SDSoC 環境ユーザー ガイド

SDSoC 環境の概要

UG1028 (v2015.4) 2015 年 12 月 14 日

本資料は表記のバージョンの英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。資料によっては英語版の更新に対応していないものがあります。日本語版は参考用としてご使用の上、最新情報につきましては、必ず最新英語版をご参照ください。

XILINX
ALL PROGRAMMABLE.

改訂履歴

次の表に、この文書の改訂履歴を示します。

日付	バージョン	改訂内容
2015 年 12 月 14 日	2015.4	ソフトウェアの変更を反映するようアップデート
2015 年 9 月 30 日	2015.2.1	ソフトウェアの変更を反映するようアップデート
2015 年 7 月 20 日	2015.2	初版

目次

第 1 章

概要

SDSoC™ (Software-Defined Development Environment for System-on-Chip) 環境は、Zynq®-7000 All Programmable SoC プラットフォームを使用してヘテロジニアス エンベデッド システムをインプリメントするための Eclipse ベースの統合設計環境 (IDE) です。SDSoC 環境では、ソフトウェア エンジニアおよびシステム アーキテクト用に、使いやすい Eclipse ベースの IDE を使用したエンベデッド C/C++ アプリケーション開発環境と、ヘテロジニアス Zynq SoC 開発用の包括的なデザイン ツールが提供されています。SDSoC 環境には、プログラマブル ロジックでの自動ソフトウェア アクセラレーションや、システム接続の自動生成などを実行する、フル システム最適化 C/C++ コンパイラが含まれます。SDSoC 環境内のプログラミング モデルは、ソフトウェア エンジニアが簡単に理解できるように設計されています。アプリケーションは C/C++ コードで記述され、プログラマがターゲット プラットフォームとハードウェアにコンパイルするアプリケーション内の関数のサブセットを特定します。この後、SDSoC システム コンパイラによりアプリケーションがハードウェアとソフトウェアにコンパイルされ、ファームウェア、オペレーティング システム、アプリケーション実行ファイルを含むブート イメージを含めた完全なエンベデッド システムが Zynq デバイスにインプリメントされます。

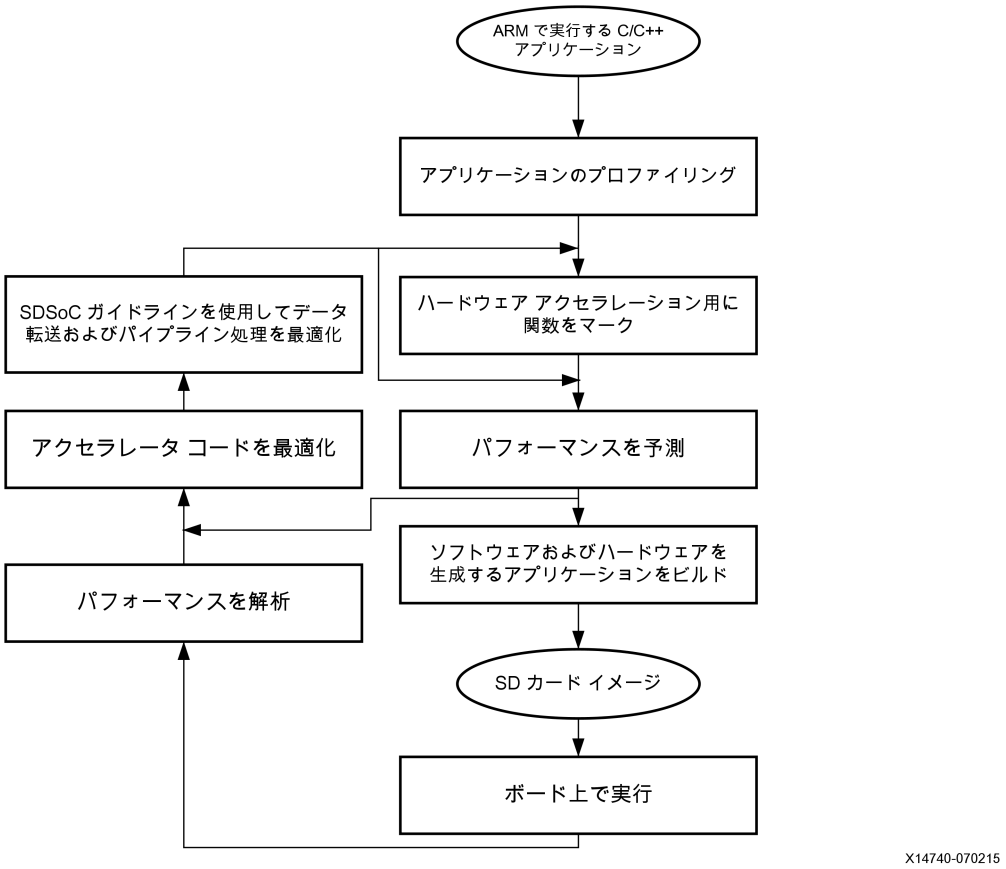
SDSoC 環境では、C/C++ 関数の Zynq デバイス内の ARM CPU だけでなくプログラマブル ロジック ファブリックへのクロスコンパイルおよびリンクを含め、ソフトウェア抽象層を増加することによりハードウェアが抽象化されます。プログラマブル ハードウェアで実行するプログラム関数のユーザー仕様に基づいて、プログラム解析、タスク スケジューリング、プログラマブル ロジックおよびエンベデッド CPU へのバインディングが実行されるほか、ハードウェアおよびソフトウェア コード生成により、ハードウェアとソフトウェア コンポーネント間の通信および連携が調整されます。

SDSoC 環境 2015.4 リリースには、Zynq-7000 All Programmable SoC を搭載した ZC702、ZC706、MicroZed、ZedBoard、および Zybo 開発ボードのサポートが含まれます。このほかにもパートナーから追加のプラットフォームが提供されています。詳細は、SDSoC 環境のウェブ ページを参照してください。

ユーザー デザイン フロー

より高いパフォーマンスを達成するための最初の手順は、アプリケーション内の計算負荷の高いホット スポットでプログラマブル ロジックに移行可能な部分を特定し、ハードウェア用にコンパイル可能な関数に分離することです。SDSoC 環境でプログラマブル ロジック用にコンパイルされた C/C++ コードは、『SDSoC 環境ユーザー ガイド』(UG1027) の「コード ガイドライン」で説明されているコーディング ガイドラインに従っている必要があり、また Vivado® 高位合成 (HLS) のガイドラインにも従う必要があります。たとえば、コードで再帰関数を呼び出したり、メモリをダイナミックに割り当てたり、ポインターを制限なしに使用したりすることはできません。詳細は、『SDSoC 環境ユーザー ガイド』(UG1027) の「プログラマ向け Vivado 高位合成ガイド」を参照してください。RTL IP は、『SDSoC 環境ユーザー ガイド : プラットフォームおよびライブラリ』(UG1146) の「ライブラリの作成」に説明されているように、C 呼び出し可能なライブラリにラップする必要があります。

図 1-1： SDSoC 環境フロー



このセクションでは、Eclipse ベースの GUI を使用した SDSoC 環境の開発プラットフォームについて説明します。この統合開発環境には、開発プロセス、プロジェクト管理、ビルド オートメーションなどをシンプルにするインタラクティブな機能が多数含まれています。これらのほとんどは、makefile を使用してスクリプト記述することもできます。

システム要件

- 次のいずれかのオペレーティング システムの 1 つを実行するホスト：
  - Linux：Red Hat Enterprise Workstation 6.5 ～ 6.6 および 7.0 ～ 7.1 (64 ビット)、Ubuntu Linux 14.04 LTS (64 ビット)
  - Windows：Windows 7.1 Professional (64 ビット) 英語版
- 次を含むザイリンクス SDSoC™ 環境のインストール：
  - SDSoC 環境 2015.4 (Eclipse/CDT ベースの GUI、高位システム コンパイラ、および ARM GNU ツールチェーンを含む)
  - Vivado® Design Suite System Edition 2015.4 (Vivado 高位合成 (HLS) およびザイリンクス ソフトウェア開発キット (SDK) を含む)
- SDSoC 環境には、ザイリンクス® ソフトウェア開発キット (SDK) 2015.4 に含まれるのと同じ GNU ARM ツールチェーンが含まれるほか、SDSoC 環境で使用されるその他のツールも提供されています。SDSoC 環境のセットアップ スクリプトを使用すると、このツールチェーンを使用するように PATH 変数が設定されます。
  - Sourcery CodeBench ツールチェーンには、32 ビットの実行ファイルが含まれているので、Linux ホスト OS のインストールには 32 ビットの互換ライブラリが含まれている必要があります。ザイリンクス バージョンのSourceery CodeBench 入門ガイドは、次の SDSoC 環境のインストール ディレクトリに含まれています。

    ```
    SDK/SDK/gnu/arm/lin/share/doc/xilinx-arm-xilinx-linux-gnueabi/pdf/getting-started.pdf
    ```

    「Host Operating System Requirements」セクションには、32 ビット ライブラリを含む Sourcery CodeBench ソフトウェアのホスト要件についてと、その入手先と入手方法について説明されています。詳細は、Mentor Graphics 社のウェブサイトから入手可能な同様の入門ガイドに記述されています。
  - RHEL 5 64 ビット x86 Linux インストールには通常 32 ビット互換ライブラリが含まれていますが、RHEL 6 には含まれていないので、別に追加する必要がある可能性があります。詳細は、https://access.redhat.com/site/solutions/36238 を参照してください。
  - RHEL、32 ビット互換ライブラリは、ルート アクセス権のあるスーパー ユーザー (またはルート管理者) になって、`yum install glibc.i686` コマンドを実行するとインストールできます。
  - Ubuntu では、32 ビット互換ライブラリは、ルート アクセス権のあるスーパー ユーザー (またはルート管理者) になって、次のコマンドを実行するとインストールできます。追加情報は、SDSoC リリース ノートを参照してください。

    ```
    sudo dpkg --add-architecture i386
    sudo apt-get update
    sudo apt-get install libc6:i386 libncurses5:i386 libstdc++6:i386
    sudo apt-get install libgtk2.0-0:i386 dpkg-dev:i386
    sudo ln -s /usr/bin/make /usr/bin/gmake
    ```
  - ツールチェーンのバージョンは、`arm-xilinx-linux-gnueabi-g++ -v` コマンドを実行すると表示できます。
  - シェル ウィンドウの最後の行に、`gcc version 4.9.2 (Sourcery CodeBench Lite 2015.05-17)` と表示されるはずです。
- ボードからの UART 出力を監視するための mini-USB ケーブル

ライセンスの取得および管理

SDSoC 環境 2015.4 リリースでは、Xilinx FLEXnet License Configuration Manager が使用されています。ライセンス キーを取得する方法については、ザイリンクスの販売代理店にお問い合わせください。

ノード ロックまたはフローティング ライセンス サーバーのどちらか該当する方法を使用してライセンス キーをインストールします。ノード ロック ライセンスは通常 `<home>/.Xilinx` (Linux) または `C:\.Xilinx` (Windows) にコピーされます。既存のフローティング ライセンスを使用したインストールの場合は、通常新しいライセンス ファイルの内容を既存のライセンス ファイルに追加してから、サーバーを再起動します。新しいフローティング ライセンスをインストールするには、たとえば、次のように FLEXnet ユーティリティの lmgrd を実行します。

```
lmgrd -c <path_to_license>/Xilinx.lic -l <path_to_license>/log1.log
```

ノード ロック ライセンスのクライアント マシンは、1 つまたは複数の固定ディレクトリからライセンスを検索します。フローティング ライセンスの場合は、XILINXD_LICENSE_FILE 環境変数に port@server 形式でライセンス ファイルまたはライセンス サーバーへのパスを追加します。

注記 : Windows Explorer を使用して `C:\.Xilinx` フォルダーを作成する場合、`C:\` で右クリックして [新規作成] → [フォルダー] をクリックし、`.Xilinx.` (ピリオド Xilinx ピリオド) というフォルダー名を入力します。Enter を押すと、Windows により `.Xilinx` というフォルダーが作成されます。最後のピリオドは、Windows でフォルダー名の最初の文字としてピリオドが許容されるようにするためのものです。

ヒント : SDSoC 環境 2015.4 リリースのライセンスは、その他のザイリンクス製品と同じ方法で管理されます。SDSoC 環境のライセンス キー ファイルのインストールについては、ローカルのザイリンクス ライセンス管理者にお問い合わせください。

ダウンロード

SDSoC™ 環境をダウンロードするには、SDSoC 環境ウェブ ページにアクセスしてください。

インストール

インストーラー ファイルをダウンロードして実行し、画面の手順に従ってください。次の手順は、典型的なインストール セッションを示しています。

1. `xsetup.exe` (Windows) または xsetup (Linux) インストーラー ファイルを実行します。

   SDSoC インストーラーの [Welcome] ページが表示されます。
2. [Next] をクリックします。

   [Accept License Agreements] ページが表示されます。
3. [I Agree] チェック ボックスをすべてオンにして、ザイリンクスおよびサードパーティのライセンス契約を許諾します。
4. [Next] をクリックします。

   [SDSoC] ページが表示されます。

5. オプションを選択してインストールをカスタマイズします。

   注記： 既に Vivado Design Suite 2015.4 をインストール済みの場合でも、Vivado ツールの SDSoC 環境をインストールする必要がありますが、ケーブル ドライバーをインストールし直す必要はありません。

6. [Next] をクリックします。

   [Select Destination Directory] ページが表示されます。

7. インストール ディレクトリおよびショートカットなどのインストール オプションを選択します。
8. [Next] をクリックします。

   [Installation Summary] ページが表示されます。

9. [Install] をクリックしてインストールを開始します。

インストールが終了したら、次のディレクトリ構造ができます。

```
<path_to_install>/SDSoC/2015.4
     arm-xilinx-eabi
     arm-xilinx-linux-gnueabi
     bin
     data
     docs
     lib
     llvm-clang
     platforms
     samples
     scripts
     SDK
     tps
     Vivado
     Vivado_HLS
     settings64.[csh|sh|bat]
```

インストールされるソフトウェアには、SDSoC 環境 2015.4、Vivado Design Suite 2015.4、Vivado HLS 2015.4、Xilinx SDK 2015.4、および環境を設定するスクリプトが含まれます。

Linux 環境設定スクリプト

SDSoC™ 環境を実行するには、インストーラーで作成される環境設定スクリプト (`settings64.csh` または `settings64.sh`) を使用します。このスクリプトでは、下位ツールのインストール ディレクトリでスクリプトが実行され、PATH 環境がアップデートされます。

環境が正しく設定されたかどうかを確認するには、次のコマンドを入力し、ツールの設定スクリプトに記述されたインストール ディレクトリが検出されたかどうかを確認します。

```
% source settings64.csh
% which sdscc # SDSoC C/C++ build tool version
% which vivado # Vivado design tool version
% which vivado_hls # Vivado High-Level Synthesis (HLS) tools
% which bootgen # Boot image creation tool (2015.4 version)
```

Linux の which コマンドで戻されたパスがインストール ディレクトリのパスと一致しない場合、またはパスが見つからなかった場合は、正しい設定スクリプトが実行されたかどうかを確認してください。

注意： SDSoC 環境を実行するのに使用された各シェルでは、ザイリンクス ツール リリースまたは上記の PATH 環境変数に該当する環境設定スクリプトのみを使用してください。同じシェル内のほかのリリースまたは追加リリースからザイリンクス デザイン ツール環境設定スクリプトを実行すると、SDSoC 環境の動作または結果が正しくなくなることがあります。

Windows 環境設定スクリプト

SDSoC 環境を起動するには、[Xilinx Design Tools] → [SDSoC 2015.4] → [SDSoC 2015.4] をクリックするか、インストーラーで作成されたデスクトップ ショートカットをダブルクリックします。

環境が正しく設定されるかどうかを確認するには、インストーラーで作成された [SDSoC 2015.4 Terminal] ショートカットをダブルクリックして SDSoC ターミナルを起動します。

次のコマンドを入力し、インストール ディレクトリが SDSoC 環境 2015.4 インストールと一致しているかどうか確認します。REM で始まるコメントは入力しないでください。

```
> REM SDSoC C/C++ build tool
> where sdscc
> REM Vivado design tool
> where vivado
> REM Vivado High-Level Synthesis (HLS) tools
> where vivado_hls
> REM Boot image creation tool (2015.4 version)
> where bootgen
```

Linux の where コマンドで戻されたパスがインストール ディレクトリへのパスと一致しない場合、またはコマンドが見つからなかった場合は、SDSoC 2015.4 ターミナルでコマンドを実行したかどうかを確認します。

注意： Cygwin がグローバル PATH 環境変数に含まれていて問題が発生した場合は、それを SDSoC™ 開発環境フローを実行する際に一時的に削除する必要がある可能性があります。

たとえば、コマンド シェルに次を入力します。

```
set PATH=%PATH:c:\cygwin\bin;=%.
```

インストールの有効化

SDSoC™ 環境のインストールを有効にする基本的なフローは、次のとおりです。ここでは、ZC702 ボード用の手順を示します。異なるボードを使用している場合は、この情報を参考として使用してください。

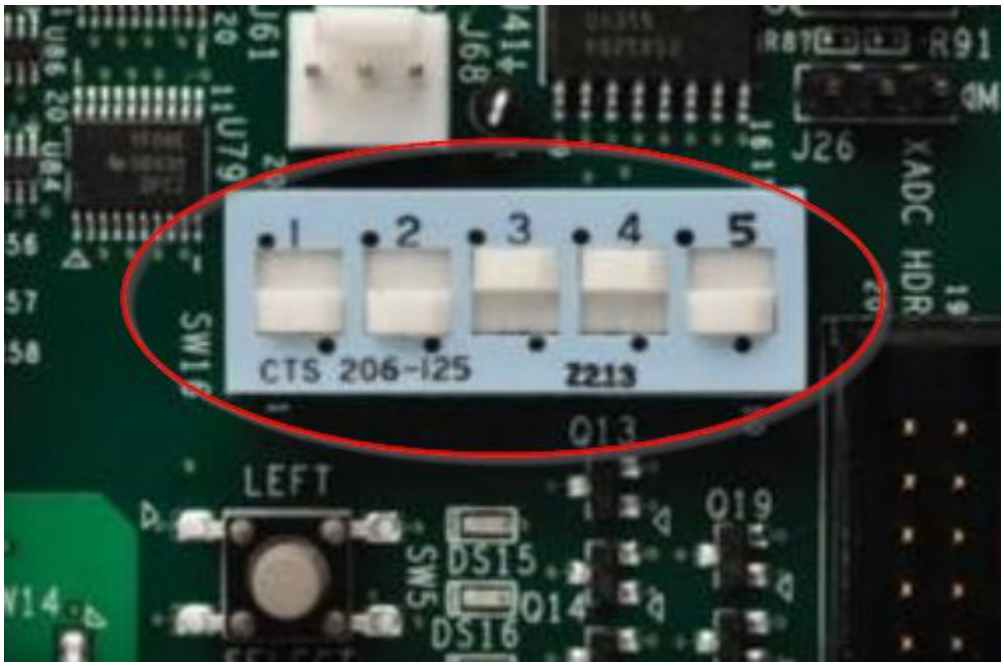
1. SD カードにコピー可能な行列乗算ビルド済みデザインを使用して、ボード設定、接続、ターミナル設定をテストします。SD カードをボードに挿入し、ボードに電源を投入し、ELF を実行して、USB UART 接続を介してボードに接続されたターミナルで行列乗算の出力を確認します。
2. SDSoC 環境を使用して単純な行列乗算アプリケーション例を作成し、この行列乗算関数がプログラマブル ロジック上のハードウェア アクセラレータ ブロックに変換されるようにします。これにより、ツールのインストールおよび環境設定が有効になります。
3. ボード上でサンプルを実行します。SDSoC 環境により、Linux ブートローダー、Linux カーネルとハードウェア アクセラレータとの通信に必要なドライバー、ファイル システム、アプリケーション ELF を含む SD カード イメージが生成されます。この SD カードを使用して ELF を実行し、出力を確認します。

SD カード ブートのボード コンフィギュレーション

SD カードからボードをブートするには、ボードのスイッチまたはジャンパー設定のいずれかを変更する必要があります。このセクションでは、ZC702 ボードの設定について説明します。異なるボードを使用している場合は、該当するボード リファレンス ガイドで適切なスイッチおよびジャンパー設定を確認してください。

1. 変更する必要があるのがジャンパーなのかスイッチなのかを特定します。

   これは、ZC702 ボードのバージョンによって異なります。



2. SD カードからブートするようにするには、ジャンパーまたはスイッチを次のように設定します。

   - リビジョン C またはそれ以前のボード：
     - J22 1-2 (ピン 1 と 2 を接続)
     - J25 1-2 (ピン 1 と 2 を接続)
   - リビジョン D またはそれ以降のボード：

