用します。QEMU は PS のエミュレーターであり、LED 出力と同等のものを持っていませんが、必要なシリアルポートメッセージを生成します。以前に定義したコンパイラシンボルは、LED 出力を無効にします。シリアルポートメッセージが [Emulation Console] ビューに表示されます。

QEMU エミュレーター用のランコンフィギュレーションを作成します。

Run コンフィギュレーションは、アプリケーションの実行時にシステムがどのように動作するかを定義します。多くのスイッチとオプションがありますが、最も一般的なエミュレーションは次の通りです。

演習 1: Vitis ソフトウェア開発ツールの実行

4-1. **drvVitis\_app** のエミュレーターをターゲットとするランコンフィギュレーションを設定します。

- 4-1-1. [Explorer] ウィンドウで **drvVitis\_app** を右クリックします (1)。
- 4-1-2. コンテキストメニューから **[Run As]** を選択します (2)。
- 4-1-3. **Launch on Emulator (Single Application Debug)** を選択します (3)。

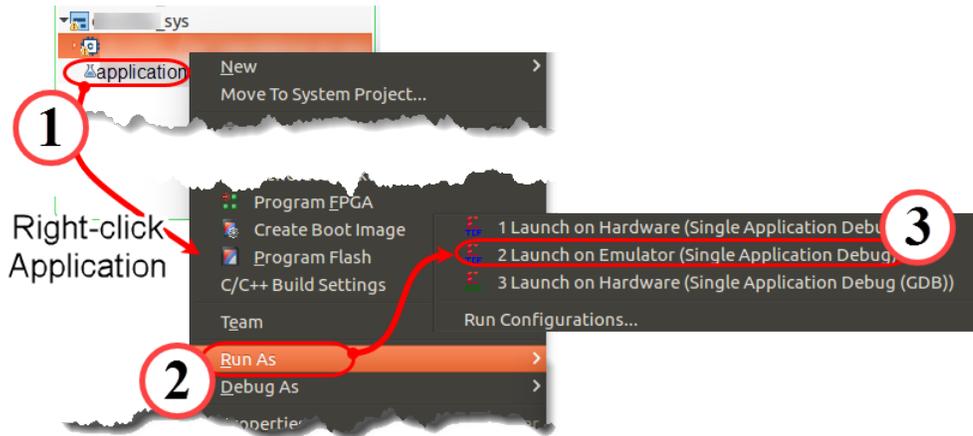


図 1-35: エミュレーターでアプリケーションの起動

[Run Configurations] ダイアログ ボックスが開きます。

- 4-1-4. アプリケーションを実行する前、エミュレーターを実行するように求められた場合、**[Start Emulator and Run]** を選択します。

エミュレーターが起動し、新しく開いたエミュレーターコンソールで進行状況をモニターできます。

## アプリケーションと対話

## 手順5

これで環境が構成され、アプリケーションが開始されるので、アプリケーションとの対話に進むことができます。これらの命令は、ハードウェアとエミュレーションの実行の両方に適用できます。

5-1. ターミナルで出力の表示を確認します。

- 5-1-1. タブを選択して、アプリケーション出力を表示します。

ハードウェアユーザーの場合: [Vitis Serial Terminal] タブを選択します。

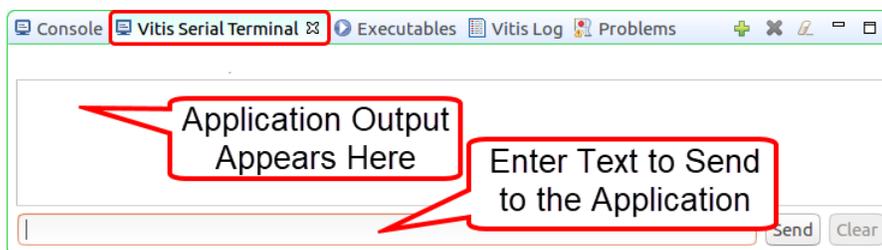


図 1-36: Vitis シリアルターミナル

エミュレーターユーザーの場合: [Emulation Console] タブを選択します。

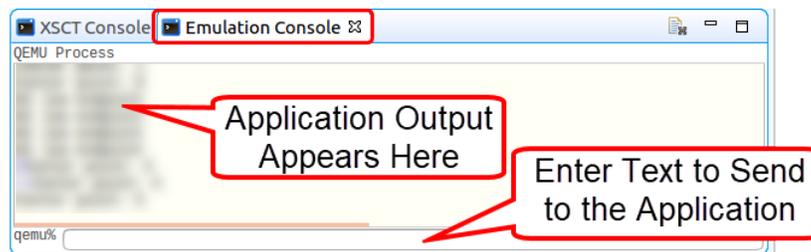


図 1-37: エミュレーションコンソール

ハードウェアで実行している場合、ターミナルに「Driving Vitis test code starting for hardware ...」というメッセージが表示されます。

