SDSoC 環境チュートリアル

入門

UG1028 (v2017.1) 2017 年 6 月 20 日

この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
2017 年 6 月 20 日	2017.1	・ SDx™ IDE 2017.1 用にアップデート。 ・ 「エミュレーション」および「GitHub 」チュートリアルを追加。

目次

概要

フローの概要

演習 1: SDSoC 開発環境の概要 5

パフォーマンスの見積もり

演習 2: パフォーマンスの見積もり 17

アプリケーション コードの最適化

演習 3: アプリケーション コードの最適化 24

アクセラレータの最適化

演習 4: 指示子を使用したアクセラレータの最適化 34

演習 5: タスク レベルのパイプライン 36

デバッグ

演習 6: デバッグ 40

演習 7: ハードウェア デバッグ 45

エミュレーション

演習 8: エミュレーション 56

GitHub

演習 9: GitHub からのアプリケーションのインストール 60

その他のリソースおよび法的通知

参考資料 63

お読みください: 重要な法的通知 64

XILINX
ALL PROGRAMMABLE™

第 1 章

概要

SDSoC™ (Software-Defined Development Environment for System-on-Chip) 環境は、Zynq®-7000 All Programmable SoC および Zynq UltraScale+™ MPSoC 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++/OpenCL 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++/OpenCL 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++/OpenCL 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 2017.1 リリースでは、Zynq-7000 All Programmable SoC を搭載した ZC702、ZC706、MicroZed、ZedBoard、および Zybo 開発ボード、Zynq UltraScale+ MPSoC を搭載した ZCU102 開発ボードがサポートされています。その他のプラットフォームは、パートナーから入手できます。詳細は、SDSoC 環境のウェブページ を参照してください。

XILINX
ALL PROGRAMMABLE™

第 2 章

フローの概要

演習 1: SDSoC 開発環境の概要

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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDSoC 環境プロジェクトを作成する際に、推奨されている [Matrix Multiplication and Addition] テンプレートが見つからない場合は、ボードと使用可能なアプリケーションの 1 つを選択してください。たとえば、より小型の Zynq-7000 デバイスを含む MicroZed ボードなどのボードの場合、使用可能なテンプレートに [Matrix Multiplication and Addition (area reduced)] アプリケーションがリストされます。このチュートリアルの学習目標は、どのアプリケーションでも達成できるようになっています。

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。

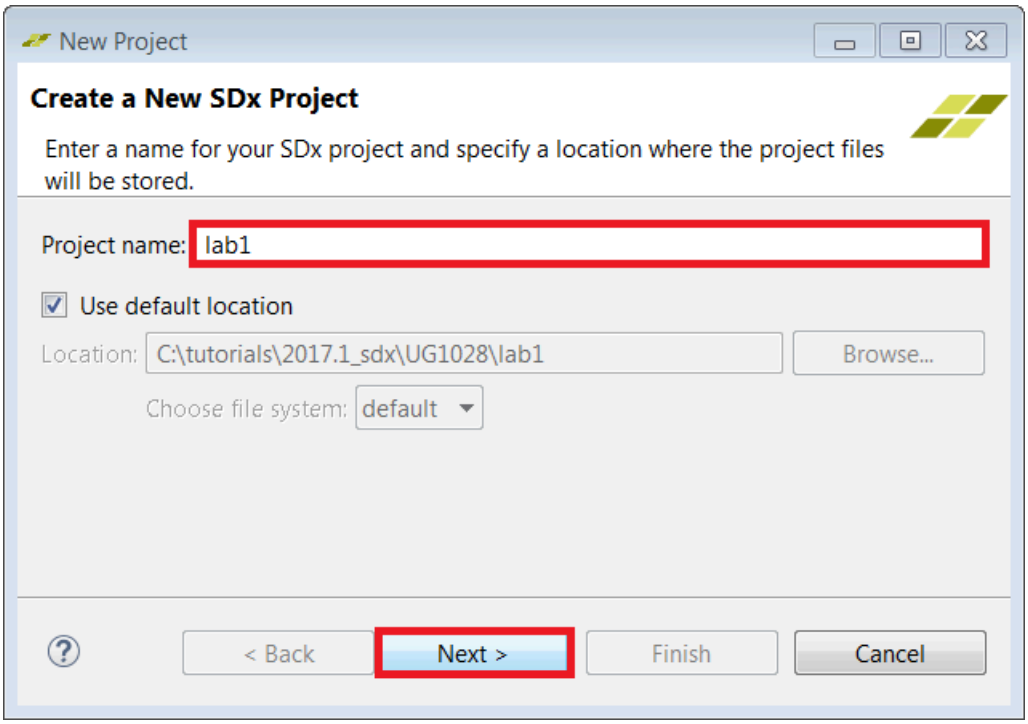
- 多くの使用可能なプラットフォームおよびプロジェクト テンプレートから、ユーザー アプリケーション用の新しい SDSoC 環境プロジェクトを作成
- ハードウェア インプリメンテーション用の関数をマーク
- ハードウェアにインプリメントされる関数を含むビットストリームと、このハードウェアでインプリメントされる関数を開始する実行ファイルを生成するプロジェクトをビルド