     DIP スイッチ SW16 (ライトブルー/グレー) 位置 3 および 4 を 1 に設定

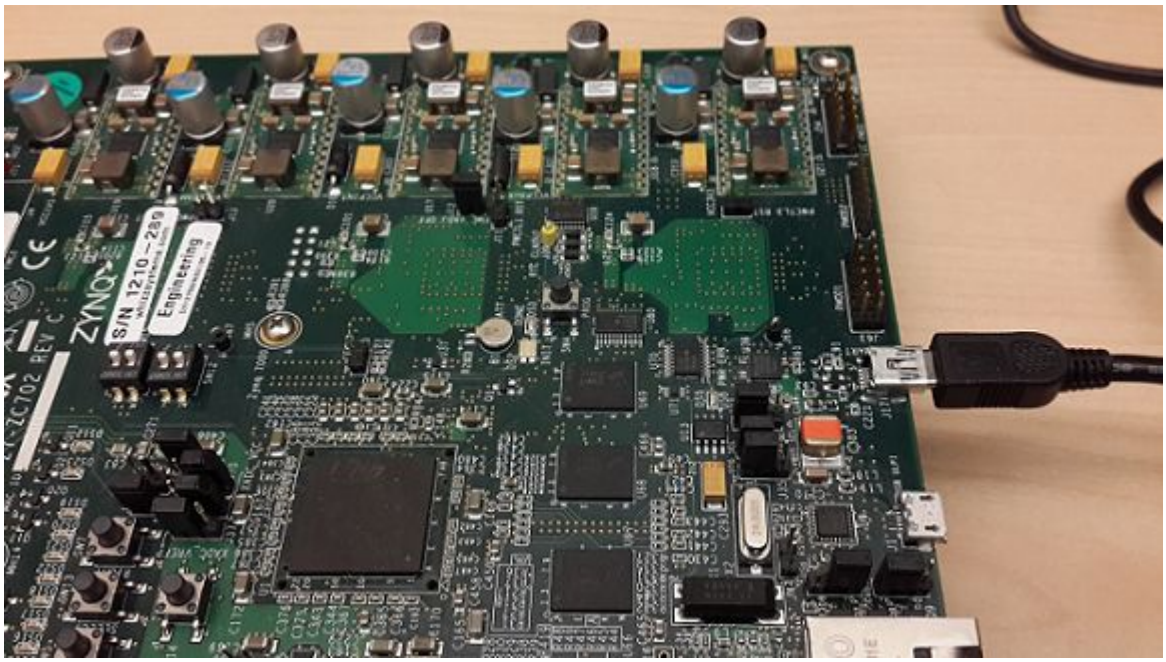
   ZC702 評価ボードの SD ブート設定は、『Zynq-7000 XC7Z020 All Programmable SoC 用の ZC702 評価ボード ユーザー ガイド』(UG850) および Wiki ページ「ビルド済みザイリンクス ZC702 イメージの起動」にも記述されています。

シリアル ターミナルへのボードの接続

ZC702 ボードをシリアル ターミナルに接続する際は、ボードの UAT ポートをシリアル ターミナルを実行するコンピューターに接続するために mini USB ケーブルが必要です。シリアル ターミナルは SDSoC IDE の一部として含まれます (メイン ウィンドウ下部の [Terminal 1] タブ)。

異なるボードを使用している場合は、該当するボード リファレンス ガイドで適切なケーブル タイプおよびコネクタと USB UART ドライバー インストールを確認してください。シリアル ターミナル設定の手順は同様です。

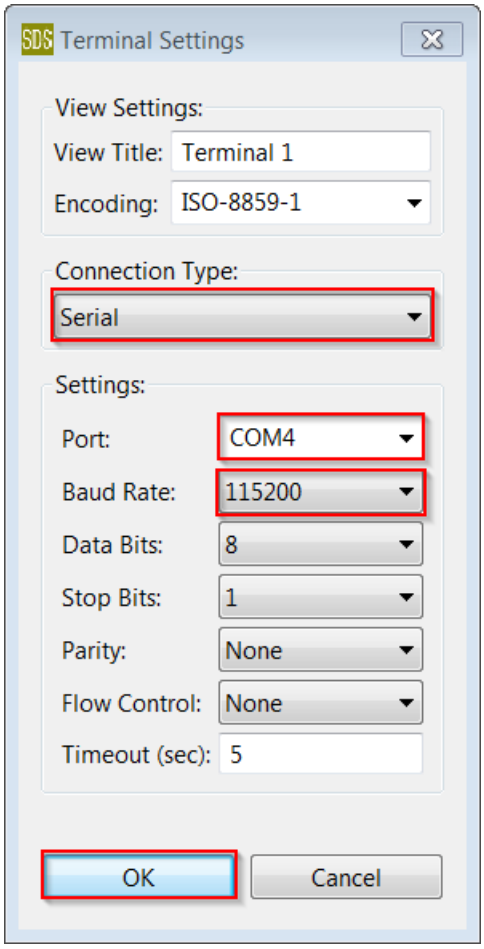
1. mini USB ケーブルを UART ポートに接続します。



2. シリアル ターミナル (PuTTY、Minicom、SDSoC 環境 SDK ターミナルまたは [Terminal] ビューなど) を設定します。SDSoC 環境を Windows ホストで実行している場合は、PuTTY、SDSoC 環境 SDK ターミナルまたは [Terminal] ビュー、Linux ホストで実行している場合は Minicom または SDSoC 環境 SDK ターミナルを推奨します。

- ボー レートを 115200 に設定します。
- Windows でシリアル ポートを COMn に設定します (n の番号は次の方法で確認可能)。
  - [スタート] → [コンピューター] を右クリックして、[プロパティ] をクリックします。
  - [デバイス マネージャー] をクリックして [ポート (COM と LPT)] を展開表示します。
  - [Silicon Labs CP210x USB to UART Bridge] という COM ポートを使用します。

注記 : [Terminal Settings] ダイアログ ボックスに正しい COM ポートが表示されない場合は、ボードが USB ポートに接続されて、オンになっているかどうかを確認してください。[File] → [Restart] をクリックして SDSoC 環境を再起動すると、COM ポートがリストに表示されるはずです。次に示すダイアログ ボックスは、[Window] → [Show] → [View] → [Other] をクリックし、[Terminal] → [Terminal] をクリックして起動した SDSoC の [Terminal] ビューからのものです。



3. ボードに電源を投入します。

   ボードは、UART を認識させてドライバーをインストールするため、Windows に mini-USB ケーブルを接続して、少なくとも 1 回は電源投入しておく必要があります。ボードの電源スイッチを切ってすぐに入れ直す必要があることもあります。ドライバーが読み込まれない場合は、http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx からダウンロードして手動でインストールしてください。

ビルド済みアプリケーションの実行

SDSoC™ 環境のインストール ディレクトリの `<path_to_install>/SDSoC/2015.4/samples` フォルダーには、複数のサンプルが含まれています。

`mmult_pipelined` サンプルには、行列乗算関数を呼び出して、`printf()` 文でその出力を stdout に表示する main アプリケーションを含んだ C++ ソース ファイルと、アプリケーションをビルドするための makefile、ビルド済みファイルを含む `sd_card_prebuilt` フォルダーなどが含まれます。`sd_card_prebuilt` フォルダーのファイルは、SDSoC 環境の使用前にボード、ボードの接続、ターミナル設定を有効にするために使用します。

`<path_to_install>/SDSoC/2015.4/samples/mmult_pipelined` フォルダーには `mmult_pipelined` サンプルが含まれ、次のようなディレクトリ構造になっています。

```
<path_to_install>/SDSoC/2015.4/samples/mmult_pipelined
    Makefile
    mmult.cpp
    mmult_accel.cpp
    mmult_accel.h
    sd_card_prebuilt
        BOOT.BIN
        README.txt          boot.bif
        devicetree.dtb          mmult.elf
        uImage
        uramdisk.image.gz
```

`mmult_pipelined` サンプルには、ZC702 ボード用のビルド済みアプリケーションが含まれます。異なるボードを使用している場合は、`<path_to_install>/SDSoC/2015.4/samples` で使用しているボード上で実行するアプリケーションがどこにあるかを確認し、アプリケーションをビルドします。各サンプルには、アプリケーションの実行方法を記述する README ファイルが含まれています。パートナー ボードまたはプラットフォームを使用している場合、ビルド済み SD カード アプリケーションが存在する可能性があります。ビルド済みアプリケーションが見つからない場合は、この手順はスキップします。

ZC702 ボードでビルド済みアプリケーションを実行するには、次の手順を使用します。

1. `sd_card_prebuilt` フォルダーの内容を SD カードのルート フォルダーにコピーします。

   SD カードは、NTFS ではなく FAT32 を使用してフォーマットする必要があります。
2. カードを ZC702 ボードの SD カード スロットに挿入します。
3. ジャンパーまたはスイッチが SD カードから起動するように設定されているかどうか確認します。「SD カード ブートのボード コンフィギュレーション」を参照してください。

4. ボードからのイーサネット ケーブルをネットワークに接続します。

   これはオプションです。接続すると、ネットワークに接続できるようになります。
5. シリアル ターミナルを設定します。「シリアル ターミナルへのボードの接続」を参照してください。
6. SD カードを挿入してケーブルを接続したら、ボードに電源を投入してシリアル ターミナル セッションを開始します。

   Done の LED が緑になって、Linux が起動されるはずです。
7. プロンプトに「`cd /mnt`」と入力します。

   これにより、アプリケーション ELF ファイルを含む SD カード フォルダーに移動します。
8. アプリケーション ELF を実行するには、「`./mmult.elf`」と入力します。
9. アプリケーションに run に関する情報と行列乗算の結果が表示されます。

   次のような出力が表示されます。

```
Testing mmult …
Average number of processor cycles for golden version: 182299
Average number of processor cycles for hardware version: 18685
TEST PASSED
```

ビルド済みアプリケーションを実行できる場合は、次のセクションに進んでください。

重要：

ビルド済みのアプリケーションが存在するのに実行できない場合は、次のセクションに進まないでください。ビルド済みアプリケーションが存在しない場合は、次のセクションの手順に従ってアプリケーションを構築し、ボード設定を確認してください。

ボードに電源を入れて、Done の LED が緑にならない場合は、ブートローダーがプログラマブル ロジックをコンフィギュレーションしていないことを意味します。ビルド済みの SD カード ファイルが SD カードのフォルダー内ではなく、ルート (最上位) ディレクトリにコピーされているかどうか、ファイル サイズが一致しているかどうかを確認してください。また、ジャンパー設定も確認してください。最初のボードが正しく動作しない場合は、別のボードを起動する SD カードを使用します。SD カードが NTFS ではなく FAT32 を使用してフォーマットされていることを確認してください。

ターミナルに Linux がブートされていることが表示されない場合は、ボー レートと COM ポート設定を確認します。USB UART ドライバーがインストールされているかどうかも確認してください。不明な場合は、アンインストールしてからインストールし直してください。

`sdscc -sds-pf-list` コマンドを使用すると、SDSoC に含まれるプラットフォームをリストできます。

サンプル アプリケーションのビルドと実行

これでサンプル デザイン `<path_to_installation>/SDSoC/2015.4/samples/mmult_pipelined` をビルドできるようになったので、ツールのインストールと環境設定を確認します。

`mmult_pipelined` サンプルは、ZC702 ボード上で実行するよう作成されています。この後の手順は、このボード用です。

異なるボードを使用している場合でも、makefile でそのボード用のプラットフォーム オプションを指定することにより (ZC702 (`-sds-pf zc702`) の代わりに ZC706 (`-sds-pf zc706`) を指定するなど)、このデザインを使用できる可能性があります。または、`<path_to_install>/SDSoC/2015.4/samples/README.txt` ファイルで使用しているボード上で実行可能なアプリケーションを確認するか、パートナー ボードおよびプラットフォームの場合はパートナーが提供するサンプルを使用します。

mmult_pipelined フォルダーには、行列乗算関数を呼び出して、printf() 文でその出力を stdout に表示する main アプリケーションを含んだ C++ ソース ファイルとが含まれます。

ユーザーの makefile により SDSoC 環境が起動され、ハードウェア アクセラレーターを含むハードウェア システムと、ハードウェアを使用するためのソフトウェア ライブラリと API が生成されます。最上位の makefile には、アプリケーション ソース ファイルからオブジェクト ファイル (.o) をビルドするためのターゲットが含まれます。これらをリンクして、アプリケーション ELF ファイルを作成して、SD カード イメージを生成します。

次のセクションに、サポートされるホストでアプリケーションをビルドする方法を説明します。

- 「Linux ホストでのアプリケーションのビルド」
- 「Windows ホストでのアプリケーションのビルド」

make コマンドが問題なく終了したら、ZC702 ボードのアプリケーションを実行します。詳細は、「サンプル アプリケーションの実行」を参照してください。

Linux ホストでのアプリケーションのビルド

Linux ホストでサンプル アプリケーションをビルドする手順は、次のとおりです。

1. インストーラーで作成された設定スクリプトを実行して、SDSoC 環境を実行する環境を設定します。
   - C シェル

     ```
     % source <path_to_install>/SDSoC/2015.4/settings64.csh
     ```

   - Bourne シェルまたは Bash シェル

     ```
     % . <path_to_install>/SDSoC/2015.4/settings64.sh
     ```

2. mmult_pipelined フォルダーを書き込み権のある作業ディレクトリにコピーします。

   ```
   % cp -r <path_to_install>/SDSoC/2015.4/samples/mmult_pipelined .
   % cd mmult_pipelined
   ```

3. アプリケーションおよび SD カード イメージをビルドします。

   ```
   % which sdscc # displays path to the sdscc tool
   % make
   ```

ビルドには多少の時間がかかります。終了すると、sd_card というフォルダーに、ZC702 ボードで Linux を開始して ELF アプリケーションを実行するのに必要な ELF ファイルとブート イメージが生成されます。

Windows ホストでのアプリケーションのビルド

Windows ホストでサンプル アプリケーションをビルドする手順は、次のとおりです。

1. [SDSoC 2015.4 Terminal] ショートカットを実行します。

   これにより、「Linux ホストでのアプリケーションのビルド」に説明されているコマンドを使用して環境が設定されます。

2. Windows コマンド シェル プロンプトの‘>’の後に次のコマンドを入力します。プロンプトに REM で始まるコメントは入力しないでください。

```
> cp -r <path_to_sdsoc_install>\SDSoC\2015.4\samples\mmult_pipelined
<path_to_user_folder>\mmult_pipelined
> cd <path_to_user_folder>\mmult_pipelined
> REM displays path to the sdscc tool
> where sdscc
> make
```

注記： cp コマンドは Linux コマンドで、SDSoC 環境から引き継がれたユーザー環境の一部です。

Linux シェル コマンドのサブセットも使用できます。または、次の例のような Windows コマンドを使用できます。

```
> xcopy <path_to_install>\SDSoC\2015.4\samples\mmult_pipelined
<path_to_user_folder>\mmult_pipelined /s /e /h
> cd <path_to_user_folder>\mmult_pipelined
> make
```

サンプル アプリケーションの実行

アプリケーションを生成したら、ZC702 ボードで実行します。詳細は、「ビルド済みアプリケーションの実行」を参照してください。次のサマリでは、前に実行した手順 (ジャンパーとスイッチ設定、イーサネット ケーブル接続、シリアル ターミナルの設定) は省略しています。

1. sd_card フォルダーの内容を SD カードにコピーします。
2. カードを ZC702 ボードの SD カード スロットに挿入し、ジャンパーまたはスイッチが SD カードから起動するように設定されているかどうかを確認します。「SD カード ブートのボード コンフィギュレーション」を参照してください。
3. mini USB を使用してボードとコンピューターを接続します。
4. SD カードを挿入してケーブルを接続したら、ボードに電源を投入してシリアル ターミナル セッションを開始します。「シリアル ターミナルへのボードの接続」を参照してください。

   Linux の起動を確認します。
5. プロンプトに「cd /mnt」と入力します。

   これにより、アプリケーション ELF ファイルを含む SD カード フォルダーに移動します。
6. アプリケーション ELF を実行するには、「./mmult.elf」と入力します。
7. アプリケーションに run に関する情報と行列乗算の結果が表示されます。

   次のような出力が表示されます。

```
Testing mmult …
Average number of processor cycles for golden version: 182299
Average number of processor cycles for hardware version: 18685
TEST PASSED
```

アプリケーションがボードで問題なく実行されたら、samples フォルダーのその他のデザインを実行したり変更したりしてみてください。『SDSoC 環境ユーザー ガイド』(UG1027) には、さまざまなインプリメンテーション例の実行を向上するための手法が示されています。

重要： ボードが正しく設定されているのにアプリケーションを実行できない場合は、ザイリンクス サポートまでご連絡ください。

第 2 章

チュートリアル：プロジェクトの作成、ビルド、実行

このチュートリアルでは、SDSoC 環境でテンプレートを使用して新しいプロジェクトを作成し、ハードウェア インプリメンテーション用の関数をマークし、ハードウェアでインプリメントされるデザインをビルドし、ZC702 でプロジェクトを実行します。

注記： チュートリアルは各手順に分けられ、それぞれで大まかな手順が説明された後、細かい手順が説明されていますので、スキル レベルに合った方の手順を参照してください。大まかな手順を終了するのにヘルプが必要な場合は詳細な手順を参照したり、細かい手順を飛ばして次の大まかな手順に進んだりできます。

チュートリアルの目標

このチュートリアル (`lab1`) を終了すると、次ができるようになります。

- 多くの使用可能なプラットフォームおよびプロジェクト テンプレートから、ユーザー アプリケーション用の新しい SDSoC 環境プロジェクトを作成
- ハードウェア インプリメンテーション用の関数をマーク
- ハードウェアにインプリメントされる関数を含むビットストリームと、このハードウェアでインプリメントされる関数を開始する実行ファイルを生成するプロジェクトをビルド

「新規プロジェクトの作成」

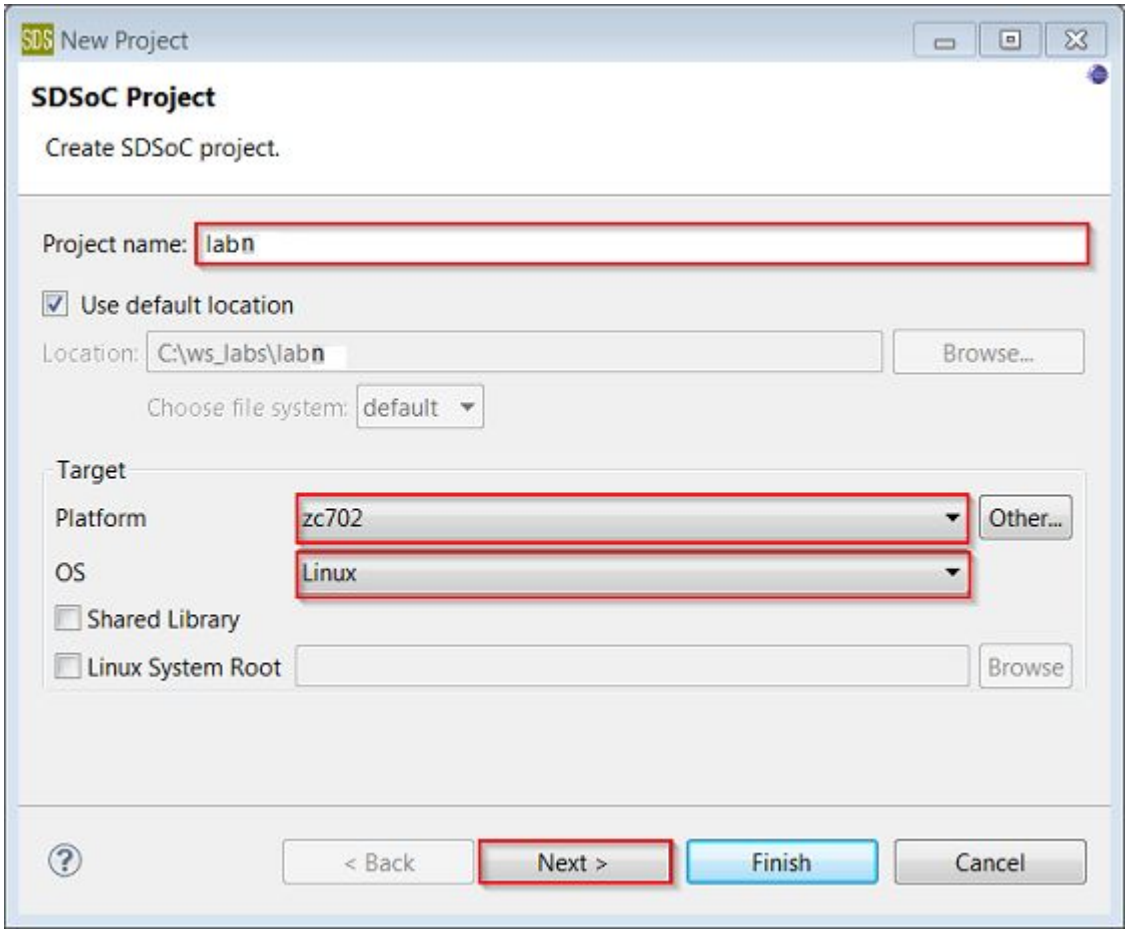
「質問およびその他の演習」

新規プロジェクトの作成

SDSoC™ 環境で行列乗算および加算を実行するプロジェクトを作成するには、次の手順に従います。

1. デスクトップ アイコンをダブルクリックするか [スタート] メニューを使用して、SDSoC 環境を起動します。
2. [Workspace Launcher] ダイアログ ボックスが表示されます。[Browse] をクリックしてプロジェクトを保存するワークスペース フォルダーを選択し、[OK] をクリックします。
3. SDSoC 環境のメイン ウィンドウが表示されます。新しいワークスペースを作成した場合は、[Welcome] タブが表示されます。このタブには、[Create SDSoC Project] などのプロジェクトを開始するためのリンクと、[SDSoC User Guide] などの資料およびチュートリアルにアクセスするリンクがあります。[Welcome] タブは、[X] をクリックして閉じるか、[Minimize] アイコンをクリックして最小化できます。