エミュレーションを使用している場合、エミュレーションコンソールに「Driving Vitis test code starting for emulation」というメッセージが表示されます。

## 5-2. LED を駆動するようにプログラムに指示します。

5-2-1. 適切なビューの入力フィールドに「+」文字を入力します。

5-2-2. 入力フィールドの横にある [Send] ボタンをクリックするか、<Enter> を押して、その文字または文字列をアプリケーションに送信します。

8 LED ボード上のアプリケーションにガウスの表示パターンで、LED を駆動するように指示します。ZCU104 などのボード (4 つのユーザー LED があり) は、下の 4 つの LED の輝度のみが変更されます。

アプリケーションが LED を異なる輝度レベルで駆動することを覚えておいてください。中心から離れた各 LED は、隣接する LED の半分の明るさです。

5-2-3. 開発ボードのユーザー LED とそのステータスメッセージ、またはエミュレーターからメッセージのみを表示します。

表示されるメッセージは、「中心」(または最も明るい LED) がどこにあるかを示します。+ シンボルは中心を片側に移動し、- シンボルは中心を反対側に移動します。

中心が 8 つの位置の範囲外に移動すると、その方向にそれ以上移動できないことを示すメッセージが表示されます。

ユーザーは、0 から 7 まで数字を入力して、中心を特定の位置に強制できます。

5-2-4. コンソールにさまざまなシンボル {+, -, 0, 1, 2, 3, 4, 5, 6, 7} を入力し、出力を確認します。

ほかの文字を入力しても効果を確認できます。

**メモ:** LED が点灯し、それぞれが異なる光の強さになります。アプリケーションによって LED の光の強さが設定されます。LED は PL のピンに接続されて、ソフトウェアでそれらを駆動する方法は、PL でインスタンス化された IP ブロックとのインターフェイスです。

## 演習 1: Vitis ソフトウェア開発ツールの実行

---

### 質問 12

writeStr\_NL() ルーチンはどこに出力されますか。入力 (stdin) および出力 (stdout) デバイスは、どこでコンフィギュレーションされますか。

---

---

---

### 質問 13

ソフトウェアを修正した場合、ハードウェアを再インプリメント (変換、マッピング、配置および配線) する必要がありますか? ソフトウェアが修正された場合、何が起きなければなりませんか。ハードウェアおよびソフトウェアの更新の関連性を説明してください。

---

---

---

## XCST (Tcl) インターフェイスの使用

## 手順6

Vitis IDE は Tcl インターフェイスをサポートしているので、手間を削減するために、複雑なプロセスや反復プロセスを自動化できます。この強力な機能のいくつかを確認します。

Vitis エンベデッドソフトウェアの開発ユーザーガイド (UG1400)、「XCST ユース ケース」の章およびザイリンクス ソフトウェア コマンド ライン ツール (XCST) リファレンス ガイド (UG1208) では、使用可能なコマンドについて説明し、多くの例を示しています。

XCST コンソールは、Vitis IDE とスタンドアロンのコマンドターミナルウィンドウの両方で使用できます。このコンソールでは、アクセス方法に関係なく、Tcl コマンドを直接入力できます。これらのコマンドは、スクリプトまたは個別のコマンドとして実行できます。

Tcl コマンドを使用できるようにすることで、特定のポイントまでデザインを構築する場合、繰り返しプロセスを実行する場合、または十分に文書化された段階的なプロセスからデザインを再構築する場合に特に役立ちます。