新規プロジェクトの作成

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、SDx IDE 2017.1 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. SDx IDE 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 SDx Project]、[Add Custom Platform]、[Create Application Project]、[Import Project]、[Tutorials]、および [Web Resources] などのリンクが含まれます。これらのリンクのいずれかをクリックすると、各リンクの下に使用可能なオプションがさらに

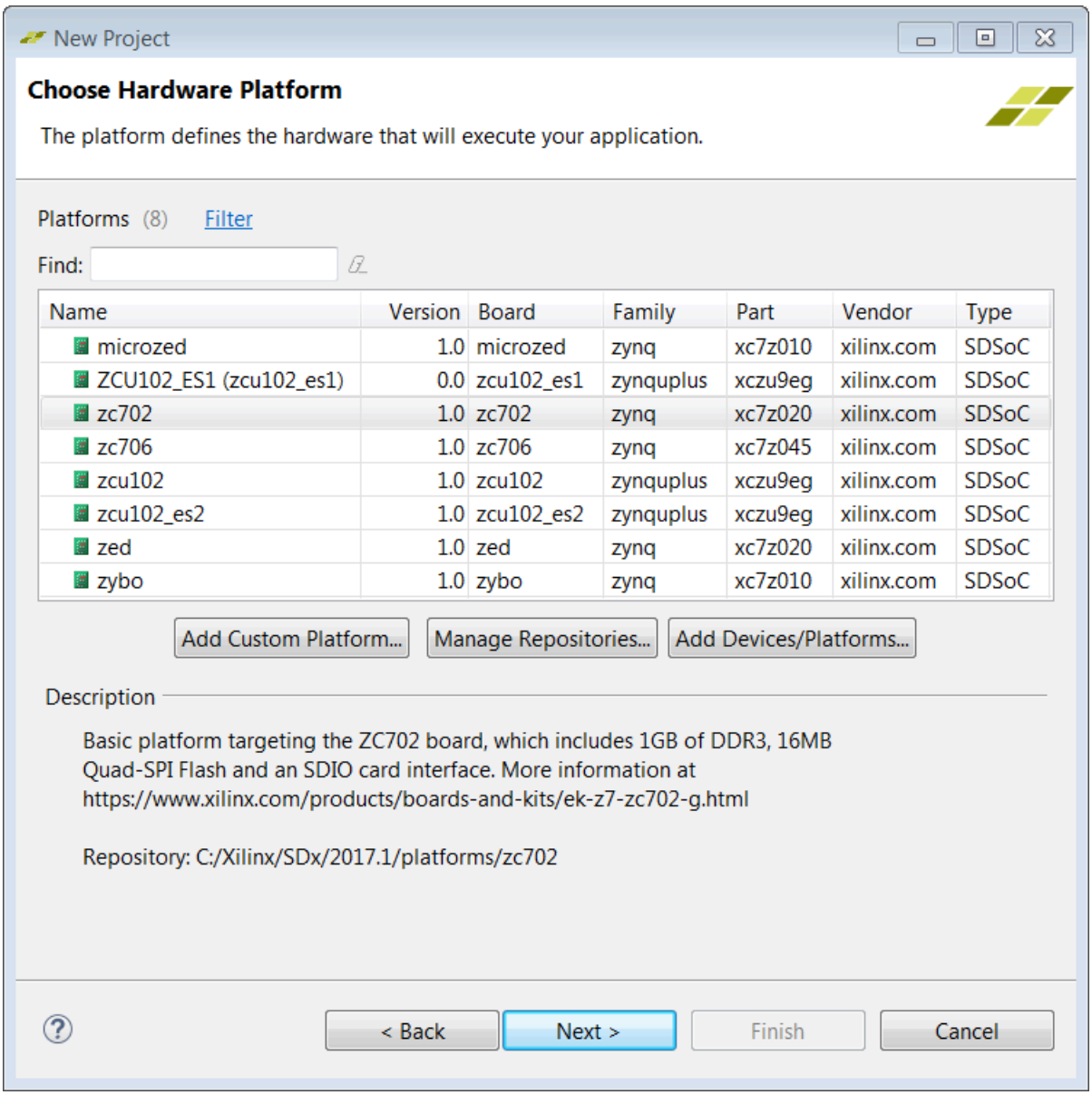
表示されます。たとえば、資料およびチュートリアルを入手するには、[Tutorials] をクリックすると、SDSoC および SDAccel の関連資料へのリンクを含むチュートリアル ページが表示されます。[Welcome] タブは、[X] をクリックして閉じるか、[Minimize] アイコンをクリックして最小化できます。