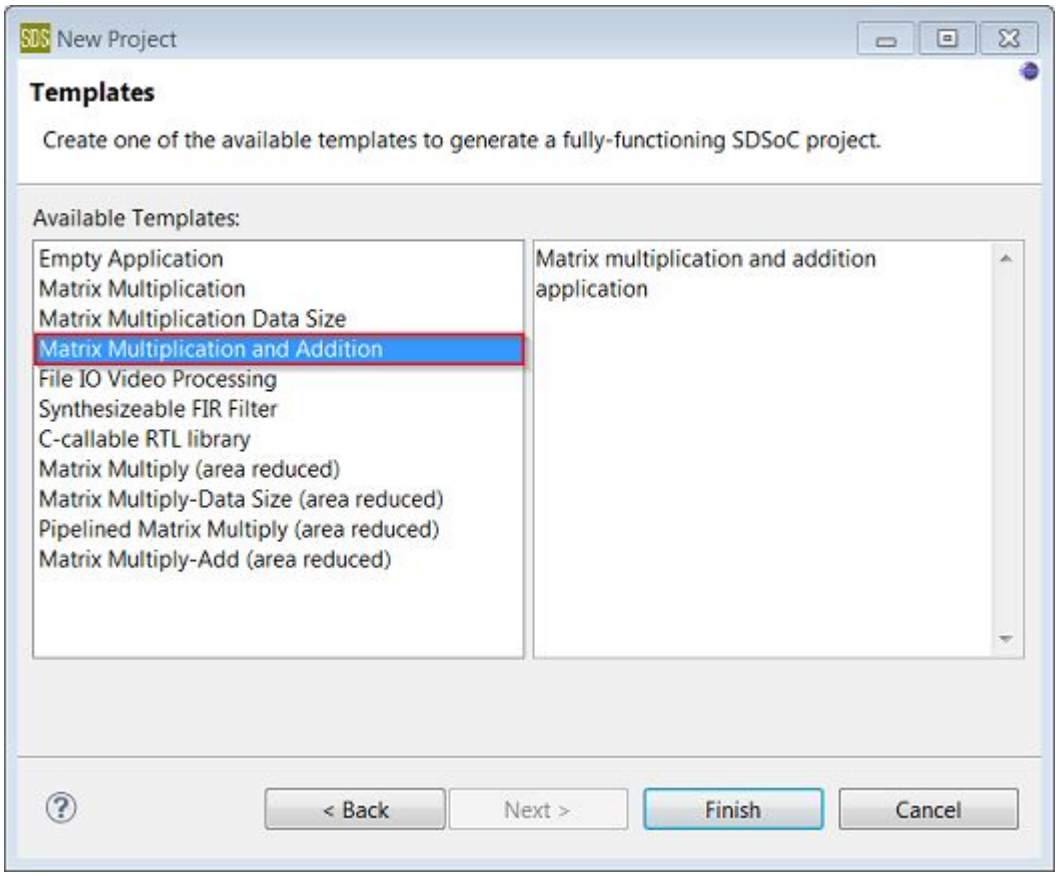
4. [Welcome] タブで [Create SDSoC Project] をクリックするか、SDSoC メニュー バーから [File] → [New] → [SDSoC Project] をクリックします。



5. プロジェクト名を指定します。上図では [Project name] に「`labn`」と示されていますが、チュートリアルを実行する際は、最初のチュートリアルの場合は `lab1`、2 番目のチュートリアルの場合は `lab2` のように指定します。
6. [Platform] ドロップダウンの使用可能なプラットフォームから [zc702] を選択します。
7. [OS] ドロップダウン リストから [Linux] を選択します。
8. [Next] をクリックします。

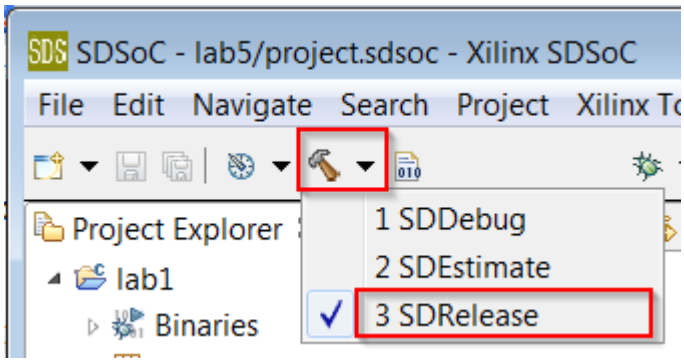
   選択したプラットフォーム用のソース コード例をリストする [Templates] ページが表示されます。

9. [Available Templates] のリストから [Matrix Multiplication and Addition] を選択し、[Finish] をクリックします。

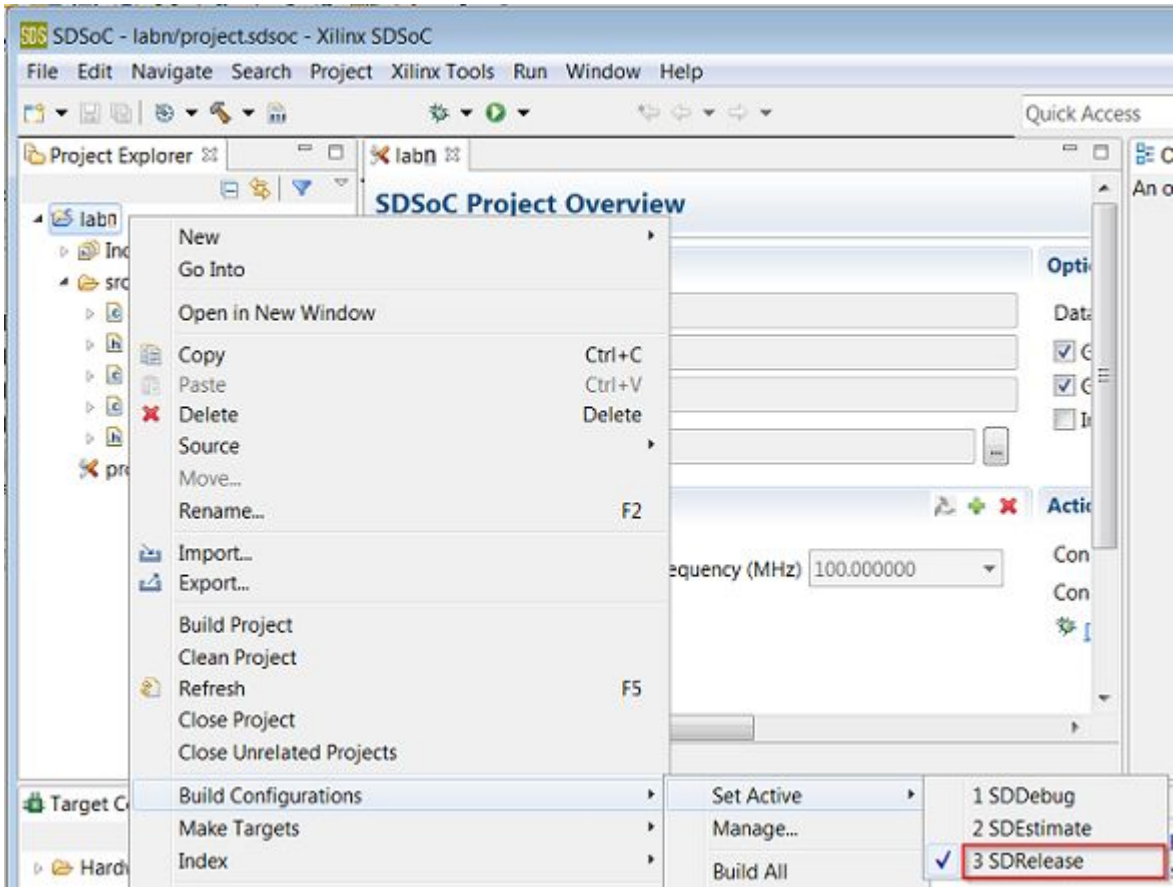


10. 標準のビルド コンフィギュレーションは [SDDebug]、[SDEstimate]、[SDRelease] です。最高のランタイム パフォーマンスを得るには、プロジェクトを選択し、[Build] アイコンの右側にある下向き矢印をクリックして [SDRelease] を選択するか、プロジェクトを右クリックして [Build Configurations] → [Set Active] → [SDRelease] をクリックして SDRelease コンフィギュレーションを使用します。SDRelease ビルド コンフィギュレーションでは、SDDebug ビルド コンフィギュレーションよりも高いコンパイラ最適化設定が使用されます。

[Build] アイコンには、ビルド コンフィギュレーションを選択するドロップダウン メニューがあります。[Build] アイコンをクリックすると、プロジェクトがビルドされます。



[SDSoC Project Overview] パネルにプロジェクト設定のサマリが表示されます。

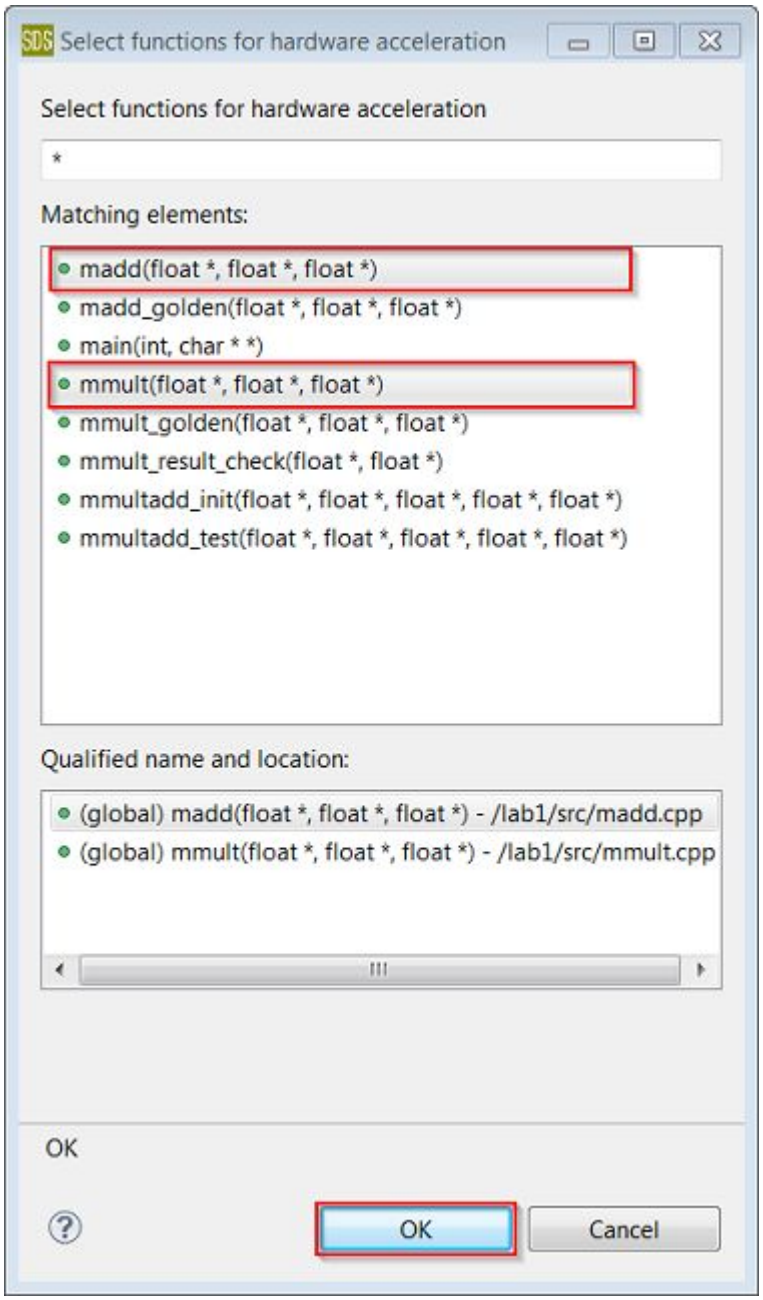


SDSoC アプリケーションをビルドする際は、ビルド コンフィギュレーション (ツール設定、フォルダー、ファイルなどのコレクション) を使用します。各ビルド コンフィギュレーションの目的は異なります。SDDebug ビルドでは、ELF (コンパイルおよびリンク済みプログラム) にデバッガーを実行するのに必要な 追加情報を含めてアプリケーションがビルドされます。ELF ファイルのデバッグ情報により、ファイル サイズが増加し、アプリケーション情報が表示されるようになります。SDRelease ビルドでは、同じ ELF ファイルが SDDebug オプションとして提供されますが、デバッグ情報が含まれない点が異なります。SDEstimate を使用すると、SDSoC 環境がアプリケーションのパフォーマンス予測を実行するモードで実行され、別の設定および手順が必要となります。詳細は、「チュートリアル : システム パフォーマンスの予測」を参照してください。

ハードウェア インプリメンテーション用の関数のマーク

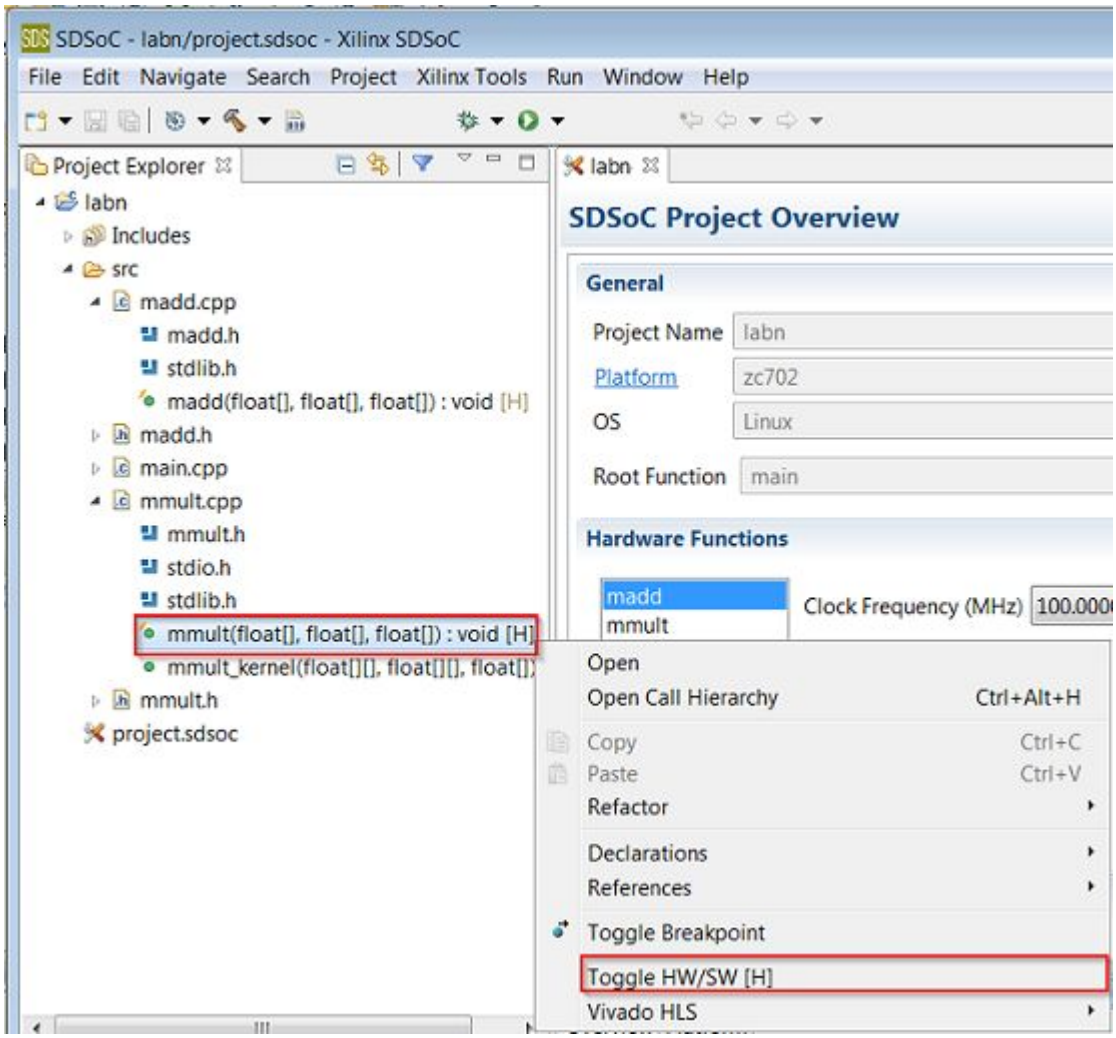
このアプリケーションには、2 つのハードウェア関数が含まれます。1 つは `mmult` で、2 つの行列を乗算して行列積を算出します。もう 1 つは `madd` で、2 つの行列を加算して行列和を算出します。これらのハードウェア関数がまとめられ、行列の乗加算関数を計算します。`mmult` と `madd` の両方のハードウェア関数をハードウェアでインプリメントされるように指定します。

1. [SDSoC Project Overview] では、プロジェクトの値を設定できます。[<name of project>] タブをクリックし (タブが開いていない場合は、[Project Explorer] タブで project.sdsoc ファイルをダブルクリック)、[Hardware Functions] セクションで [Add Hardware Function] アイコン をクリックしてハードウェア関数を指定します。



2. [Matching elements] リストで Ctrl キーを押しながら mmult と madd 関数をクリックして選択します。[OK] をクリックして、両方の関数を [Hardware Functions] セクションに追加します。

   または、[Project Explorer] タブで mmult.cpp および madd.cpp を展開表示し、mmult および madd 関数を右クリックして [Toggle HW/SW] をクリックします。関数が既にハードウェア用にマークされている場合は、[Toggle HW/SW [H]] と表示されます。ソース ファイルをエディターで開いている場合は、[Outline] タブでハードウェア関数を選択することもできます。



注意 : すべての関数がハードウェアにインプリメントできるわけではありません。詳細は、『SDSoC 環境ユーザー ガイド』(UG1027) の「コード ガイドライン」を参照してください。

ハードウェア アクセラレータを使用したデザインのビルド

プロジェクトをビルドして実行ファイル、ビットストリーム、SD カード ブート イメージを生成するには、次の手順に従います。

1. [Project Explorer] タブで [lab1] を右クリックし、[Build Project] をクリックします。

   SDSoC™ システム コンパイラの stdout が [Console] タブに表示されます。ハードウェア用に選択された関数が Vivado® HLS を使用して IP ブロックにコンパイルされ、選択したベース プラットフォームに基づいて生成された Vivado ツール ハードウェア システムに統合されます。この後、システム コンパイラにより Vivado 合成、配置配線ツールが起動されてビットストリームがビルドされ、ARM GNU コンパイラとリンカーが起動されて、アプリケーション ELF 実行ファイルが生成されます。

2. Vivado 合成には時間がかかるので、プロジェクトをビルドする代わりに、次の手順でビルド済みのファイルをワークスペースにインポートできます。

   a. [File] → [Import] をクリックし、[General] → [Existing Projects into Workspace] を選択して [Next] をクリックします。

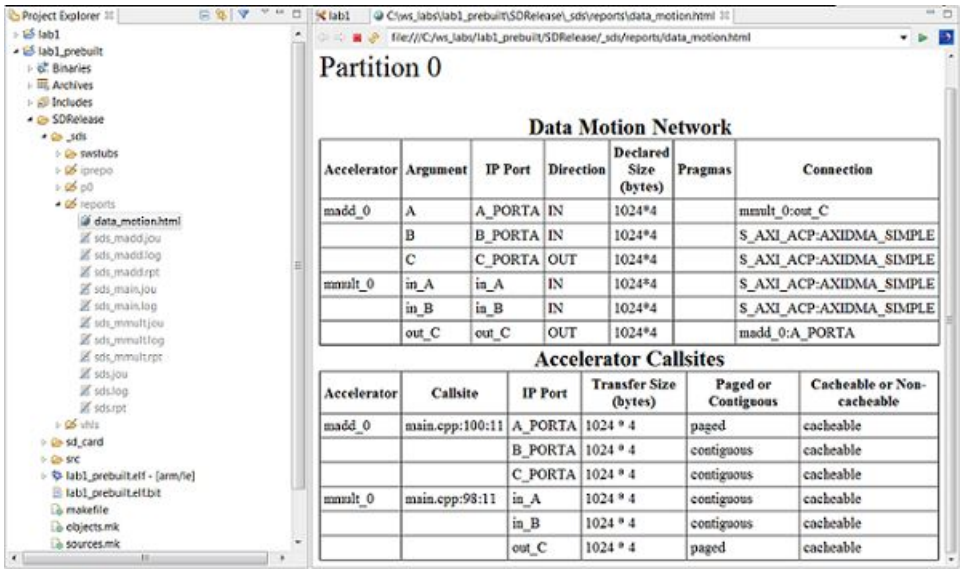
   b. [Select archive file] をオンにして [Browse] をクリックし、`<path to install>/SDSoC/2015.4/docs/labs/lab1_prebuilt.zip` を選択します。

   c. [lab1_prebuilt.zip] を選択し、[開く] をクリックします。

   d. [Finish] をクリックします。

3. [SDSoC Project Overview] の [Reports] パネルで [Data motion] をクリックしてデータ モーション ネットワーク レポートを表示します。

   このレポートには、SDSoC 環境で実行された接続と、ハードウェアにインプリメントされた各関数のデータ転送タイプが示されます。詳細は、「チュートリアル : システム最適化」を参照してください。