ここで、XCST コンソールを使用して確認します。

6-1. ザイリンクスソフトウェア開発ツールに Tcl ベースのコマンドを直接入力できるように、XSCT コンソールを開きます。

6-1-1. [Window] (1) → [Show view] (2) を選択して、開かれているビューのリストを表示します。

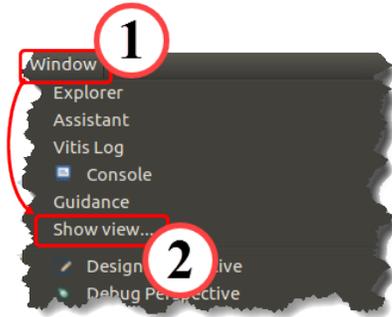


図 1-38: [Show View] の選択

6-1-2. Xilinx エントリを展開すると、ザイリンクス固有のビューが表示されます (3)。

6-1-3. [XSCT Console] を選択します (4)。

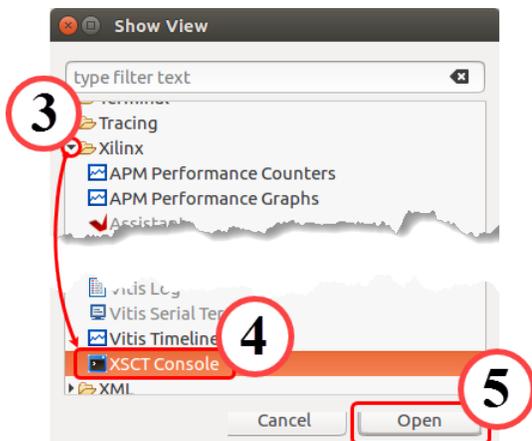


図 1-39: XSCT コンソールの場所

6-1-4. [Open] をクリックして、XSCT コンソールを開きます (5)。

通常、このコンソールは、[Vitis Serial Terminal] タブと同じグループのタブで開きます。また、タブをクリックして、ほかのウィンドウグループにドラッグできます。

または、[XSCT Console] アイコン (  ) をクリックします。

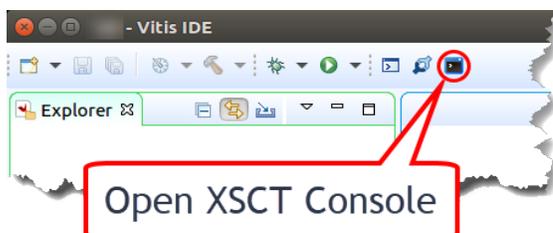


図 1-40: XSCT コンソールを開く

## 演習 1: Vitis ソフトウェア開発ツールの実行

6-1-5. XSCT コンソールを拡大するか、コンソールをメインの作業領域にドラッグします。

[Explorer] タブが表示されたままになっていることを確認します。

6-2. Tcl コマンドのみを使用して、以下の仕様の新しいアプリケーションプロジェクトを作成します。

- [Application name]: hello\_app
- [Domain project]: スタンドアロン BSP の名前: APU1\_dom
- プロセッサ: psu\_cortexa53\_1
- [System Project]: <system\_project>
- [Platform Project]: drvVitis\_plat
- [Template]: Hello World

多くの XSCT コマンドは、シングルコマンドラインで複数のプロジェクトの作成を実行できます。ここで、プロジェクトを個別に作成します。

6-2-1. XSCT コンソールの下部にある XSCT 入力バーに次のコマンドを入力して、drvVitis\_plat プラットフォームプロジェクトの追加ドメインを作成します。

\$DrivingVitis/support にある Lab1\_section6.txt もご覧ください。

```
domain create -name APU1_dom -proc psu_cortexa53_1 -os standalone -desc {BSP  
supporting second A53};
```

注意)表示の都合で折り返していますが、コマンドは改行を入れずに入力してください。

## 質問 14

どうすれば、このコマンドのオプションに何があるかを知ることができますか?

---

---

---

6-2-2. [platform] → psu\_cortexa53\_1 を展開して、ドメインが作成されることを確認します。

新しいドメイン (APU1\_dom) がリストされているはずです。

XSCT コンソールに domain list を入力して、使用可能なすべてのドメインを表示できます。

6-2-3. 次のコマンドを入力して、新しいドメイン用の Hello World アプリケーションを作成し、既存のシステムプロジェクトに追加します。

\$DrivingVitis/support にある Lab1\_section6.txt もご覧ください。

```
app create -name hello_app -domain APU1_dom -lang C -template {Hello World}  
-platform drvVitis_plat -sysproj drvVitis_sys;
```

注意)表示の都合で折り返していますが、コマンドは改行を入れずに入力してください。

このコマンドの実行には数秒かかります。[Vitis IDE Console] タブ (XSCT Console タブではありません) で進捗状況を追跡できます。

## 質問 15

このコマンドの結果は何ですか?