4. SDx IDE メニュー バーから [File] → [New] → [Xilinx SDx Project] をクリックします。[New Project] ダイアログ ボックスが開きます。



5. プロジェクト名を指定します (例: lab1)。
6. [Next] をクリックします。

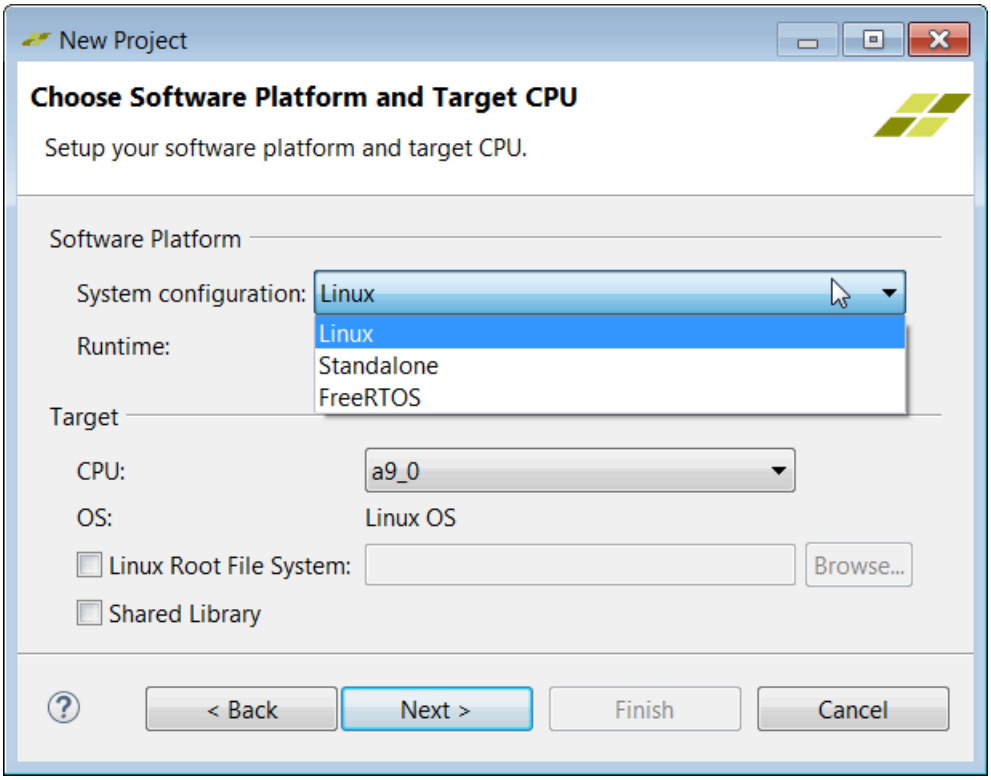
7. [Choose Hardware Platform] ページから zc702 プラットフォームを選択します。