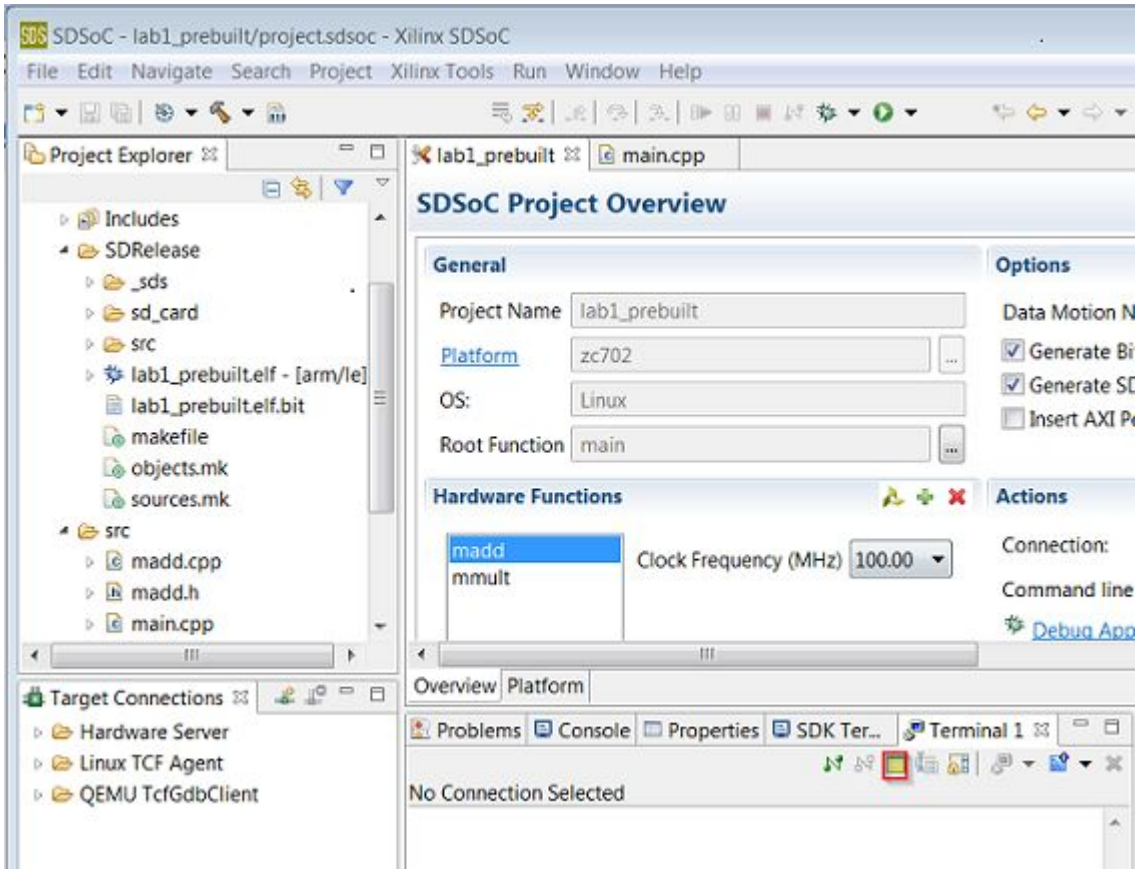
4. `lab1_prebuilt/SDRelease/_sds/swstubs/mmult.cpp` を開き、SDSoC システム コンパイラにより元の `mmult` 関数が `cf_send` および `cf_receive` を使用した FPGA に対して入出力転送を実行する `_p0_mmult_0` という関数に置き換えられたことを確認します。`mmult` への呼び出しも `lab1_prebuilt/SDRelease/_sds/swstubs/main.cpp` 内の `_p0_mmult_0` に置き換えられます。SDSoC システム コンパイラで、これらの記述し直されたソース ファイルを使用してハードウェア関数にアクセスする ELF がビルドされます。

プロジェクトの実行

ZC702 ボードでプロジェクトを実行する手順は、次のとおりです。

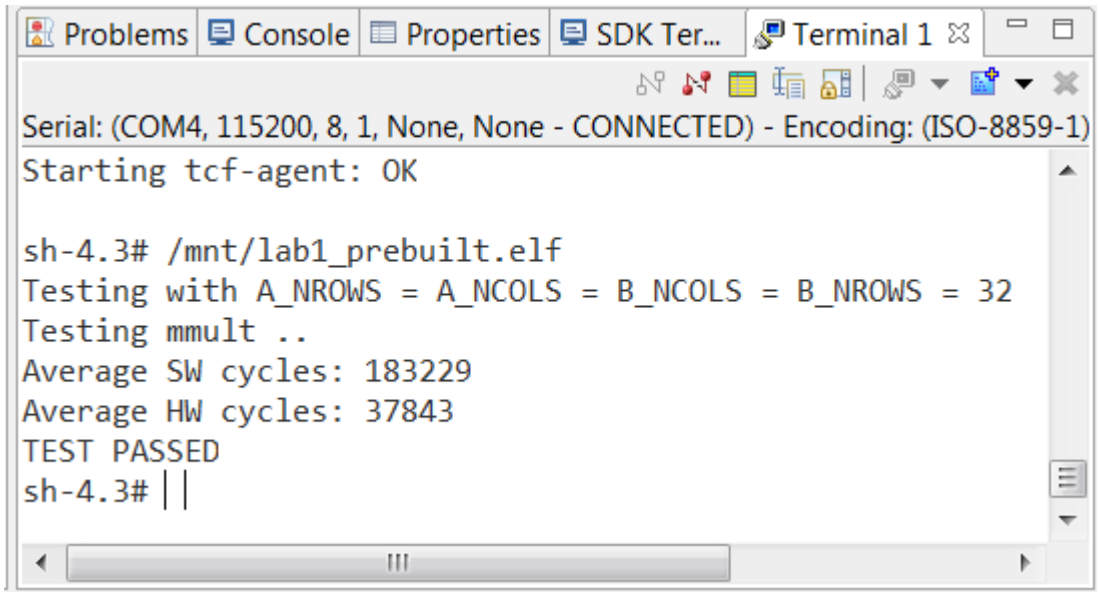
1. [Project Explorer] タブで `lab1_prebuilt/SDRelease` を展開表示し、`sd_card` ディレクトリ内のすべてのファイルを SD カードのルート ディレクトリにコピーします。

2. SD カードを ZC702 に挿入し、ボードに電源を投入します。
3. SDSoC 環境の [Terminal] タブのシリアル ターミナルからボードに接続します。黄色のパッド アイコンをクリックして [Terminal Settings] ダイアログ ボックスを開きます。



4. シリアル ターミナルを設定します。「シリアル ターミナルへのボードの接続」を参照してください。

5. ボードが起動したら、Linux プロンプトでアプリケーションを実行します。「`/mnt/lab1_prebuilt.elf`」と入力します。



質問およびその他の演習

次の質問に答えて、理解度を確認してください。

- ハードウェアにインプリメントできる関数の数がデバイスによって違うのはなぜですか。
- ハードウェアに mmult および madd カーネルをインプリメントすると、どれくらいスピードアップしますか。
- SDSoC™ システム コンパイラにより起動されるサブツールは何ですか。
- `SDRelease/_sds` フォルダーの `report` フォルダーを確認してください。このフォルダーには、複数のログ ファイルとレポート ファイル (`.rpt`) があり、ビルドにより起動されたすべてのツールからの詳細なログおよびレポートが含まれます。
- Vivado® IP インテグレーターについて詳しい場合は、[Project Explorer] タブで `SDRelease/_sds/p0/ipi/zc702.xpr` をダブルクリックしてください。これは、アプリケーション ソース コードから生成されたハードウェア デザインです。ブロック図を開いて、生成された IP ブロックを確認してみてください。

回答

- プログラマブル ロジックの量は、デバイスによって異なります。大型デバイスの方が小型デバイスよりもハードウェアに多数の関数をインプリメントできます。
- スピードは、約 4.3 倍速くなります。プロセッサで実行されるアプリケーションは 18 万サイクルかかりますが、プロセッサと FPGA 両方で実行されるアプリケーションは 4 万 1 千サイクルかかります。
- `sdscc`、`sds++`、`arm-xilinx-linux-gnueabi-gcc`、`arm-xilinx-linux-gnueabi-g++`、`vivado_hls`、`vivado`、`bootgen`
  - `sdscc` : C 言語ソースをコンパイルするのに使用されます。
  - `sds++` : C++ 言語をコンパイルするのに使用されるほか、`sdscc` と `sds++` で作成されるオブジェクト ファイルをリンクするのにも使用されます。
  - `arm-xilinx-linux-gnueabi-gcc` : `sdscc` から呼び出され、プロセッサをターゲットにする C 言語ソースのオブジェクト コードを生成します。
  - `arm-xilinx-linux-gnueabi-g++` : `sds++` から呼び出され、プロセッサをターゲットにする C++ 言語ソースのオブジェクト コードを生成するほか、すべてのオブジェクト ファイルをリンクして、プロセッサで実行する実行ファイルを作成します。
  - `vivado_hls` : `sdscc/sds++` から呼び出され、ハードウェア インプリメンテーション用にマークされている C/C++ 関数の RTL コードを生成します。
  - `vivado` : `sds++` から呼び出され、ビットストリームを生成します。
  - `bootgen` : `sds++` から呼び出され、チップの PL または FPGA ロジック部分のビットストリームと共に、プロセッサで実行される実行ファイルを含むブータブル イメージを作成します。

第 3 章

チュートリアル : システム最適化

このチュートリアルでは、SDSoC 環境で生成されたハードウェア/ソフトウェア システムを最適化するためにコードを変更する方法について説明します。ビルド エラーの詳細を確認して、コードを修正する方法についても説明します。

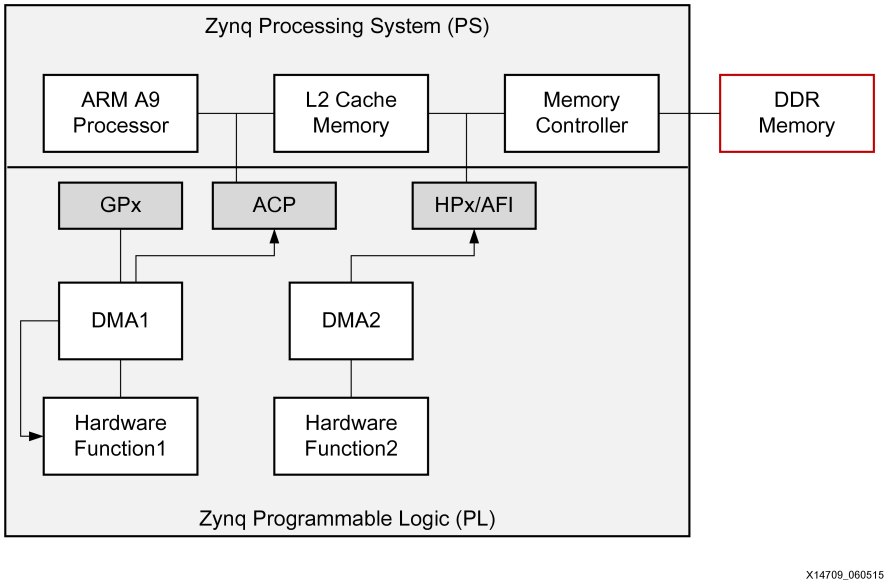
注記 : チュートリアルは各手順に分けられ、それぞれで大まかな手順が説明された後、細かい手順が説明されていますので、スキル レベルに合った方の手順を参照してください。大まかな手順を終了するのにヘルプが必要な場合は詳細な手順を参照したり、細かい手順を飛ばして次の大まかな手順に進んだりできます。

システム ポートおよび DMA の概要

Zynq®-7000 All Programmable SoC デバイス システムでは、ARM A9 プロセッサがアクセスするメモリに、オンチップ キャッシュと大型のオフチップ DDR メモリの 2 レベルがあります。プログラマブル ロジック側からは、ハードウェア デザインが作成されます。このハードウェア デザインには、ハードウェア関数がシステム インターフェイス ポートを介してプロセッサ システム メモリにデータを直接読み書きできるように、ダイレクト メモリ アクセス (DMA) ブロックが含まれることがあります。

次の簡略化された図に示すように、Zynq デバイスのプロセッシング システム (PS) ブロックには 3 種類のシステム ポートが含まれ、プロセッサ メモリから Zynq デバイスのプログラマブル ロジックにデータを転送するために使用されます。これらの 3 種類のシステム ポートは、ハードウェアがコヒーレント方式でプロセッサの L2 キャッシュに直接アクセスできるようにするアクセラレータ コヒーレンシ ポート (ACP)、Asynchronous FIFO Interface (AFI) を使用してプロセッサ キャッシュをバイパスしてハードウェアから DDR メモリまたはオンチップ メモリにダイレクト バッファー アクセスを提供するハイ パフォーマンス ポート 0 ～ 3 (HP0 ～ 3)、およびプロセッサがハードウェア レジスタに対して読み出し/書き込みを実行できるようにする汎用 I/O (GP0/GP1) です。

図 3-1 : メモリ アクセス ポートとメモリを示した Zynq + DDR の簡略図



ARM A9 プロセッサで実行されるソフトウェアがハードウェア関数を呼び出す場合、実際には SDSoC 環境で生成されたスタブ関数が呼び出され、3 種類のシステム ポート (GPx、ACP、AFI) を介してプロセッサ メモリからデータをハードウェア関数に送信し、ハードウェア関数からプロセッサ メモリにデータを戻す下位ドライバーが呼び出されます。

次の表に、これらのシステム ポートとその特性を示します。SDSoC 環境では、データ転送に最適なシステム ポートが自動的に選択されますが、プラグマを使用してこの選択を変更することもできます。

システム ポート	特性
ACP	プロセッサおよびハードウェア関数が共有メモリと同じ高速キャッシュ メモリにアクセス。
AFI (HPx)	ハードウェア関数が DDR からデータを読み出す前にドライバーにより DDR へのキャッシュをフラッシュする必要あり。
GPx	プロセッサがハードウェア関数のデータを直接読み出し/書き込み。大型データ転送には不向き。

チュートリアルの目標

このチュートリアル (lab2) を終了すると、次ができるようになります。

- プラグマを使用してデータ転送用の ACP または AFI ポートを選択
- SDSoC 環境のエラー検出とレポート機能を確認

その他の演習を実行すると、次も学ぶことができます。

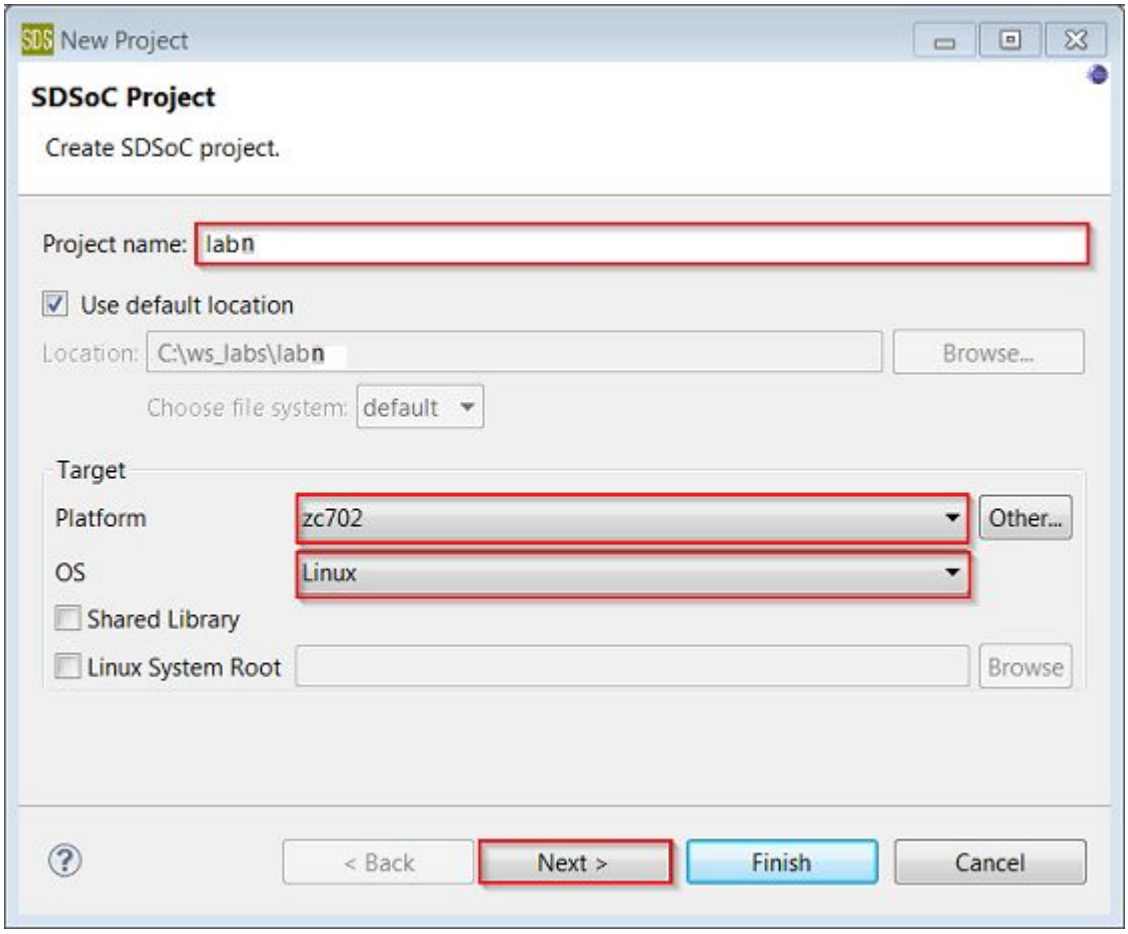
- プラグマを使用してハードウェア関数の引数に別のデータ ムーバーを選択
- `sds_alloc()` の使用について理解
- プラグマを使用してハードウェア関数から転送またはハードウェア関数へ転送されるデータ エレメント数を制御

新規プロジェクトの作成

SDSoC™ 環境で行列乗算および加算を実行するプロジェクトを作成するには、次の手順に従います。

1. デスクトップ アイコンをダブルクリックするか [スタート] メニューを使用して、SDSoC 環境を起動します。
2. [Workspace Launcher] ダイアログ ボックスが表示されます。[Browse] をクリックしてプロジェクトを保存するワークスペース フォルダーを選択し、[OK] をクリックします。
3. SDSoC 環境のメイン ウィンドウが表示されます。新しいワークスペースを作成した場合は、[Welcome] タブが表示されます。このタブには、[Create SDSoC Project] などのプロジェクトを開始するためのリンクと、[SDSoC User Guide] などの資料およびチュートリアルにアクセスするリンクがあります。[Welcome] タブは、[X] をクリックして閉じるか、[Minimize] アイコンをクリックして最小化できます。

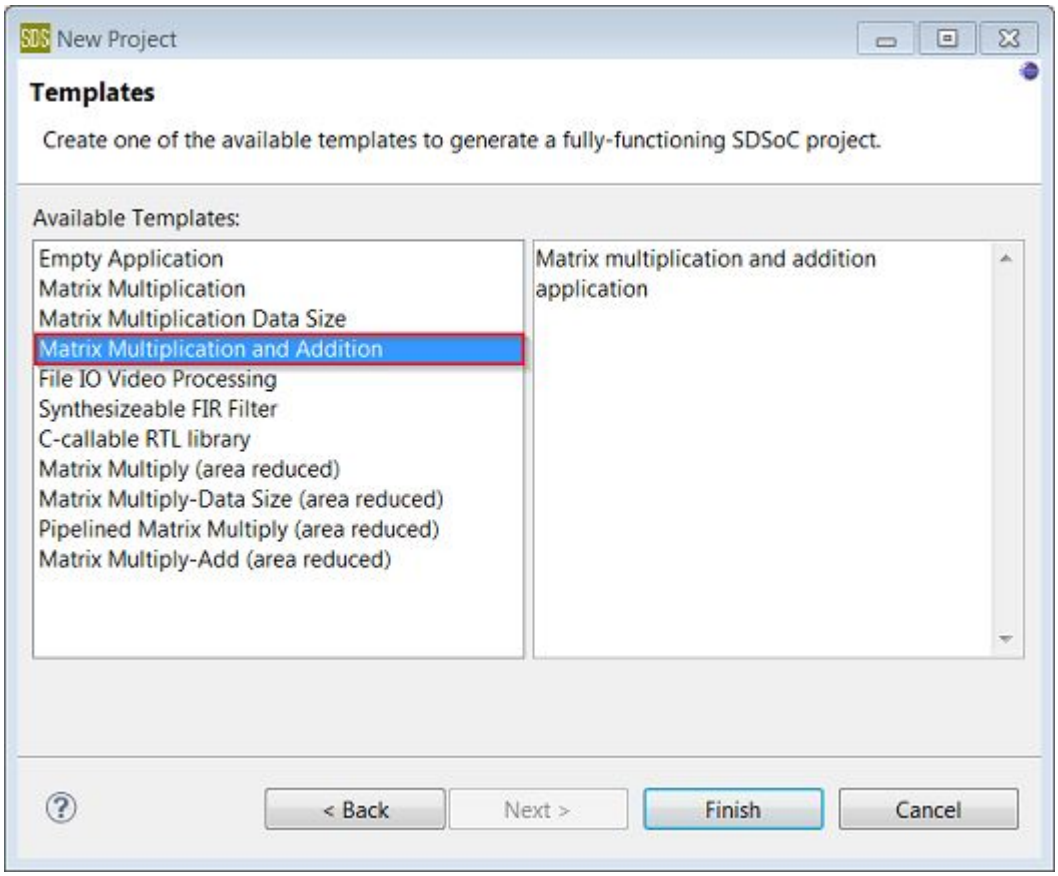
4. [Welcome] タブで [Create SDSoC Project] をクリックするか、SDSoC メニュー バーから [File] → [New] → [SDSoC Project] をクリックします。



5. プロジェクト名を指定します。上図では [Project name] に「`labn`」と示されていますが、チュートリアルを実行する際は、最初のチュートリアルの場合は `lab1`、2 番目のチュートリアルの場合は `lab2` のように指定します。
6. [Platform] ドロップダウンの使用可能なプラットフォームから [zc702] を選択します。
7. [OS] ドロップダウン リストから [Linux] を選択します。
8. [Next] をクリックします。

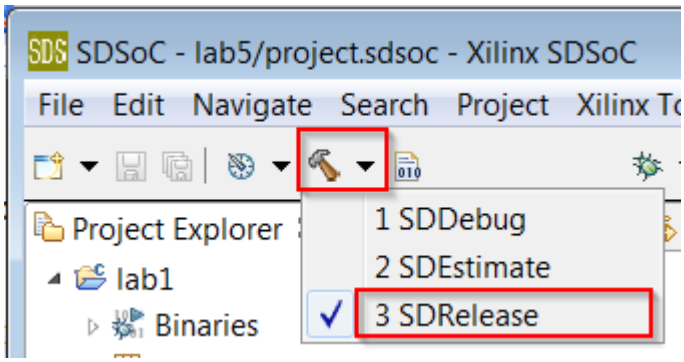
   選択したプラットフォーム用のソース コード例をリストする [Templates] ページが表示されます。