---

---

---

6-2-4. 次のように入力して、既存のアプリケーションを表示します。

```
app list
```

6-2-5. 次のように入力して、この手順の最後の指示に備えて作成したアプリケーションを削除します。

```
app remove hello_app
```

[Explorer] ペインに表示されているアプリケーションの変化に注目してください。

6-2-6. 次のように入力して、既存のドメインを表示します。

```
domain list
```

6-2-7. 次のように入力して、この手順の最後の指示に備えて作成したドメインを削除します。

```
domain remove APU1_dom
```

[Explorer] ペインのプラットフォームセクションの下にあるドメインの変化に注目してください。

6-2-8. 上記のプロセスを繰り返して、`drvVitis_app` および `pus_cortexa53_0` の `standalone_domain` を削除します。

これにより、プラットフォームプロジェクトと空のシステムプロジェクトが残ります。

6-3. Tcl コマンドのみを使用して、3 つの新しいアプリケーションプロジェクトを作成します。

このプロセスを自動化するために、Tcl のいくつかの基本機能を使用します。

ループを 3 回繰り返し、アプリケーションごとに新しい名前とアプリケーションの新しいドメインを提供する Tcl proc (または「プロシージャ」) を完成させます。すべてのドメインが既存のプラットフォームプロジェクトに追加され、すべてのアプリケーションが既存のシステムに追加されます。

6-3-1. 任意のテキストエディターを使用して、`$TRAINING_PATH` → `DrivingVitis` → `support` → `xsct_example.tcl` を開きます。

6-3-2. `generate3apps` と呼ばれるプロシージャ、または 82 行目付近を見つけます。

6-3-3. 94 行目付近またはその近くのコメントを見つけます: 「domain creation goes here」。

6-3-4. 前述のステップで示したコマンドを用いて、Tcl 変数が使用されるようにドメインプロジェクトを作成するコードを記述します。

6-3-5. 98 行目付近またはその近くのコメントを見つけます: 「application creation goes here」。

6-3-6. 前述のステップで示したコマンドを用いて、Tcl 変数が使用されるようにアプリケーションを作成するコードを記述します。

完成した Tcl スクリプトは `[support]` → `[golden]` ディレクトリにあります。

6-3-7. Tcl ファイルを保存します。

## 演習 1: Vitis ソフトウェア開発ツールの実行

---

6-3-8. 新しいコードを実行するには、XSCT Console タブに戻ります。

6-3-9. コードがあるディレクトリに移動するには、次のように入力します。

```
cd $::env(Driving_Vitis)/support
```

6-3-10. スクリプトを実行するには、以下を入力します。

```
source xsct_example.tcl  
generate3apps
```

6-3-11. XSCT コンソールと [Explorer] の両方でコードの進捗状況をモニターします。

これは、XSCT で利用可能な機能の一部にすぎません。このツールの機能の詳細については、Xilinx Software Command-Line Tool (XSCT) Reference Guide (UG1208) を参照して下さい。

### 6-4. Vitis IDE を閉じます。

6-4-1. [File] → [Exit] を選択して設定を保存し、ツールを終了します。

終了することを確認するダイアログ ボックスが表示されます (オプションの設定によっては表示されない場合があります)。

終了時の確認が表示されないようにするオプションがあります。

6-4-2. このダイアログが表示されたら [Yes] をクリックして設定の保存を終了し、ツールを終了します。

オプション: VirtualBox ファイル システムをクリーンアップします。

いくつかのシステムは、メモリに制約がある場合があります。ワークスペースを削除すると、ディスク領域の一部が解放され、ほかの演習を実行できるようになります。

グラフィカルインターフェイスまたはコマンドラインインターフェイスを使用して、実行した演習を含むディレクトリを削除できます。どちらの方法も選択できます。両方のプロセスは、\$DrivingVitis ディレクトリのすべてのファイルを再帰的に削除します。

### 6-5. VirtualBox ファイル システムをクリーンアップします。

GUI の使用の場合:

6-5-1. グラフィカルブラウザ (Windows の場合: <Windows> キー + <E> を押し、Linux の場合: <Ctrl + N> を押します) を使用して、\$DrivingVitis に移動します。

6-5-2. **DrivingVitis** を選択します。

6-5-3. <Delete> を押します。

または

コマンド ラインの使用の場合:

6-5-4. ターミナルウィンドウを開きます (Windows の場合: <Windows> キー + <R> を押してから、cmd と入力します; Linux の場合: <Ctrl + Alt + T> を押します)。

6-5-5. 次のコマンドを入力して、ワークスペースの内容を削除します。

**Windows の場合:** rd /s /q \$DrivingVitis

**Linux の場合:** rm -rf \$DrivingVitis

## まとめ

今、エンベデッドソフトウェアプロジェクトを構築するために必要なすべての手順が完了しました。

- ハードウェア、BSP、およびアプリケーションプロジェクトの作成
- アプリケーションプロジェクトに既存のソースコードの追加
- 構文エラーの検索と修正
- ハードウェアまたはエミュレーションでアプリケーションの実行