注記: カスタム プラットフォームを使用する場合は、サポートされるプラットフォームにはリストされないので、[Add Custom Platform] をクリックしてカスタム プラットフォームを追加します。

8. [Next] をクリックします。

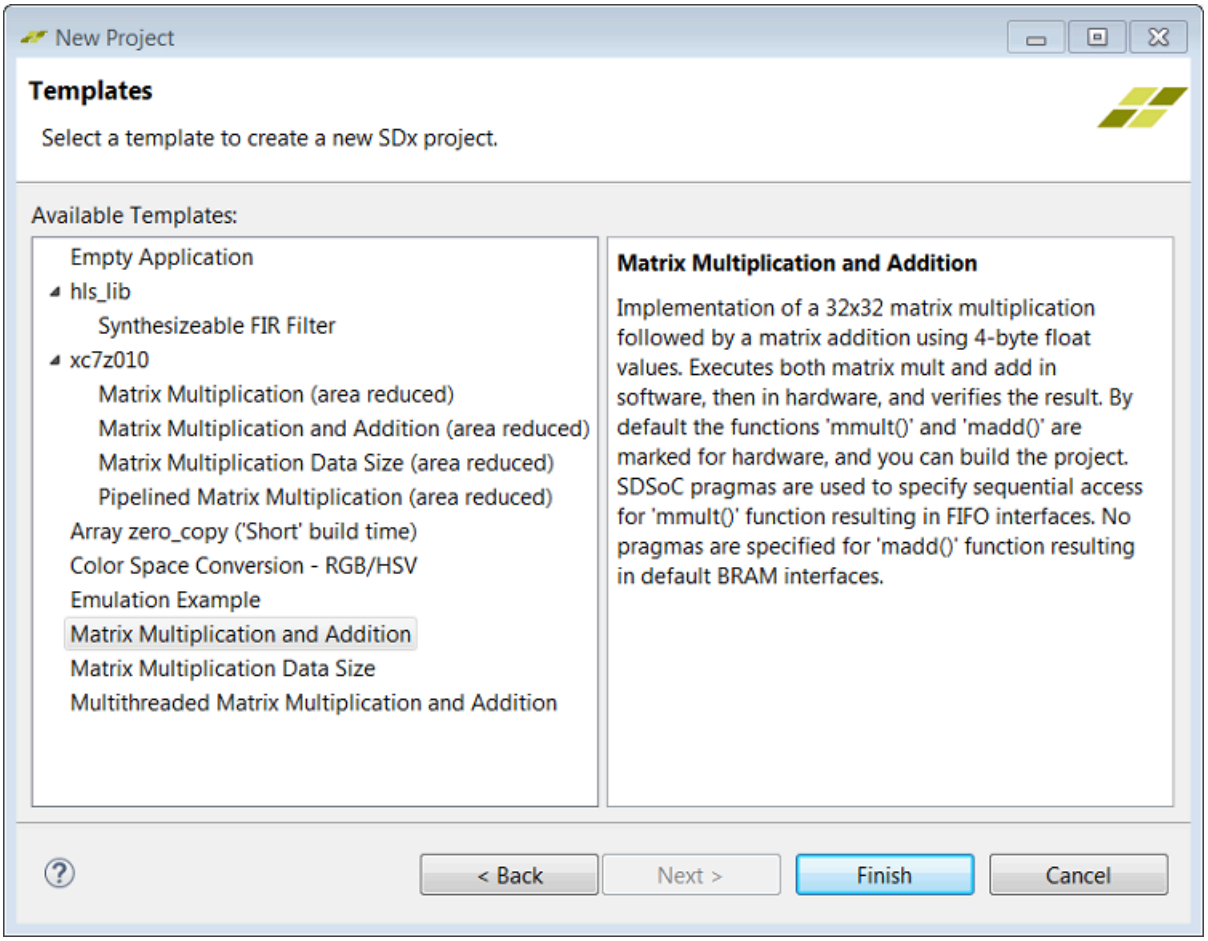
9. [System configuration] ドロップダウン リストから [Linux] を選択します。その他のフィールドはすべてデフォルトのままにしておきます。