9. [Available Templates] のリストから [Matrix Multiplication and Addition] を選択し、[Finish] をクリックします。

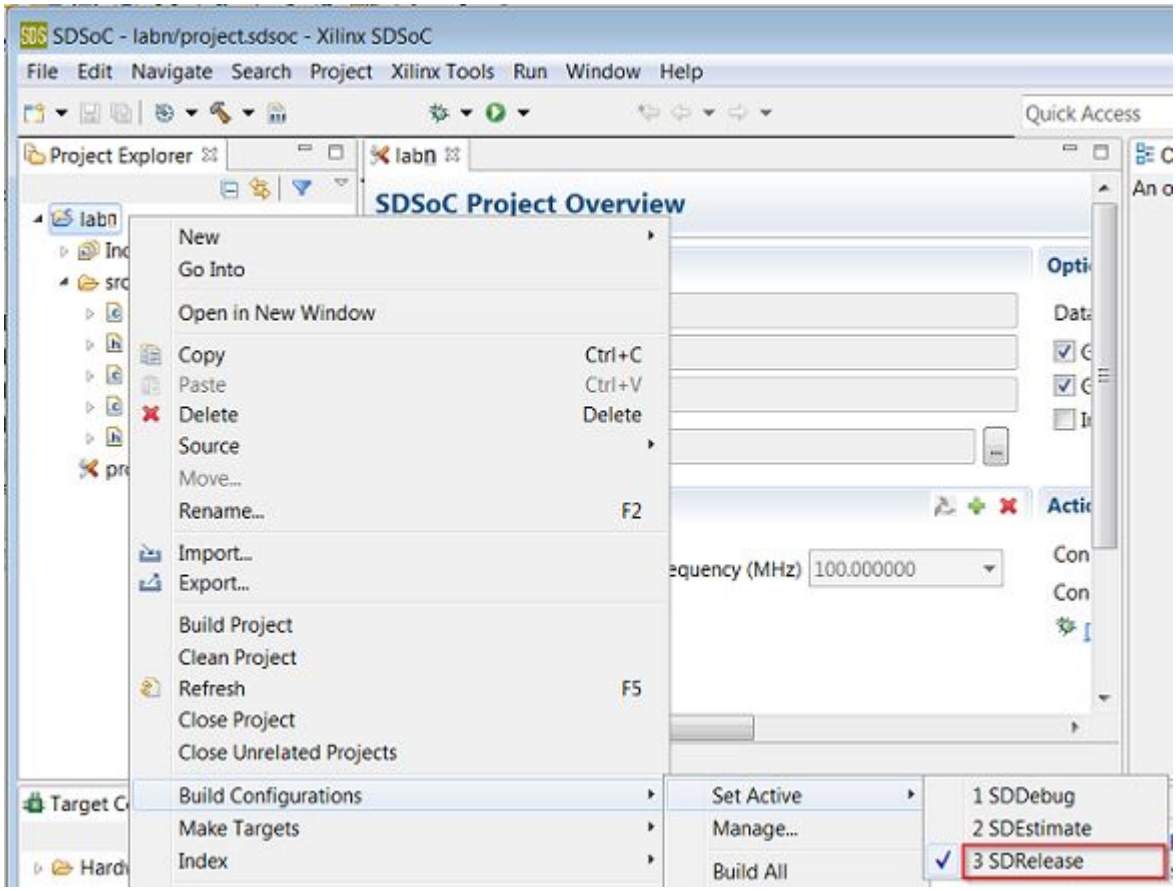


10. 標準のビルド コンフィギュレーションは [SDDebug]、[SDEstimate]、[SDRelease] です。最高のランタイム パフォーマンスを得るには、プロジェクトを選択し、[Build] アイコンの右側にある下向き矢印をクリックして [SDRelease] を選択するか、プロジェクトを右クリックして [Build Configurations] → [Set Active] → [SDRelease] をクリックして SDRelease コンフィギュレーションを使用します。SDRelease ビルド コンフィギュレーションでは、SDDebug ビルド コンフィギュレーションよりも高いコンパイラ最適化設定が使用されます。

[Build] アイコンには、ビルド コンフィギュレーションを選択するドロップダウン メニューがあります。[Build] アイコンをクリックすると、プロジェクトがビルドされます。



[SDSoC Project Overview] パネルにプロジェクト設定のサマリが表示されます。

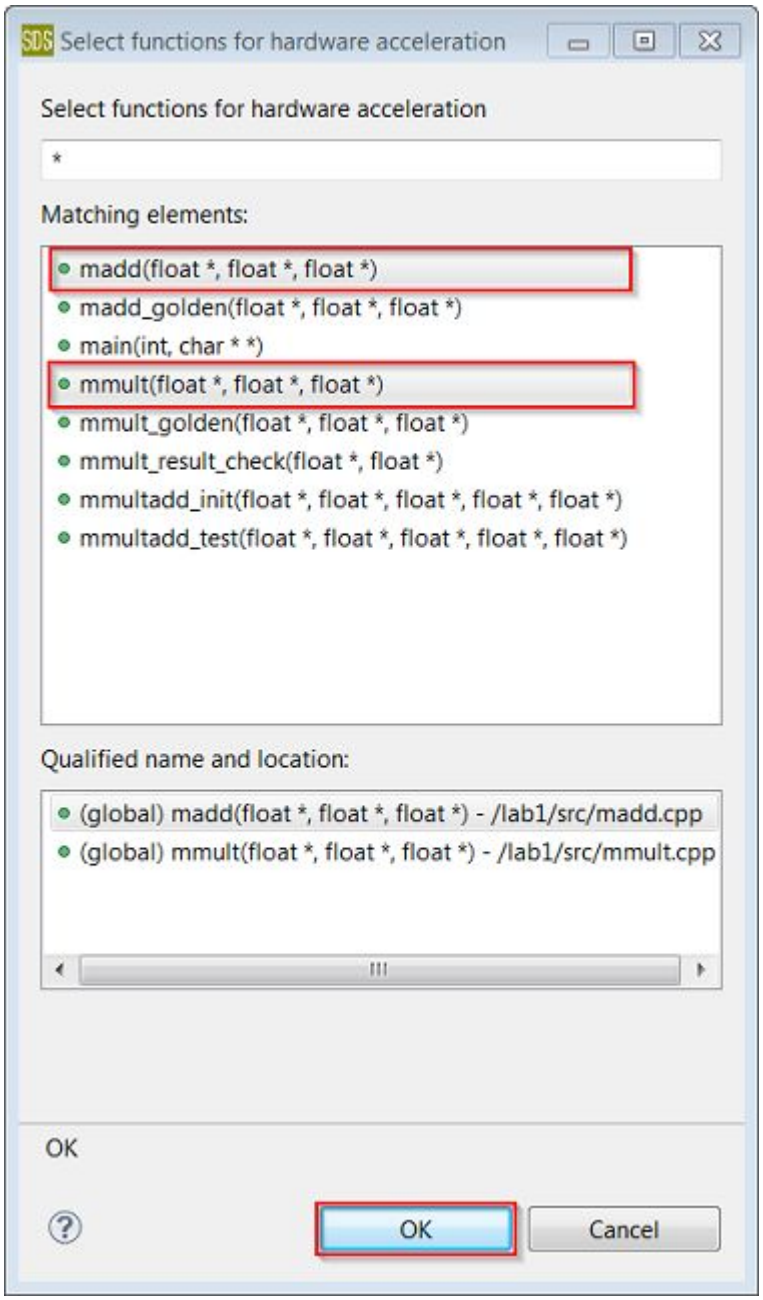


SDSoC アプリケーションをビルドする際は、ビルド コンフィギュレーション (ツール設定、フォルダー、ファイルなどのコレクション) を使用します。各ビルド コンフィギュレーションの目的は異なります。SDDebug ビルドでは、ELF (コンパイルおよびリンク済みプログラム) にデバッガーを実行するのに必要な 追加情報を含めてアプリケーションがビルドされます。ELF ファイルのデバッグ情報により、ファイル サイズが増加し、アプリケーション情報が表示されるようになります。SDRelease ビルドでは、同じ ELF ファイルが SDDebug オプションとして提供されますが、デバッグ情報が含まれない点が異なります。SDEstimate を使用すると、SDSoC 環境がアプリケーションのパフォーマンス予測を実行するモードで実行され、別の設定および手順が必要となります。詳細は、「チュートリアル : システム パフォーマンスの予測」を参照してください。

ハードウェア インプリメンテーション用の関数のマーク

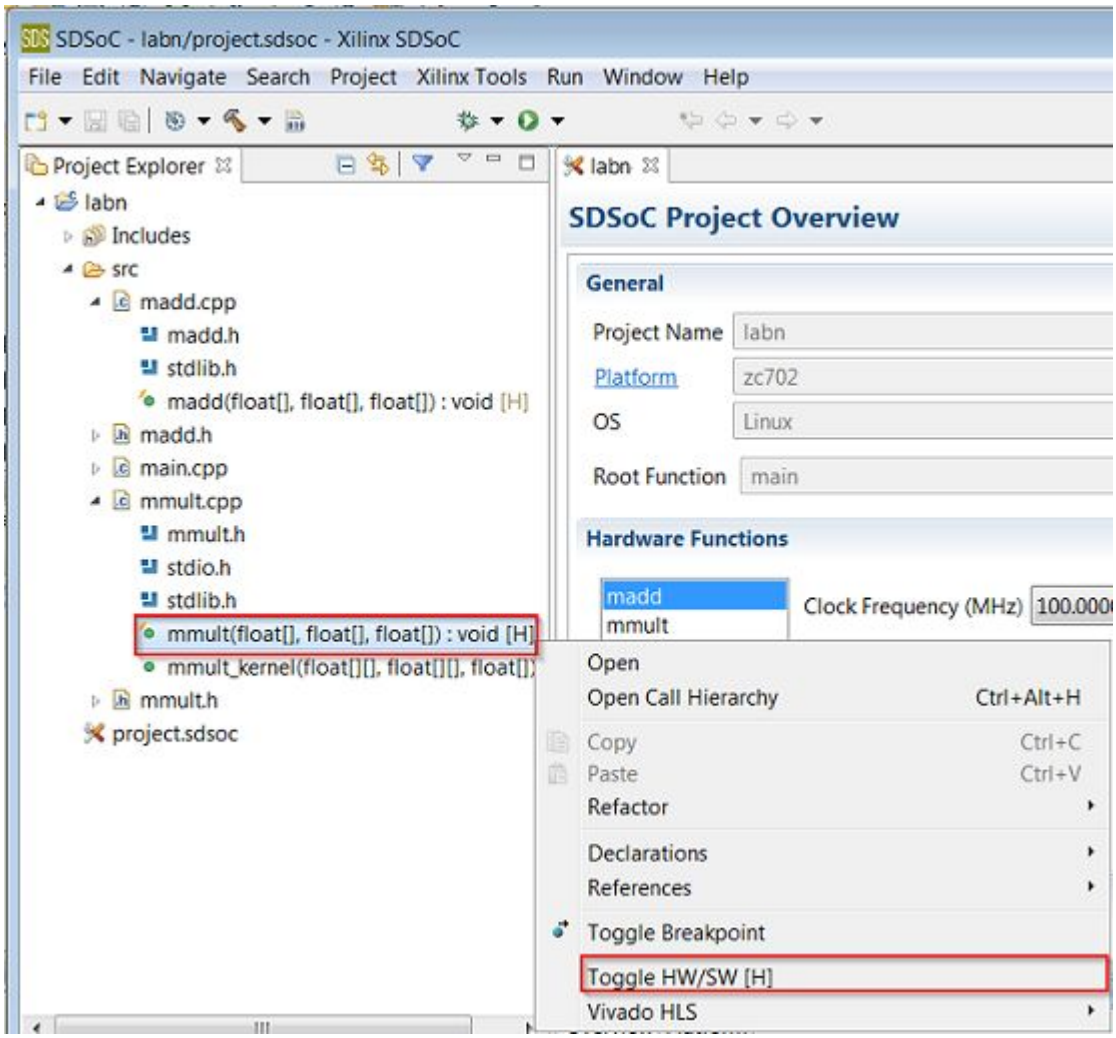
このアプリケーションには、2 つのハードウェア関数が含まれます。1 つは `mmult` で、2 つの行列を乗算して行列積を算出します。もう 1 つは `madd` で、2 つの行列を加算して行列和を算出します。これらのハードウェア関数がまとめられ、行列の乗加算関数を計算します。`mmult` と `madd` の両方のハードウェア関数をハードウェアでインプリメントされるように指定します。

1. [SDSoC Project Overview] では、プロジェクトの値を設定できます。[<name of project>] タブをクリックし (タブが開いていない場合は、[Project Explorer] タブで project.sdsoc ファイルをダブルクリック)、[Hardware Functions] セクションで [Add Hardware Function] アイコン をクリックしてハードウェア関数を指定します。



2. [Matching elements] リストで Ctrl キーを押しながら `mmult` と `madd` 関数をクリックして選択します。[OK] をクリックして、両方の関数を [Hardware Functions] セクションに追加します。

   または、[Project Explorer] タブで `mmult.cpp` および `madd.cpp` を展開表示し、`mmult` および `madd` 関数を右クリックして [Toggle HW/SW] をクリックします。関数が既にハードウェア用にマークされている場合は、[Toggle HW/SW [H]] と表示されます。ソース ファイルをエディターで開いている場合は、[Outline] タブでハードウェア関数を選択することもできます。



注意 : すべての関数がハードウェアにインプリメントできるわけではありません。詳細は、『SDSoC 環境ユーザー ガイド』(UG1027) の「コード ガイドライン」を参照してください。

システム ポートの指定

`sys_port` プラグマを使用すると、SDSoC システム コンパイラ ポートの代わりに、ACP または Zynq-7000 AP SoC Processing System (PS) の AFI ポートのいずれかを選択してプロセッサ メモリにアクセスできます。

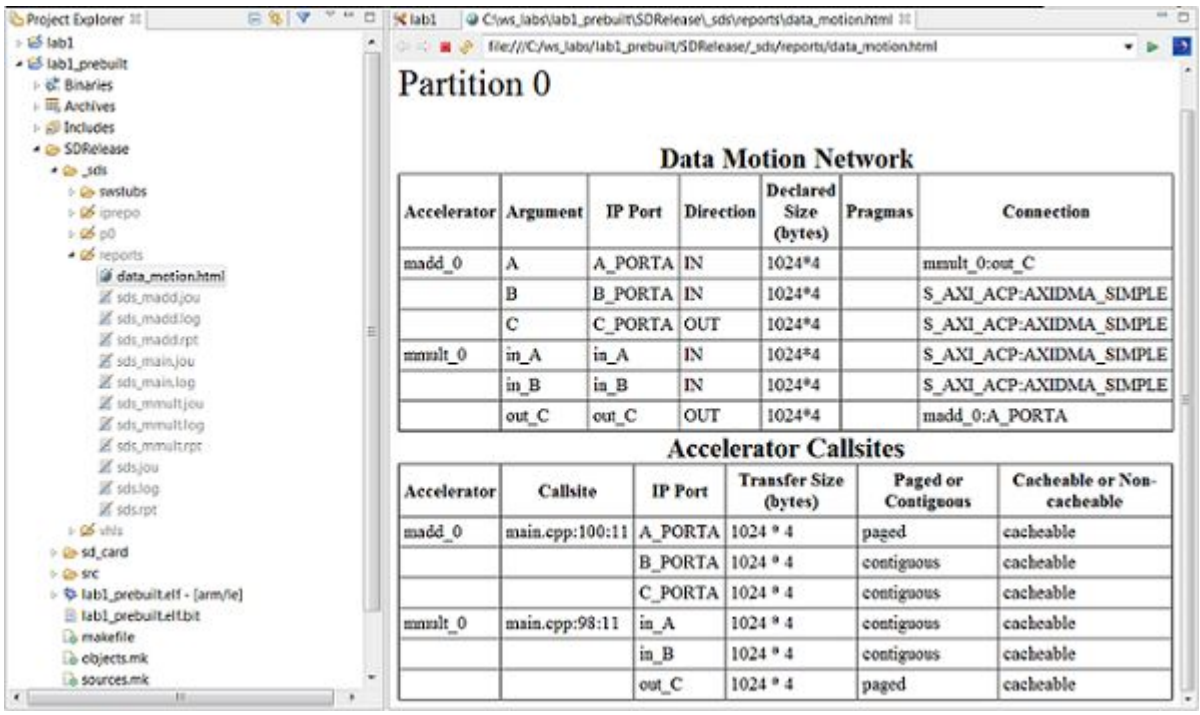
1. SDSoC システム コンパイラで生成されるシステムの構造を検証するために SD カードのブート イメージを生成する必要はないので、プロジェクト リンカー オプションをビットストリーム、ブート イメージ、ビルドが生成されないように設定します。
   a. [lab2] タブをクリックして [SDSoC Project Overview] を開きます。
   b. [Options] パネルで [Generate Bit Stream] および [Generate SD card Image] チェック ボックスをオフにします。
2. [Project Explorer] タブのプロジェクトの最上位フォルダーを右クリックして [Build Project] クリックします。

重要 : ビルド プロセスが完了するまでに、約 5 ～ 10 分かかります。

---

3. ビルドが完了したら、[SDSoC Project Overview] の [Reports] パネルで [Data motion] をクリックしてデータ モーション ネットワーク レポートを表示します。このレポートには、各ハードウェア関数のハードウェア/ソフトウェア接続性を記述する表が含まれます。

一番右の [Connection] 列には、行列乗算の入力配列に割り当てられた DMA のタイプ (`AXIDMA_SIMPLE`= simple DMA) と使用された Processing System 7 IP ポートが示されます。次の図は、`sys_port` プラグマを追加する前の `data_motion.html` ファイルの一部を表示しています。



4. `sys_port` プラグマを追加します。
   a. [Project Explorer] タブで `mmult.h` ファイルをダブルクリックしてソース エディターでファイルを開きます。
   b. `mmult` 関数の宣言直前に、次を挿入して、各入力配列それぞれに異なるシステム ポートを指定します。

```
#pragma SDS data sys_port(in_A:ACP, in_B:AFI)
```

   c. ファイルを保存します。

5. プロジェクトの最上位フォルダーを右クリックして [Clean Project] をクリックします。
6. プロジェクトの最上位フォルダーを右クリックして [Build Project] をクリックします。

重要 : ビルド プロセスが完了するまでに、約 5 ～ 10 分かかります。

7. ビルドが終了したら、`data_motion.html` ファイルを開きます。

   [Connection] 列に、行列乗算の各入力/出力配列に割り当てられたシステム ポートが表示されます。

エラーのレポート

次の手順を実行してエラーを発生させることができます。SDSoC 環境でエラーがどのように示されるかを確認してください。

1. ソース ファイル `main.cpp` を開き、ファイルの最後の方にある `std::cout` 文の最後のセミコロン (;) を削除します。

   行の左端に黄色のボックスが表示されます。
2. 黄色のボックスにカーソルを置くと、ツール ヒントにセミコロンが足りないことが示されます。
3. セミコロンを挿入すると、黄色のボックスは消えます。
4. `std::cout` を `std::cou` に変更し、行の左端にピンク色のボックスが表示されることを確認します。
5. ピンク色のボックスにカーソルを置くと、正しい表記 `std::cout` が表示されます。
6. `std::cou` を `std::cout` に変更してエラーを修正します。
7. `main()` で使用される変数を宣言する行をコメント アウトして、別のエラーを発生させます。
8. プロジェクトを保存してビルドします。ビルドが終了するまで待つ必要はありません。
9. コンソールをスクロールすると、エラー メッセージを確認できます。`SDRelease/_sds/reports/sds.log` および `SDRelease/_sds/reports/sds_mmult.log` ファイルを開いて、詳細なエラー レポートを確認します。

その他の演習

注記 : このセクションの手順は、オプションです。

Linux がアプリケーションのターゲット OS として使用される場合、アプリケーションのメモリ割り当ては Linux とサポートされるライブラリで処理されます。スコープ内のスタックで配列を宣言する場合 (int a[10000];) や、標準の malloc() 関数を使用してダイナミックに割り当てる場合は、プロセッサと Linux により提供される仮想アドレス空間の連続するメモリのセクションが取得されます。このバッファーは、通常物理アドレス空間の複数の不連続ページに分割され、Linux でソフトウェアがその配列にアクセスするたびに仮想/物理アドレス変換が自働的に処理されます。ただし、ハードウェア関数および DMA は物理アドレス空間にのみアクセスできるので、ソフトウェア ドライバーは各配列を仮想アドレスから物理アドレスに明示的に変換し、この物理アドレスを DMA またはハードウェア関数に供給する必要があります。各配列が物理アドレス空間の複数の不連続ページに分散していることがあるので、ドライバーは DMA に物理ページ アドレスのリストを供給する必要があります。1 つの配列用にページのリストを処理できる DMA はスキャッター ギャザー DMA と呼ばれ、1 つの物理アドレスのみを処理できる DMA はシンプル DMA と呼ばれます。シンプル DMA は、エリアとパフォーマンスの面ではスキャッター ギャザー DMA よりも安価ですが、sds_alloc() という特別のアロケーターを使用して各配列ごとに物理的に連続するメモリを取得する必要があります。

「チュートリアル：プロジェクトの作成、ビルド、実行」では、sds_alloc() を使用してシンプル DMA が使用されるようにしました。次の演習では、プラグマを使用してスキャッター ギャザー DAM や AXIFIFO などのほかのデータ ムーバーが使用されるようにします。ソース コードで sds_alloc() を malloc() にを変更し、スキャッター ギャザー DMA が自働的に選択されることを確認します。