## 解答

1. この問題を解決するにはどのような手順が必要ですか？

この問題は、関数内で `delay` という名前の変数を使用する前に定義することで解決できます。

2. このエラーを確認して修正する最良の方法は何ですか？

このエラーの直前に警告アイコンが表示されているので、関係する可能性があります。警告は、変数 `dely` が使用されていないことを示します。これは単なる入力ミスであり、`[dely]` の部分は `[delay]` とするのが正しです。

3. エディターのデフォルトの設定では行番号が表示されます。必要に応じて、行番号をオフにするにはどうすればよいですか？

エディターの左マージンを右クリックし、コンテキストメニューから `[Show Line Numbers]` をクリックします。

4. ファイルの再コンパイルの結果として何が起こりましたが？ ヒント: `[Problems]` タブを表示します。

変数名を修正することで、`LED_drivers.c` のエラーがなくなります。これにより、別のファイルの別のエラーが表示されます。

5. これらの小さな赤と黄色の矩形は、何ですか？

これらは、ソースファイルの警告およびエラーの位置に対応するマーカーです。黄色は警告を表し、赤色はエラーを表します。赤色の矩形をクリックすると、ソースコードビューが変更され、そのエラーに対応するコード行が表示されます。

6. 小かっこを追加するとどうなりますか？

特定の種類のエラーと警告は、コードを再コンパイルしなくても検出できます。括弧が閉じられると、ツールは変更を検出してソースコードを再スキャンします。条件文が完成すると、多くの警告とエラーが消えます。ただし、エラーマーカーは、コードが再構築されるまで残ります。

7. `[Outline]` ビューはどのように使用しますか？

`[Outline]` ビューにリストされているエレメントをクリックすると、ソースファイル内のその位置にジャンプします。

8. `LED_driver_select(int)` 機能は、どの行番号から開始されますか？ (`LED_drivers.c` にあります)

この関数は、64 行目付近から始まります。

9. 別のソースタブをクリックすると、何が起こりますか？

`[Outline]` タブは、アクティブなソースタブに合わせて変更されます。

10. `utils_print.h` を開く簡単な方法は何ですか？

`[Explorer]` ビューに戻って `utils_print.h` を検索してダブルクリックするより、`<Ctrl>` を押したままソースファイルの `utils_print.h` にマウスを置く方がはるかに簡単です。これは、`[include]` をハイパーリンクに変更します。ハイパーリンクされた `include` ファイルをクリックして、新しいウィンドウで `include` ファイルを開きます。

さらに簡単な方法は、[Outline] ビューで `utils_print.h` をダブルクリックして、新しいエディターウィンドウで開くことです。

11. 何がハイパーリンクになりますか？

〈Ctrl〉を押したまま、さまざまなエレメントの上にカーソルを置きます。[Outline] タブにリストされているものはすべてハイパーリンクになります。これは、関数、マクロ拡張などのためにソースを早く検索する際に利用できます。

12. `writeStr_NL()` ルーチンはどこに出力されますか？ 入力 (`stdin`) および出力 (`stdout`) デバイスは、どこでコンフィギュレーションされますか？

`writeStr_NL()` ルーチンは、シリアルポート出力デバイスにマップされている `stdout` に出力します。同様に、`stdin` もシリアルポート入力デバイスにマップされます。

これらのストリームは、さまざまなペリフェラルに接続でき、デフォルトでシリアルポートに接続できます。これらは、デバッグプラットフォーム (`platform.spr`) に移動し、構成するドメインのボードサポートパッケージを選択することでカスタマイズできます。設定は、[Overview] → [standalone] → {`stdin` | `stdout`} にあります。

13. ソフトウェアを修正した場合、ハードウェアを再インプリメント (変換、マッピング、配置および配線) する必要がありますか？ ソフトウェアが修正された場合、何が起きなければなりませんか。ハードウェアおよびソフトウェアの更新の関連性を説明してください。

依存関係はハードウェアによって異なります。つまり、ソフトウェアの変更はハードウェアに影響を与えません。ただし、ハードウェアの変更はプラットフォームプロジェクトに再インポートする必要があり、ドメインの再構築が必要になります。

BSP の再構築はアプリケーションに影響を与える可能性があります。ソフトウェアがアクセスしているペリフェラルがデザインから削除されたとします。ソフトウェアは存在しないペリフェラルを参照することになるため、多数のエラーが発生します。

14. どうすれば、このコマンドのオプションに何があるかを知ることができますか？

XSCT にはビルトインのヘルプメニューがあります。 `domain create -help` と入力して、ドメイン作成のオプションといくつかの例を表示します。

15. このコマンドの結果は何ですか？

アプリケーションプロジェクト、BSP プロジェクト、ハードウェアプロジェクトの 3 つのプロジェクトが作成されます。