10. [Next] をクリックします。

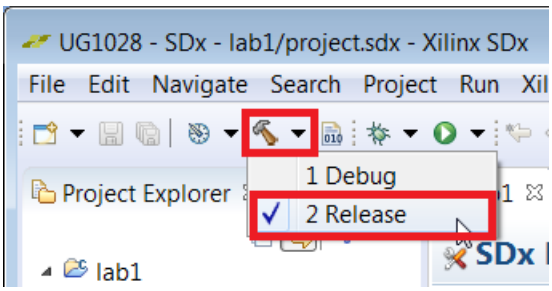
選択したプラットフォーム用のソース コード例をリストする [Templates] ページが表示されます。

11. [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。

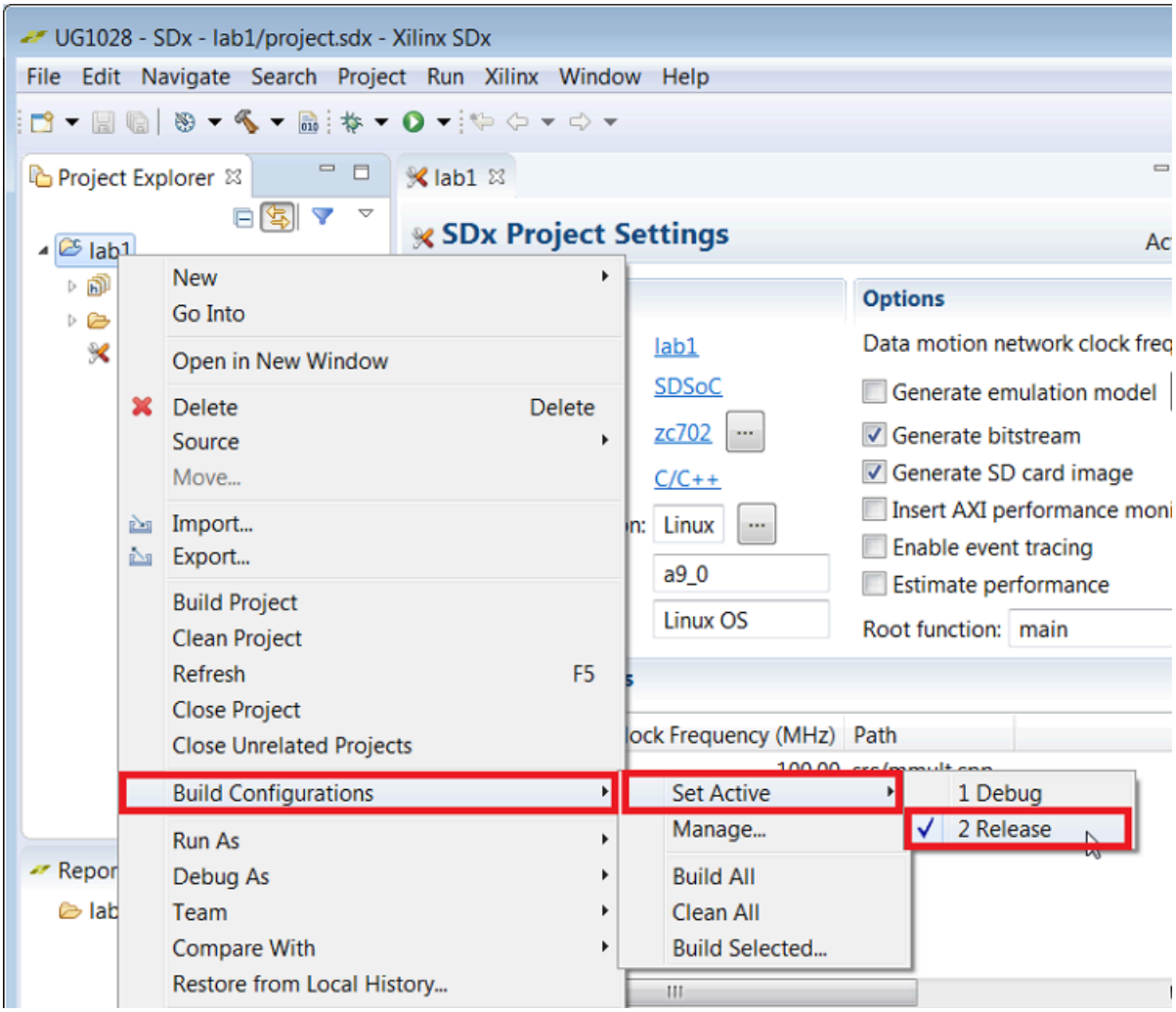


12. 標準ビルド コンフィギュレーションは Debug および Release で、追加のビルド コンフィギュレーションも作成できます。最高のランタイム パフォーマンスにするには、[Release] コンフィギュレーションに切り替えます。[Release] ビルド コンフィギュレーションでは、[Debug] ビルド コンフィギュレーションよりも高いコンパイラ最適化設定が使用されます。