データ ムーバー選択の制御

この演習では、「チュートリアル：プロジェクトの作成、ビルド、実行」(lab1) のソース コードにデータ ムーバー プラグマを追加して、ハードウェアとソフトウェア間での配列の転送に使用するデータ ムーバーを指定します。その後プロジェクトをビルドして、生成されたレポート (data_motion.html) を確認して、これらのプラグマの効果を確認します。ビルドでハードウェアが合成されないように、ビットストリームおよびブートファイルの生成はオフにしておきます。

データ ムーバー プラグマを追加して各配列に使用されるデータ ムーバーのタイプを指定するには、次の手順に従います。

1. [Project Explorer] タブで lab1/src の下の mmult.h をダブルクリックします。
2. mmult 関数宣言の上に次の行を挿入して各配列に別のデータ ムーバーを指定し、ファイルを保存します。

```
#pragma SDS data data_mover(in_A:AXIDMA_SG, in_B:AXIDMA_SIMPLE, out_C:AXIFIFO)
```

3. プロジェクトの最上位フォルダーを右クリックして、[Clean Project] をクリックします。
4. プロジェクトの最上位フォルダーを右クリックして、[Build Project] をクリックします。

重要： ビルド プロセスが完了するまでに、約 5 ～ 10 分かかります。

5. ビルドが完了したら、[Project Explorer] タブで SDRelease/_sds/reports/data_motion.html をダブルクリックして開きます。

   一番右の [Connection] 列に、行列乗算の各入力/出力配列に割り当てられたデータ ムーバーが示されます。

   注記： [Pragmas] 列には、使用されたプラグマがリストされます。AXIFIFO データ ムーバーは M_AXI_GP0 ポートに割り当てられ、その他 2 つのデータ ムーバーは S_AXI_ACP に関連付けられています。

sds_alloc() を malloc() に変更

この演習では、「チュートリアル : プロジェクトの作成、ビルド、実行」(lab1) のソース ファイルで sds_alloc() を malloc() に変更し、データ ムーバーがシンプル DMA からスキャッター ギャザー DMA に変更されることを確認します。[Generate Bit Stream] と [Generate SD card Image] をオフにしてビットストリームとブート ファイルが生成されないようにします。前のセクションから続けてこの演習を実行している場合は、lab1/src/mmult.h の mmult 宣言のデータ ムーバー プラグマを削除しておく必要があります。

1. [Project Explorer] タブで main.cpp をダブルクリックし、ソース エディター ビューで開きます。
2. バッファーが sds_alloc() で割り当てられている行をすべて検索して、sds_alloc() を malloc() に置き換えます。また、すべての sds_free() 呼び出しも free() に置換します。
3. ファイルを保存します。
4. プロジェクトの最上位フォルダーを右クリックして、[Clean Project] をクリックします。
5. プロジェクトの最上位フォルダーを右クリックして、[Build Project] をクリックします。

重要 : ビルド プロセスが完了するまでに、約 5 ～ 10 分かかります。

6. ビルドが完了したら、[Project Explorer] タブで SDRelease/_sds/reports/data_motion.html をダブルクリックして開きます。
7. 一番右の [Connection] 行に、行列乗算の入力配列に割り当てられた DMA のタイプ (AXIDMA_SG = スキャッター ギャザー DMA) と使用された Processing System 7 IP ポート (S_AXI_ACP) が示されます。[Accelerator Callsites] の表には、各転送に使用されるメモリ割り当てが連続なのか、ページなのかが示されます。

Vivado HLS ベースのアクセラレータ最適化の使用

この演習では、演習 1 のソースを変更して、生成したハードウェアのパフォーマンスへの Vivado HLS プラグマの影響について確認します。このトピックに関する詳細は、『SDSoC 環境ユーザー ガイド』(UG1027) の「プログラマ向け Vivado 高位合成ガイド」を参照してください。[Generate Bit Stream] と [Generate SD card Image] をオンにしてビットストリームとブート ファイルが生成されるようにします。前のセクションから続けて実行している場合は、malloc() を sds_alloc() に、free() を sds_free() に戻します。

1. [Project Explorer] タブで mmult.cpp をダブルクリックして、ソース エディター ビューを開きます。
2. HLS pipeline および HLS array_partition プラグマが記述されている行を見つけます。
3. これらのプラグマの行をコメントアウトします。
4. ファイルを保存します。
5. プロジェクトの最上位フォルダーを右クリックして、[Clean Project] をクリックします。
6. プロジェクトの最上位フォルダーを右クリックして、[Build Project] をクリックします。
7. ビルドが完了したら、sd_card フォルダーを SD カードにコピーして、ボードで実行します。

   プラグマがコメントアウトされていたときのパフォーマンスと比較すると、array_partition プラグマにより配列の要素を並列に読み出せるようになったので、内部ループのメモリ帯域幅が増加したことがわかります。pipeline プラグマでは、ループのパイプライン処理が実行され、ループの複数の反復が並列に実行されるようなったことがわかります。

転送されるデータ量を制御するプラグマの追加

この演習では、別のテンプレートを使用してコピー プラグマを使用する方法を示します。このテンプレートでは、dim1 と呼ばれる追加のパラメーターが行列乗算関数に渡されます。このパラメーターを使用すると、最大 32*32 までの任意サイズ dim1*dim1 の正方行列 2 つを乗算する行列乗算関数を使用できます。この行列の最上位の割り当てにより、最大 32x32 までのサイズの行列が作成されます。dim1 パラメーターは、行列乗算関数で乗算する行列のサイズを指定します。data copy プラグマは ™、最大の行列サイズではなく、実際の行列サイズに相当するデータ量を転送することを指定します。

1. SDSoC 環境を起動し、可変データ サイズを使用する行列乗算デザイン テンプレートを使用して、ZC702 と Linux プラットフォーム用の新しいプロジェクトを作成します。
   a. [File] → [New] → [SDSoC Project] をクリックします。
   b. New Project ウィザードにプロジェクトの名前 (たとえば lab2a) を入力します。
   c. [zc702] と [Linux] を選択します。
   d. [Next] をクリックします。
   e. [Available Templates] から [Matrix Multiplication Data Size] を選択し、[Finish] をクリックします。
   f. src/mmult_accel/mmult_accel 関数をハードウェア アクセラレーション用にマークします。
2. ビットストリームとブート ファイルがビルドされないよう設定します。
3. [Project Explorer] タブで mmult_accel.h をダブルクリックしてソース エディター ビューで開き、data copy プラグマを追加します。

   次のように、各配列に異なるデータ コピー サイズを指定します。プラグマでは、関数のスカラー引数のどれでも使用してデータ コピー サイズを指定できます。ここでは、サイズを指定するのに dim1 を使用しています。

```
#pragma SDS data copy(in_A[0:dim1*dim1])
#pragma SDS data copy(in_B[0:dim1*dim1])
#pragma SDS data copy(out_C[0:dim1*dim1])
void mmult_accel (float in_A[A_NROWS*A_NCOLS],
                  float in_B[A_NCOLS*B_NCOLS],
                  float out_C[A_NROWS*B_NCOLS],
                  int dim1);
```

4. プロジェクトの最上位フォルダーを右クリックして、[Clean Project] をクリックします。
5. プロジェクトの最上位フォルダーを右クリックして、[Build Project] をクリックします。

重要 : ビルド プロセスが完了するまでに、約 5 ～ 10 分かかります。

6. ビルドが完了したら、[Project Explorer] タブで SDDebug /_sds/reports/data_motion.html をダブルクリックして開きます。

7. 右から 2 番目の [Pragmas] 列に、各配列のデータ転送の長さが表示されます。2 つ目の表には、各ハードウェア関数呼び出しサイトの転送サイズが示されます。

Partition 0

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
mmult_accel_0	in_A	in_A	IN	1024*4	• length:(dim1*dim1)	S_AXI_ACP:AXIDMA_SIMPLE
	in_B	in_B	IN	1024*4	• length:(dim1*dim1)	S_AXI_ACP:AXIDMA_SIMPLE
	out_C	out_C	OUT	1024*4	• length:(dim1*dim1)	S_AXI_ACP:AXIDMA_SIMPLE
	dim1	dim1	IN	4		M_AXI_GP0:AXILITE:0x80

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Cacheable or Non-cacheable
mmult_accel_0	mmult.cpp:78:5	in_A	(dim1*dim1) * 4	contiguous	cacheable
		in_B	(dim1*dim1) * 4	contiguous	cacheable
		out_C	(dim1*dim1) * 4	contiguous	cacheable
		dim1	4	paged	cacheable

第 4 章

チュートリアル：システムのデバッグ

このチュートリアルでは、SDSoC™ 環境のインタラクティブ デバッガーを使用する方法を示します。

まず、デザインのターゲットをスタンドアロン オペレーティング システムまたはプラットフォームに設定し、ザイリンクス SDSoC 環境を使用してスタンドアロン アプリケーションを実行して、アプリケーションをデバッグします。

その後 Linux アプリケーションを作成し、インタラクティブ デバッガーを使用してコードをステップ スルーします。

このチュートリアルでは、アクセラレーションされたシステムで実行しているアプリケーションをデバッグします。

注記： チュートリアルは各手順に分けられ、それぞれで大まかな手順が説明された後、細かい手順が説明されていますので、スキル レベルに合った方の手順を参照してください。大まかな手順を終了するのにヘルプが必要な場合は詳細な手順を参照したり、細かい手順を飛ばして次の大まかな手順に進んだりできます。

チュートリアルの目標

このチュートリアルを終了すると、次ができるようになります。

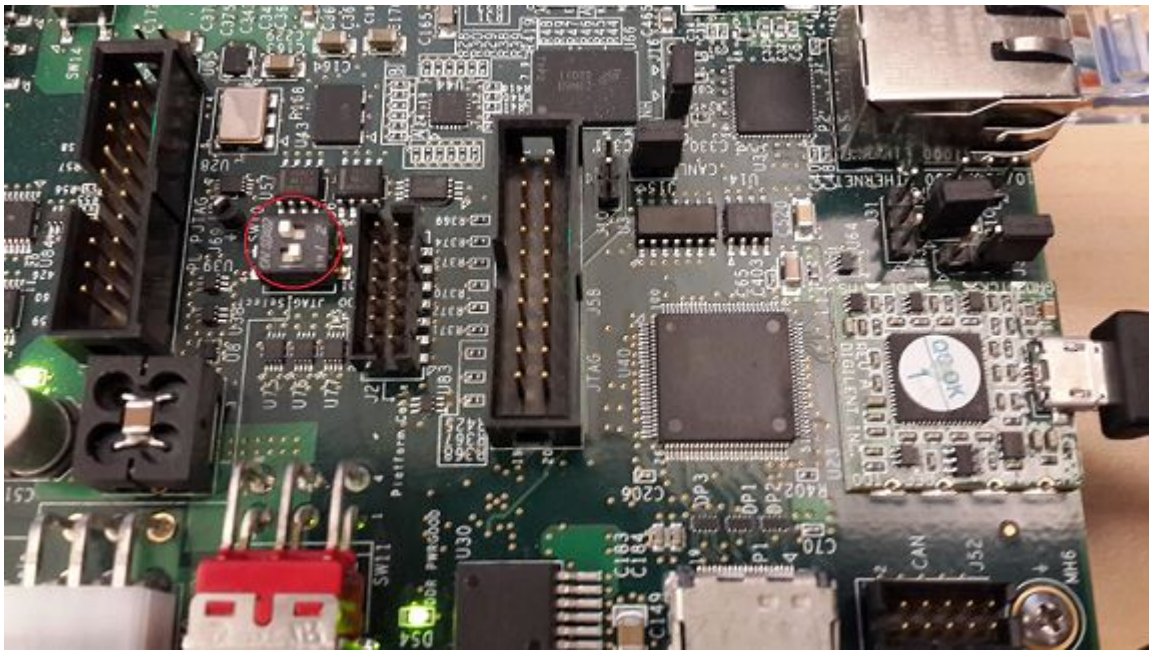
- SDSoC 環境を使用して、スタンドアロン アプリケーションをダウンロードして実行
- オプションで SDSoC 環境でソース コードをステップ スルーして、さまざまなレジスタおよびメモリを確認 (ARM A9 で実行するコードに制限され、ハードウェア関数に変換されたコードには適用されない)

ボードの設定

ボードの UART ポートに接続するには mini USB ケーブルが必要です。これにより SDSoC 環境のシリアル ターミナルに通信できるようになります。ボードの Digilent ポートに接続するには Micro USB ケーブルも必要で、これによりビットストリームおよびバイナリをダウンロードできます。最後に、SD カードから起動できるように、SD カード スロットのサイドのジャンパーが正しく設定されているかどうか確認します。

1. mini USB ケーブルを UART ポートに接続します。「シリアル ターミナルへのボードの接続」を参照してください。

2. JTAG モードが Digilent ケーブルを使用するように設定されており、Micro USB ケーブルが接続されていることを確認します。



3. ジャンパーを SD ブート モードに設定します。SD カードは挿入しないでください。「SD カード ブートのボード コンフィギュレーション」を参照してください。
4. ボードに電源を投入します。

Windows で USB-UART ドライバーと Digilent ドライバーがインストールされるようにし、SDSoC 環境がボードと通信できるようにします。

重要 : ボードのジャンパーが SD ブートまたは JTAG ブートに設定されていることを確認します。このようにしておかないと、ボードが QSPI ブートなどのその他のモードでパワーアップし、QSPI デバイスまたはその他のブート デバイスからこの演習に関係のないものが読み込まれてしまいます。

スタンドアロン プロジェクトの作成

[Matrix Multiplication and Addition] デザイン テンプレートを使用して ZC702 プラットフォームおよびスタンドアロン OS の新しい SDSoC™ 環境プロジェクト (lab3) を作成します。

SDSoC 環境でスタンドアロン プロジェクトを作成する手順は、次のとおりです。

1. SDSoC 環境を起動します。
2. [File] → [New] → [SDSoC Project] をクリックします。
3. [Project name] テキスト ボックスにプロジェクト名 (たとえば lab3) を指定します。
4. [Platform] ドロップダウンから [zc702] を選択します。
5. [OS] ドロップダウンから、[Standalone] を選択します。

6. [Next] をクリックします。

   [Templates] ページが表示されます。

7. [Available Templates] のリストから [Matrix Multiplication and Addition] を選択し、[Finish] をクリックします。
8. [lab3] タブをクリックし (タブが開いていない場合は、[Project Explorer] タブで `project.sdsoc` ファイルをダブルクリック)、[Hardware Functions] パネルで [Add Hardware Function] アイコン をクリックしてハードウェア関数を指定するダイアログ ボックスを表示します。
9. [Matching elements] リストで Ctrl キーを押しながら `mmult` と `madd` 関数をクリックして選択します。[OK] をクリックして、両方の関数を [Hardware Functions] セクションに追加します。
10. [Project Explorer] タブでプロジェクトを右クリックし、[Build Project] をクリックします。

    SDSoC によりプロジェクトがビルドされます。ビルド プロセスのステータスを示すダイアログ ボックスが表示されます。

重要 : ビルド プロセスが完了するまでに、約 30 ～ 45 分かかります。プロジェクトをビルドする代わりに、ビルド済みプロジェクトを使用して時間を節約することもできます。ビルド済みプロジェクトをインポートするには、[File] → [Import] をクリックし、[General] → [Existing Projects into Workspace] を選択して [Next] をクリックします。[Select archive file] としてプロジェクト ファイル フォルダーの `lab3_standalone_prebuilt.zip` ファイル (`<path to install>/SDSoC/2015.4/docs/labs/lab3_standalone_prebuilt.zip`) を指定します。[開く] をクリックします。[Finish] をクリックします。

注記 : プロジェクトをインポートした場合は、バイナリ ELF ファイルにソース デバッグ用の正しいパスが含まれません。ELF を再ビルドする必要がありますが、プログラマブル ロジック ビットストリームは再ビルドしたくありません。この場合、[Project Explorer] タブで lab3_standalone_prebuilt プロジェクトを展開表示し、`project.sdsoc` をダブルクリックして [SDSoC Project Overview] を表示します。[Options] パネルで [Generate Bit Stream] をオフにして、[Generate SD card Image] はオンにままにします。lab3_standalone_prebuilt プロジェクトを右クリックして [Clean Project] をクリックしてプロジェクトをクリーンアップし、lab3_standalone_prebuilt を右クリックして [Build Project] をクリックして再ビルドします。

デバッグ コンフィギュレーションの設定

デバッグ コンフィギュレーションを設定するには、次の手順に従います。

1. [Project Explorer] タブの lab3_standalone_prebuilt プロジェクトの SDDebug フォルダーにある ELF (`.elf`) ファイルをクリックし、[SDSoC Project Overview] で [Debug application] をクリックします。または、プロジェクトを右クリックし、[Debug As] → [Launch on Hardware (SDSoC Debugger)] をクリックします。[Confirm Perspective Switch] ダイアログ ボックスが表示されます。

重要 : プロジェクトをデバッグする前にボードのスイッチがオンになっていることを確認してください。

---

2. ダイアログ ボックスで [Yes] をクリックします。

   これで、SDSoC 環境が [Debug] パースペクティブになりました。デバッガーによりシステムがリセットされ、デバイスがプログラムおよび初期化され、`main` 関数でブレークされます。中央のパネルにソース コード、右上のパネルにローカル変数、右下のパネルに SDK ログが表示されます。

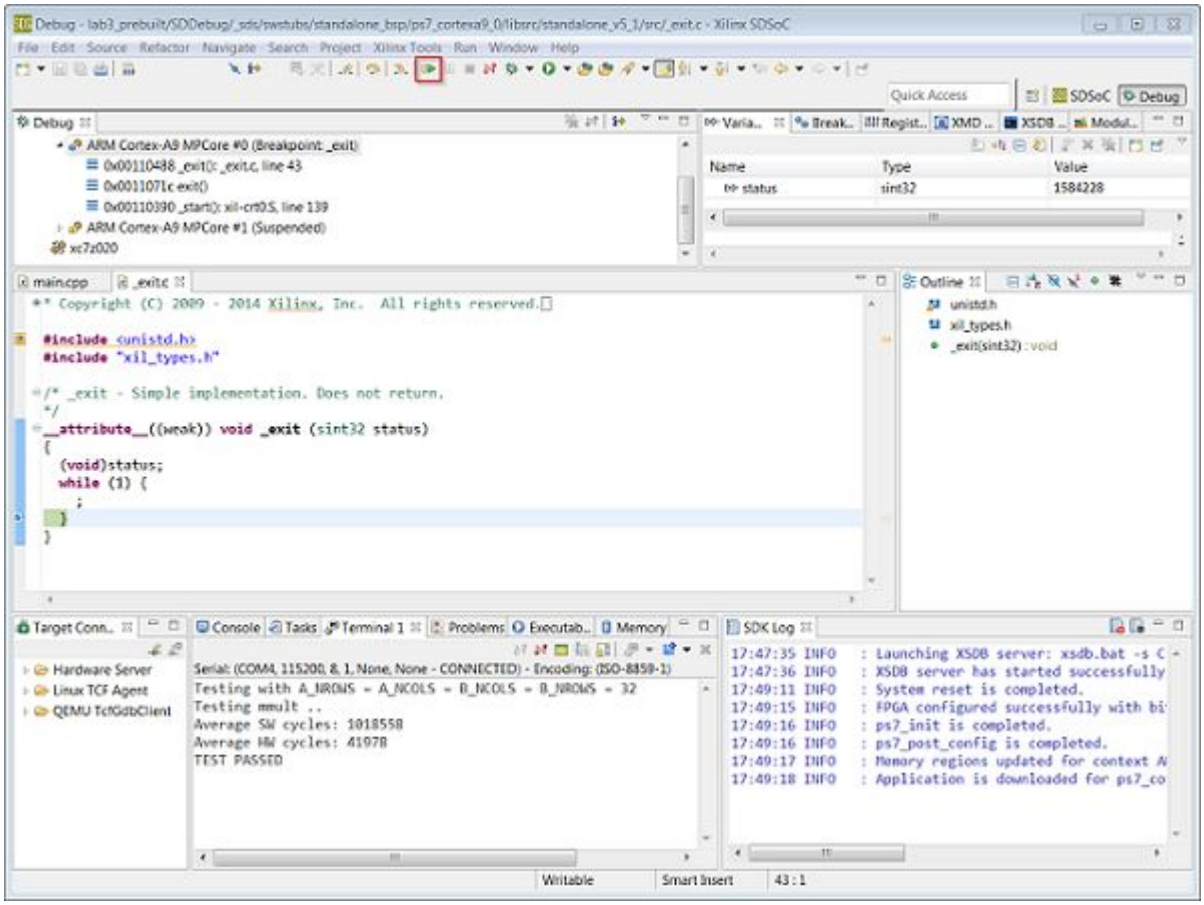
3. アプリケーションを実行する前に、シリアル ターミナルをボードに接続して、プログラムからの出力が表示されるようにする必要があります。この例では、[Window] → [Show View] → [Other] をクリックし、[Terminal] → [Terminal] をクリックして起動した SDSoC 環境ターミナルを使用します。[Debug] ウィンドウの下部にある [Terminal] タブをクリックします。[Connection Type] : Serial、[Port] : COM<n>、[Baud Rate] :115200 baud に設定されているので、[Connect] アイコン をクリックしてターミナルを電源が投入済みのボードに接続します。

アプリケーションの実行

アプリケーションを実行する手順は、次のとおりです。

[Resume] アイコン をクリックしてアプリケーションを実行し、出力をターミナル ウィンドウで確認します。

注記 : ソース コード ウィンドウに `_exit` 関数が表示され、[Terminal] タブに行列乗算アプリケーションからの出力が表示されます。



その他の演習

注記 : このセクションの手順は、オプションです。

アプリケーションを使用したデバッグ/ステップの方法、SD ブート モードでの実行方法、Linux アプリケーションのデバッグ方法について説明します。

コードのステップ スルー

[Debug] パースペクティブには、この演習では説明しなかったその他多くの機能が含まれます。最も重要なのは、デバッグするコードをステップ スルーする機能です。

1. [Debug] パースペクティブのデバッグ階層の最上位のフォルダーを右クリックして、[Disconnect] をクリックします。
2. 最上位のデバッグ フォルダーをもう 1 度右クリックして、[Remove all Terminated] をクリックします。
3. バグ アイコンをクリックしてデバッガーを起動し、[step-into]、[step-over]、[step-return] ボタンを使用してコードをステップ スルーします。
4. コードをステップ スルーしながら、さまざまな変数の値を確認します。

重要 :

[terminate] および [relaunch] ボタンを使用すると、™起動できなかったことを示すエラー メッセージが表示されることがあります。この場合、SDSoC 環境を再起動してボードの電源をいったん切って入れ直します。

「デバッグ コンフィギュレーションの設定」を実行せず、ビットストリーム生成をオフにしなかった場合は、デバッガーを起動しようとしたときに SDSoC 環境がアプリケーションをクリーンアップしてビルドし直そうとします。この例の場合、これに約 30 分かかります。

SD からのブート

1. Windows のファイルのコピー/貼り付けを使用して、`Debug` フォルダー内の `sd_card` フォルダーから `BOOT.BIN` ファイルを SD カードにコピーします。
2. ZC702 ボードに SD カードを挿入して、ジャンパーを SD ブート モードに設定して、ボードに電源を投入します。
3. SDSoC IDE で [Terminal] タブがまだ接続されていることを確認し、そのターミナルに表示されているアリケーションの出力を表示します。

Linux アプリケーションのデバッグ

SDSoC 環境で Linux アプリケーションをデバッグするには、次の手順に従います。

1. ZC702 と Linux をターゲットにしたプロジェクトを作成または選択します。

   詳細は、「新規プロジェクトの作成」を参照してください。
2. ハードウェア インプリメンテーション用の関数をマークします。

   詳細は、「ハードウェア インプリメンテーション用の関数のマーク」を参照してください。

3. プロジェクトをビルドして実行ファイル、ビットストリーム、SD カード ブート イメージを生成します。

   詳細は、「ハードウェア アクセラレータを使用したデザインのビルド」を参照してください。

重要： 実行ファイルのビルドには、マシンによって 30 ～ 60 分かかります。プロジェクトをビルドする代わりに、ビルド済みプロジェクトを使用して時間を節約することもできます。ビルド済みプロジェクトをインポートするには、[File] → [Import] をクリックし、[General] → [Existing Projects into Workspace] を選択して [Next] をクリックします。[Select archive file] としてプロジェクト ファイル フォルダーの `lab3_linux_prebuilt.zip` ファイル (`<path to install>/SDSoC/2015.4/docs/labs/lab3_linux_prebuilt.zip`) を指定します。[開く] をクリックします。[Finish] をクリックします。

注記： プロジェクトをインポートした場合は、バイナリ ELF ファイルにソース デバッグ用の正しいパスが含まれません。ELF を再ビルドする必要がありますが、プログラマブル ロジック ビットストリームは再ビルドしたくありません。この場合、[Project Explorer] タブで lab3_linux_prebuilt プロジェクトを展開表示し、`project.sdsoc` をダブルクリックして [SDSoC Project Overview] を表示します。[Options] パネルで [Generate Bit Stream] をオフにして、[Generate SD card Image] はオンにままにします。lab3_linux_prebuilt プロジェクトを右クリックして [Clean Project] をクリックしてプロジェクトをクリーンアップし、lab3_linux_prebuilt を右クリックして [Build Project] をクリックして再ビルドします。

4. ここでは、[Window] → [Show View] → [Other] をクリックし、[Terminal] → [Terminal] をクリックして起動した SDSoC 環境ターミナルを使用します。[Debug] ウィンドウの下部にある [Terminal] タブをクリックし、設定 ([Connection Type]： Serial、[Port] :COM<n>、[Baud Rate]： 115200 ボー) を確認します。

   COM ポート設定が表示されるようにするには、ボードに電源を投入する必要があります。

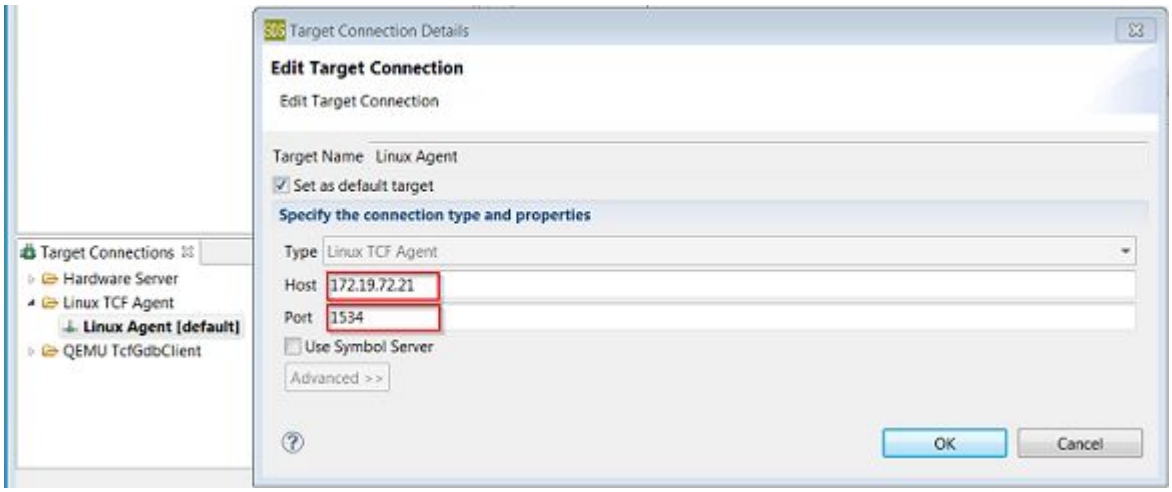
   - SD カードを挿入せずにボードに電源を投入します。
   - [Terminal] タブの [Settings] アイコン をクリックし、コンフィギュレーションを設定し、[OK] をクリックします。
   - ターミナルに接続されていることが示されます。赤い [Disconnect] アイコン をクリックしてボードからターミナルの接続を解除して、ボードの電源を切ります。

5. 生成した `sd_card` ディレクトリの内容を SD カードにコピーして、SD カードを ZC702 ボードに挿入します。
6. ボードがイーサネット ケーブルを使用してイーサネット ルーターに接続されていることを確認します。ボードに電源を投入します。[Terminal] タブをクリックして、緑の [Connect] アイコンをクリックして、ターミナルをボードに接続します。Linux のブート ログがターミナルに表示されます。「Sending select for 172.19.73.248…Lease of 172.19.73.248 obtained」というボードに割り当てられた IP アドレスがレポートされている行を見つけます。このアドレスを次の手順で使用するために記録します。

   IP アドレスが表示されていない場合は、ターミナルで `ifconfig` と入力します。

7. SDSoC 環境に戻り、[Target Connections] タブで [Linux TCF Agent] を展開表示して [Linux Agent (default)] を右クリックし、[Edit] をクリックします。

8. [Target Connection Details] ダイアログ ボックスで IP アドレスとポート (1534) を設定します。



9. [OK] をクリックします。
10. [SDSoC Project Overview] の [Actions] パネルで [Connection] が [Linux Agent] に設定されていることを確認し、[Debug Application] をクリックして [Debug] パースペクティブに移動して、コードを実行するかステップ スルーします。

注記 : アプリケーションの出力は [Terminal] タブではなく [Console] タブに表示されます。

第 5 章

チュートリアル : システム パフォーマンスの予測

このチュートリアルでは、ビルド サイクル全体を実行せずに、アプリケーションのパフォーマンスを予測する方法について説明します。

注記 : チュートリアルは各手順に分けられ、それぞれで大まかな手順が説明された後、細かい手順が説明されていますので、スキル レベルに合った方の手順を参照してください。大まかな手順を終了するのにヘルプが必要な場合は詳細な手順を参照したり、細かい手順を飛ばして次の大まかな手順に進んだりできます。

チュートリアルの目標

チュートリアルを終了すると、SDSoC 開発を使用して、選択した関数に基づいてスピードアップを予測できるようになります。

「ボードの設定」

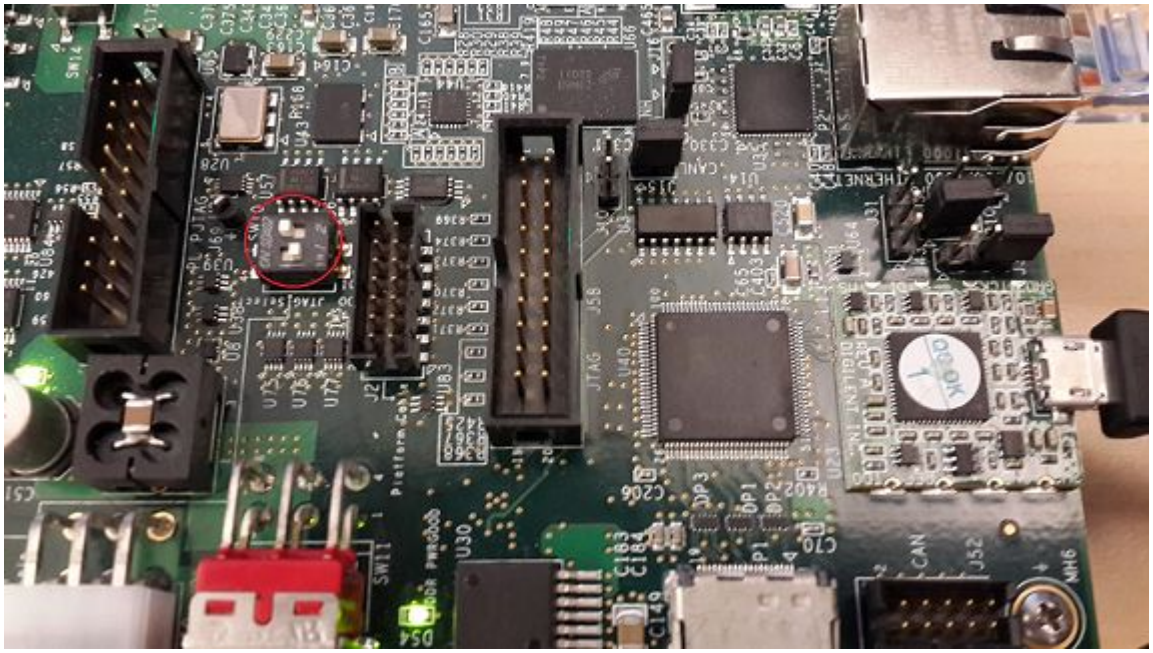
「その他の演習」

ボードの設定

ボードの UART ポートに接続するには mini USB ケーブルが必要です。これにより SDSoC 環境のシリアル ターミナルに通信できるようになります。ボードの Digilent ポートに接続するには Micro USB ケーブルも必要で、これによりビットストリームおよびバイナリをダウンロードできます。最後に、SD カードから起動できるように、SD カードスロットのサイドのジャンパーが正しく設定されているかどうか確認します。

1. mini USB ケーブルを UART ポートに接続します。「シリアル ターミナルへのボードの接続」を参照してください。

2. JTAG モードが Digilent ケーブルを使用するように設定されており、Micro USB ケーブルが接続されていることを確認します。



3. ジャンパーを SD ブート モードに設定します。SD カードは挿入しないでください。「SD カード ブートのボード コンフィギュレーション」を参照してください。
4. ボードに電源を投入します。

Windows で USB-UART ドライバーと Digilent ドライバーがインストールされるようにし、SDSoC 環境がボードと通信できるようにします。

重要 : ボードのジャンパーが SD ブートまたは JTAG ブートに設定されていることを確認します。このようにしておかないと、ボードが QSPI ブートなどのその他のモードでパワーアップし、QSPI デバイスまたはその他のブート デバイスからこの演習に関係のないものが読み込まれてしまいます。

SDEstimate コンフィギュレーションを使用するプロジェクトの設定

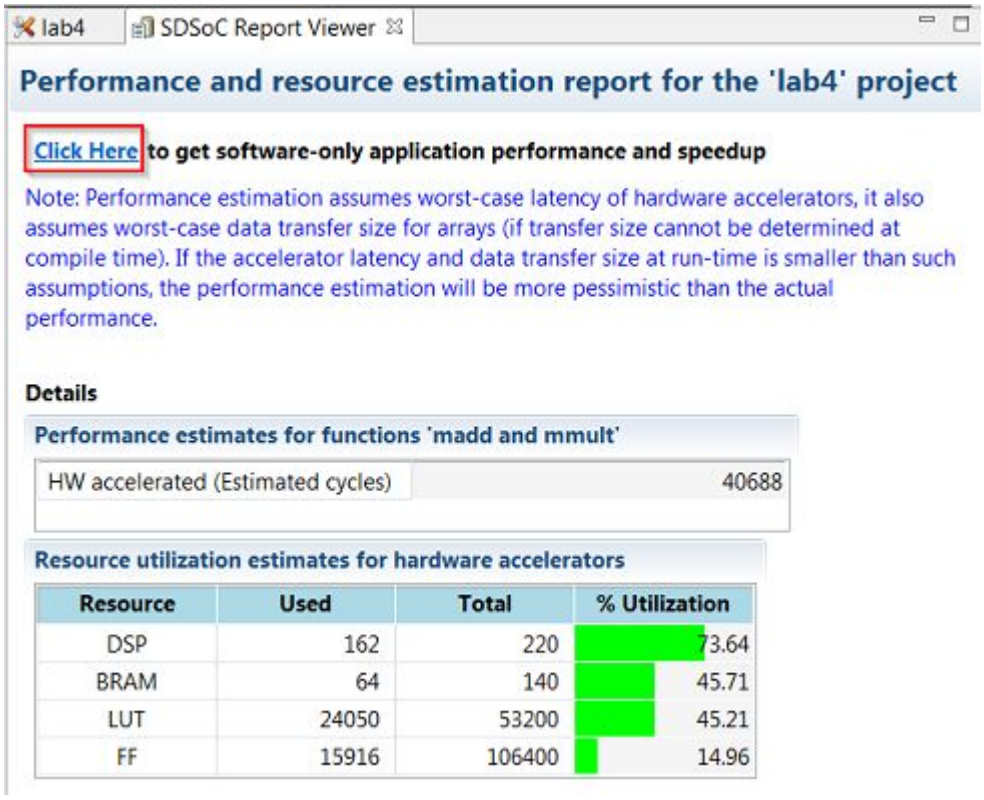
プロジェクトを作成して SDEstimate コンフィギュレーションを使用する手順は、次のとおりです。

1. [Matrix Multiplication and Addition] デザイン テンプレートを使用して ZC702 プラットフォームおよびスタンドアロンの新しい SDSoC™ 環境プロジェクト (lab4) を作成します。
2. [lab4] タブをクリックして [SDSoC Project Overview] を開きます。タブが表示されていない場合は、[Project Explorer] タブの [lab4] プロジェクトの下の `project.sdsoc` ファイルをダブルクリックします。
3. [Hardware Functions] パネルで [Add Hardware Function] アイコン ( ) をクリックして、ハードウェア関数を指定するダイアログ ボックスを起動します。

4. [Matching elements] で Ctrl キーを押しながら `madd` および `mmult` 関数をクリックし、[Qualified name and location] リストの表示されるようにします。[OK] をクリックします。
5. [SDSoC Project Overview] の [Actions] パネルで [Estimate Performance Speedup for HW Functions] をクリックします。これにより、SDEstimate ビルド コンフィギュレーションが選択され、予測フローが実行されます。
6. [Build Project] ダイアログ ボックスに、プロジェクトをビルドするかどうかを尋ねるメッセージが表示されます。[OK] をクリックします。

SDSoC 環境でプロジェクトがビルドされます。ビルド プロセスのステータスを示すダイアログ ボックスが表示されます。

ビルドが完了したら、初期レポートを表示できます。このレポートには、ハードウェアのみの予測サマリが含まれ、リンクをクリックすると、ソフトウェアの実行データを取得できます。これにより、ハードウェア インプリメンテーションとソフトウェアのみの情報が比較されてレポートがアップデートされます。



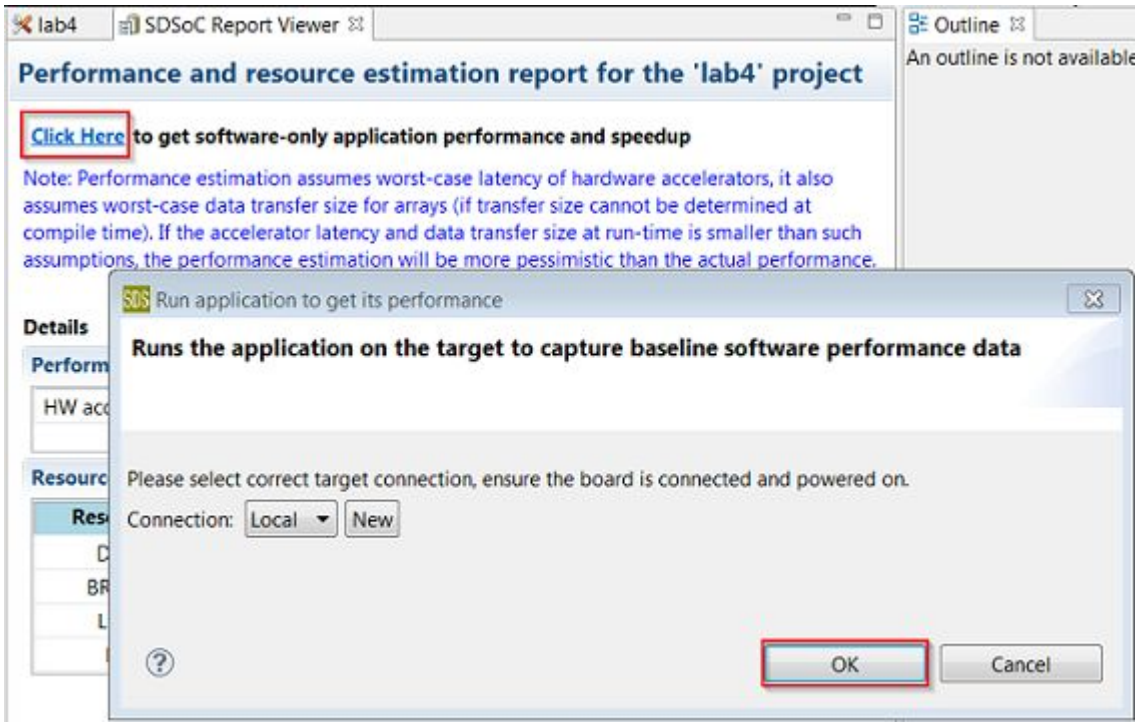
ソフトウェアとハードウェアのパフォーマンス比較

重要 : このセクションの手順を実行する前に、ボードのスイッチがオンになっていることを確認してください。

ソフトウェア実行データを収集してパフォーマンス予測レポートを生成するには、次の手順に従います。

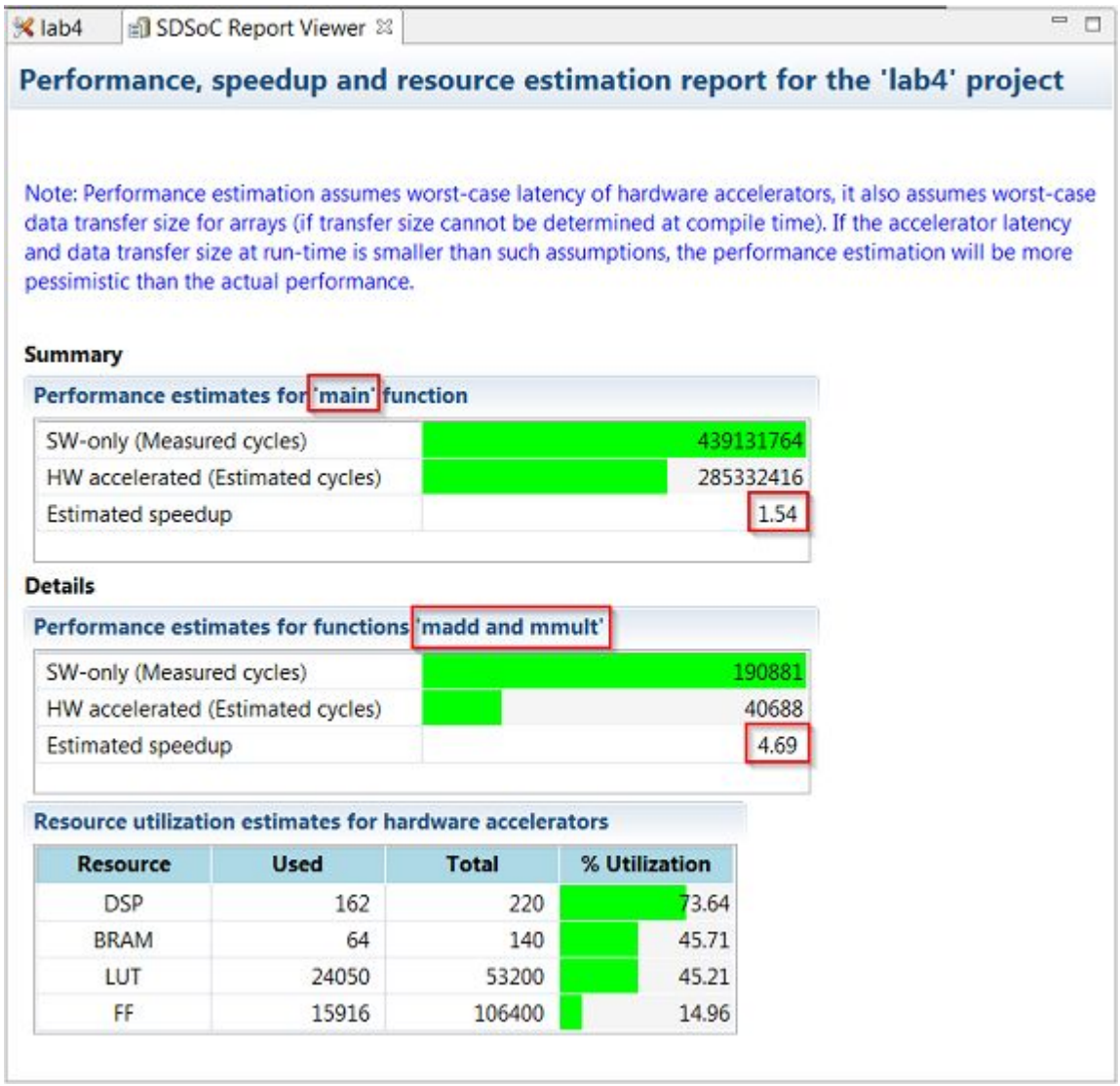
1. [SDSoC Report Viewer] タブを開きます。

2. [Click Here] リンクをクリックして、ボードのアプリケーションを起動します。
   [Run application to get its performance] ダイアログ ボックスが表示されます。
3. 既存の接続を選択するか、新しい接続を作成してターゲット ボードに接続します。



4. [OK] をクリックします。

   デバッガーによりシステムが利せてとされ、FPGA がプログラムされて初期化され、アプリケーションのソフトウェアのみのバージョンが実行されます。この後、パフォーマンス データが収集されて、パフォーマンス予測レポートを表示するのに使用されます。

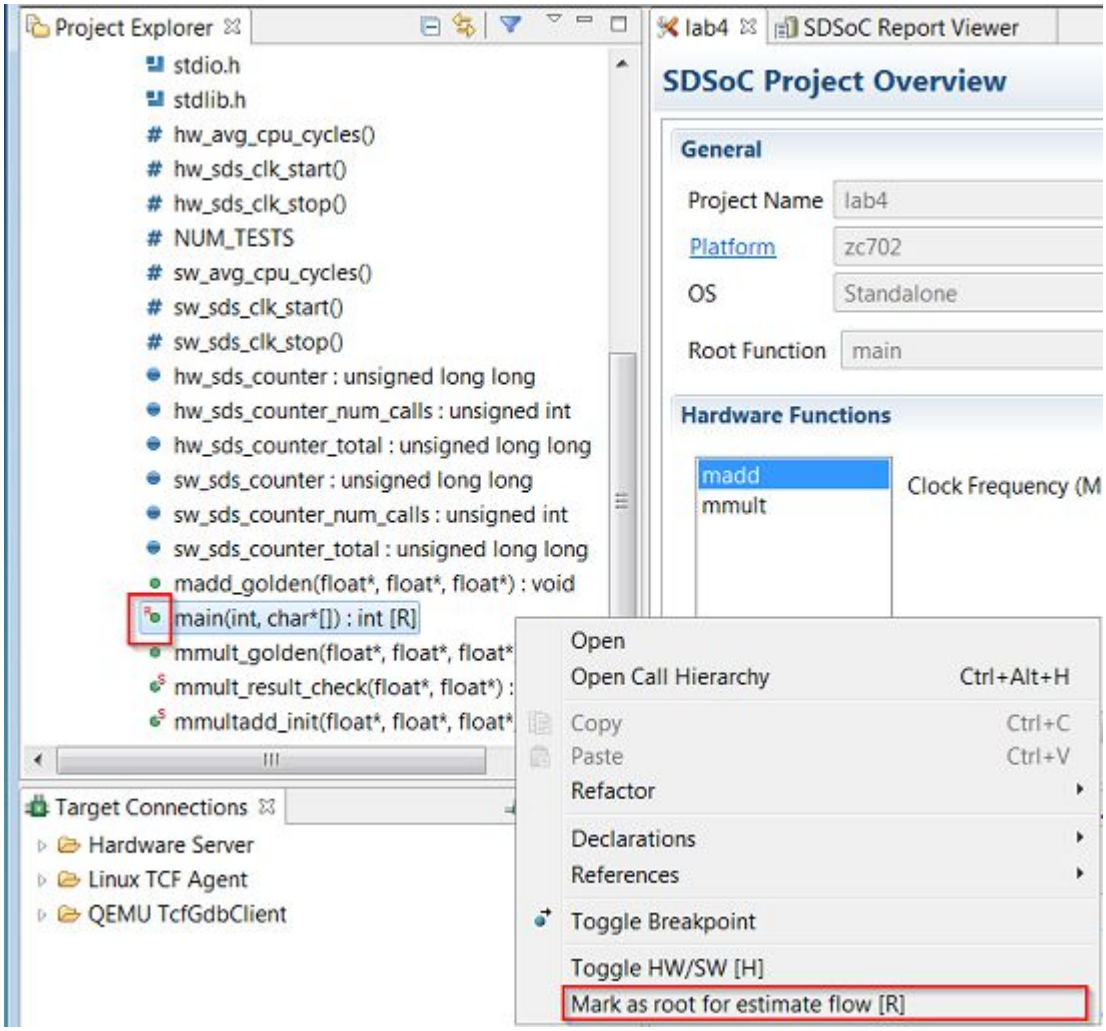


全体的なスピードアップ比較のスコープ変更

パフォーマンス予測レポートの最初の行には、最上位関数 (perf root) の予測スピードアップが示されます。この関数は、デフォルトで main に設定されますが、たとえばバッファーの割り当て、初期化、設定など、この比較から除外したいコードがあることもあります。その他の関数を考慮する際に全体的なスピードアップを確認する場合は、パフォーマンス予測フローのルートとして別の関数を指定します。このフローは、ハードウェア アクセラレーションに選択したすべての関数がルートの子である場合に使用できます。

1. パフォーマンス スピードアップの予測に使用する最上位関数 (perf root) を変更するには、SDSoC IDE の右上の [SDSoC] ボタンをクリックして SDSoC パースペクティブに戻し、[Project Explorer] タブでルートとして選択する関数を右クリックして、[Mark as root for estimate flow] をクリックします。

   次の図に示すように、その関数のアイコンの左上に小さな R が表示されます。選択した関数は、ハードウェア アクセラレーションに選択された関数の親です。



2. [Project Explorer] タブでプロジェクトを右クリックして [Clean Project] をクリックし、[Build Project] をクリックします。[SDSoC Project Overview] で [EstimatePerformance Speedup for HW Functions] をクリックし、予測レポートを生成し直して、選択した関数の基づいて全体的なスピードアップ予測を取得します。

その他の演習

注記 : このセクションの手順は、オプションです。

アプリケーションのターゲット OS として Linux を使用する場合に、パフォーマンス予測フローを使用する方法について説明します。

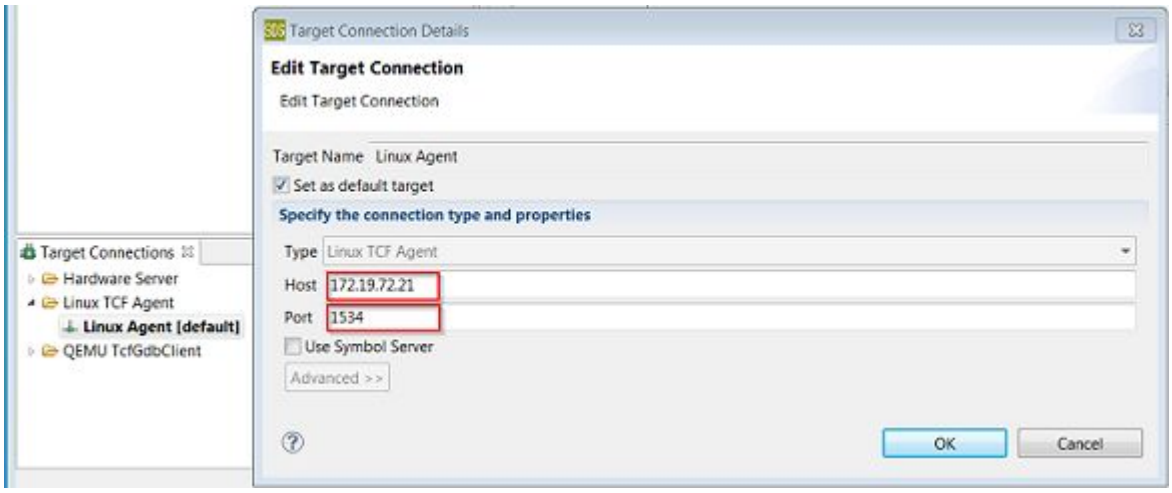
Linux でのパフォーマンス予測フローの使用

Linux でパフォーマンス予測フローを使用する手順は、次のとおりです。

1. [Matrix Multiplication and Addition] デザイン テンプレートを使用して ZC702 プラットフォームおよび Linux OS の新しい SDSoC™ 環境プロジェクト (lab4_linux) を作成します。
2. [lab4_linux] タブをクリックします。タブが表示されていない場合は [Project Explorer] タブで [lab4_linux] プロジェクトの `project.sdsoc` をダブルクリックします。[Hardware Functions] パネルで [Add Hardware Function] アイコン ( ) をクリックして、ハードウェア関数を指定するダイアログ ボックスを起動します。
3. [Matching elements] で Ctrl キーを押しながら `madd` および `mmult` 関数をクリックし、[Qualified name and location] リストに表示されるようにします。[OK] をクリックします。
4. [SDSoC Project Overview] の [Actions] パネルで [Estimate Performance Speedup for HW Functions] をクリックします。これにより、SDEstimate ビルド コンフィギュレーションが選択され、予測フローが実行されます。
5. [Build Project] ダイアログ ボックスに、プロジェクトをビルドするかどうかを尋ねるメッセージが表示されます。[OK] をクリックします。

   SDSoC 環境でプロジェクトがビルドされます。ビルド プロセスのステータスを示すダイアログ ボックスが表示されます。
6. [SDEstimate] の下の `sd_card` フォルダーの内容を SD カードにコピーして、ボードを起動します。ボードがイーサネット ケーブルを使用してイーサネット ルーターに接続されていることを確認します。シリアル ターミナルが接続されていることを確認します。
7. ボードに電源を投入すると、Linux ブート ログがターミナルに表示されます。「Sending select for 172.19.73.248…Lease of 172.19.73.248 obtained」のような、ボードに割り当てられた IP アドレスがレポートされている行を見つけます。

   注記 : このアドレスは、次の手順で使用します。スクロールしてもこの文が見つからない場合は、`ifconfig` コマンドを実行すると、ボードの IP アドレスを取得できます。
8. SDSoC 環境に戻り、[Target Connections] タブで [Linux TCF Agent] を展開表示して [Linux Agent (default)] を右クリックし、[Edit] をクリックします。
9. [Target Connection Details] ダイアログ ボックスで IP アドレスとポート (1534) を設定し、[OK] をクリックします。



10. [SDSoC Report Viewer] タブを開きます。

11. [Click Here] リンクをクリックして、ボードのアプリケーションを起動します。

    [Run application to get its performance] ダイアログ ボックスが表示されます。

12. [Linux Agent] 接続を選択し、[OK] をクリックします。

    SDSoC 環境でアプリケーションのソフトウェアのみのバージョンが実行されます。この後、パフォーマンスデータが収集されて、パフォーマンス予測レポートを表示するのに使用されます。

第 6 章

チュートリアル：タスクのパイプライン処理最適化

このチュートリアルでは、SDSoC 環境で生成されたハードウェア/ソフトウェア システムをタスク レベルのパイプライン処理を使用して最適化するためにコードを変更する方法について説明します。パイプライン処理のパフォーマンスに対する影響を確認します。

注記： チュートリアルは各手順に分けられ、それぞれで大まかな手順が説明された後、細かい手順が説明されていますので、スキル レベルに合った方の手順を参照してください。大まかな手順を終了するのにヘルプが必要な場合は詳細な手順を参照したり、細かい手順を飛ばして次の大まかな手順に進んだりできます。

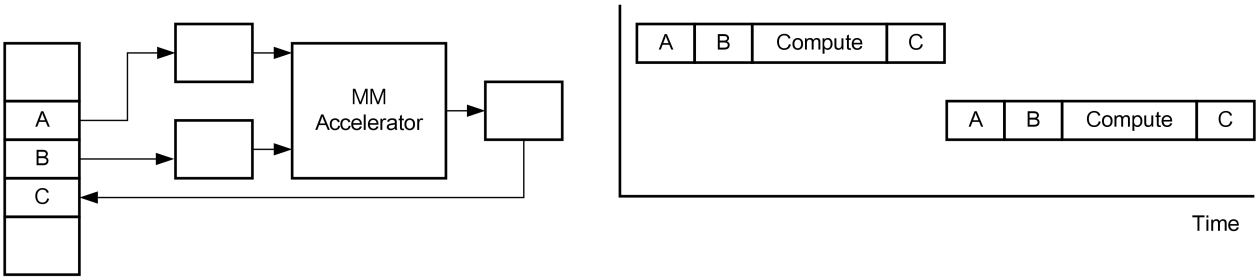
タスクのパイプライン処理

アプリケーションにアクセラレータへの呼び出しが複数ある場合、これらの呼び出しをパイプライン処理して、設定およびデータ通信とアクセラレータ計算が並列処理されるように構成できます。行列乗算アプリケーションの場合は、次のイベントが実行されます。

1. 行列 A と B がメイン メモリからアクセラレータのローカル メモリに転送されます。
2. アクセラレータが実行されます。
3. 結果 C がアクセラレータからメイン メモリに戻されます。

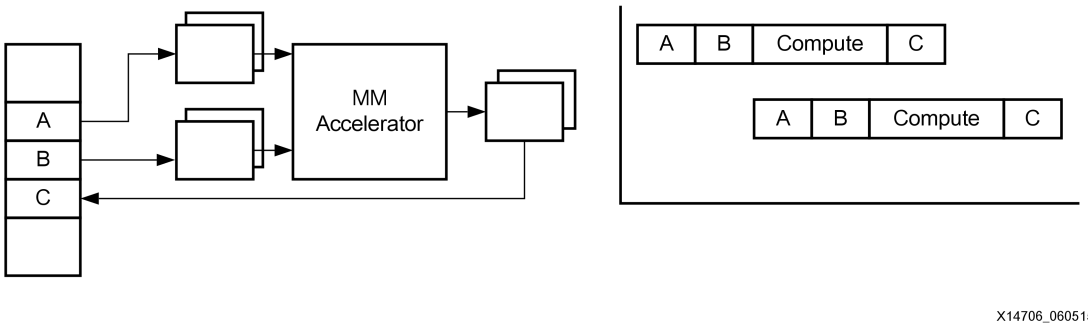
次の図の左側に行列乗算デザイン、右側に順次実行される連続した 2 つの呼び出しに対するこれらのイベントのタイムチャートを示します。

図 6-1：行列乗算呼び出しの順次実行



次の図に、これら 2 つの呼び出しをパイプライン処理して実行した場合を示します。2 番目の呼び出しのデータ転送は、最初の呼び出しのデータ転送が終わるとすぐに開始し、最初の呼び出しの実行と同時に実行されます。ただし、パイプライン処理をイネーブルにするには、アクセラレータが 1 つ目の引数セットを使用して計算を実行している間、2 番目の引数セットを格納しておくローカル メモリが必要です。SDSoC 環境では、ユーザーの指定に基づき、このためにマルチバッファーと呼ばれるメモリが生成されます。

図 6-2： 行列乗算呼び出しのパイプライン実行



タスク レベルのパイプライン処理を指定するには、`async(id)` および `wait(id)` プラグマを使用して呼び出しコードを記述し直す必要があります。SDSoC 環境には、`async` プラグマの使用方法を示すサンプルが含まれており、このチュートリアルではこの行列乗算パイプライン処理のサンプルを使用します。

チュートリアルの目標

このチュートリアルを終了すると、次ができるようになります。

- SDSoC 環境を使用してタスク レベルのパイプライン処理を実行することにより、アプリケーションを実行時間を短縮するよう最適化
- アクセラレータでの計算と入力および出力通信を同時に実行するアクセラレータへのパイプライン呼び出しのパフォーマンスに対する影響を確認

行列乗算サンプルでのタスクのパイプライン処理

SDSoC 環境には、タスク レベルのパイプライン処理をインプリメントする `async` プラグマの使用方法を示す行列乗算のパイプライン処理のサンプルが含まれています。この演習では、この手法を使用することによるランタイムの向上を確認します。

1. [File] → [New] → [SDSoC Project] をクリックし、新しい SDSoC 環境プロジェクト (lab5) を作成します。[Project name] に「`lab5`」と入力し、[Platform] に [zc702]、[OS] に [Linux] を選択して [Next] をクリックします。
2. 選択したプラットフォーム用のソース コード例をリストする [Templates] ページが表示されます。[Available Templates] から [Empty Application] を選択し、[Finish] をクリックします。

3. 使用しているオペレーティング システムのファイル マネージャーを使用して `<path to install>/SDSoC/2015.4/samples/mmult_pipelined` を開き、そのディレクトリにあるソースファイル (`mmult_accel.cpp`、`mmult_accel.h`、`mmult.cpp`) を作成したプロジェクトの `src` フォルダーにコピーします。
4. `mmult_accel.cpp` ファイルの `mmult_accel` を、[SDSoC Project Overview] で [Add Hardware Function] アイコンをクリックするか、[Project Explorer] タブで右クリックして [Toggle HW/SW] をクリックしてマークします。
5. ビルド コンフィギュレーションを [SDRelease] に変更し、プロジェクトをビルドします。

重要 : ビルド プロセスが完了するまでに、約 30 ～ 45 分かかります。プロジェクトをビルドする代わりに、ビルド済みプロジェクトを使用して時間を節約することもできます。ビルド済みプロジェクトをインポートするには、[File] → [Import] をクリックし、[General] → [Existing Projects into Workspace] を選択して [Next] をクリックします。[Select archive file] としてプロジェクト ファイル フォルダーの `lab5_prebuilt.zip` ファイル (`<path to install>/SDSoC/2015.4/docs/labs/lab5_prebuilt.zip`) を指定します。[開く] をクリックします。[Finish] をクリックします。

---

6. `sd_card` フォルダーのファイルを SD カードにコピーし、ターミナルを設定して、ボード上で生成されたアプリケーションを実行します。アプリケーションの引数としてパイプライン段数を指定する必要があります。パイプラン段数を 1、2、および 3 に設定してアプリケーションを実行し、パフォーマンスを記録します。

付録 A

トラブルシューティング

インストール後に SDSoC™ 環境を使用していて問題が発生した場合は、このセクションにリストされている問題およびその回避策を参照してください。

パス名が長すぎる

Windows プラットフォームで Vivado ツールを使用する場合は、パス名が 260 文字を超えると、次のようなエラーメッセージが表示されることがあります。

ERROR: [Common 17-143] Path length exceeds 260-Byte maximum allowed by Windows: <LongPathtoFileName>.

パス名を短くしたり、このエラーが発生しないようにする方法については、アンサー 52787 を参照してください。たとえば、短いパス名を使用したり、新しいドライブ文字をパスの下位ディレクトリにマップしたりといった方法があります。

誤ったツール スクリプトの使用

SDSoC™ 環境を実行するのに使用される各シェルでは、ザイリンクス ツール リリースまたは推奨される PATH 環境設定に対応する環境設定スクリプトのみを使用してください。

同じシェル内のほかのリリースまたはその他のリリースからザイリンクス デザイン ツールの環境設定スクリプトを実行すると、SDSoC 環境の動作または結果が正しくなくなります。

付録 B

その他のリソースおよび法的通知

ザイリンクス リソース

アンサー、資料、ダウンロード、フォーラムなどのサポート リソースについては、ザイリンクス サポート サイトを参照してください。

ソリューション センター

デバイス、ツール、IP のサポートについては、ザイリンクス ソリューション センターを参照してください。デザイン アシスタント、アドバイザリ、トラブルシューティングのヒントなどが含まれます。

参考資料

このガイドの補足情報は、次の資料を参照してください。

日本語版のバージョンは、英語版より古い場合があります。

1. 『SDSoC 環境ユーザー ガイド : SDSoC 環境の概要』(UG1028) (SDSoC 環境の docs フォルダーからも入手可能)
2. 『SDSoC 環境ユーザー ガイド』(UG1027) (SDSoC 環境の docs フォルダーからも入手可能)
3. 『SDSoC 環境ユーザー ガイド : プラットフォームおよびライブラリ』(UG1146) (SDSoC 環境の docs フォルダーからも入手可能)
4. 『UltraFast エンベデッド デザイン設計手法ガイド』(UG1046 : 英語版、日本語版)
5. 『ZC702 評価ボード (Zynq-7000 XC7Z020 All Programmable SoC 用) ユーザー ガイド』(UG850)
6. 『Vivado Design Suite ユーザー ガイド : 高位合成』(UG902)
7. 『PetaLinux ツール資料ワークフロー チュートリアル』(UG1156)
8. Vivado® Design Suite 資料
9. 『Vivado Design Suite ユーザー ガイド : カスタム IP の作成とパッケージ』(UG1118)

お読みください：重要な法的通知

この資料に関するフィードバックおよびリンクなどの問題につきましては、jpn_trans_feedback@xilinx.com まで、または各ページの右下にある [フィードバック送信] ボタンをクリックすると表示されるフォームからお知らせください。フィードバックは日本語で入力可能です。いただきましたご意見を参考に早急に対応させていただきます。なお、このメール アドレスへのお問い合わせは受け付けておりません。あらかじめご了承ください。