[Sdx Project Settings] では、アクティブ コンフィギュレーションを選択したり、ビルド コンフィギュレーションを作成したりできます。

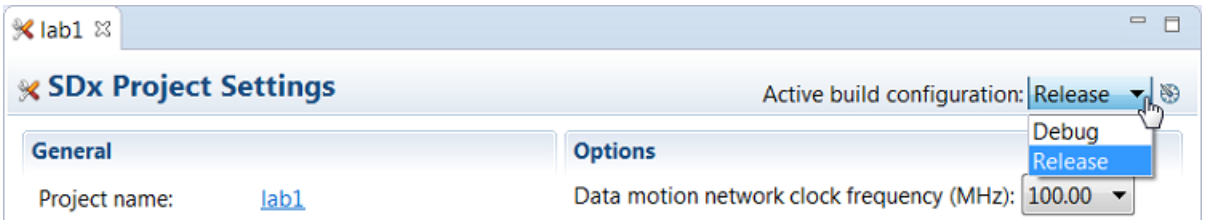
    [Build] アイコンには、ビルド コンフィギュレーションを選択するドロップダウン リストがあります。[Build] アイコンをクリックすると、プロジェクトがビルドされます。



    [Project Explorer] ビューでプロジェクトを右クリックし、ビルド コンフィギュレーションを選択できます。



    [SDx Project Settings] での [Build Configuration] ドロップダウンで、アクティブ コンフィギュレーションを選択したり、ビルド コンフィギュレーションを作成したりできます。



[SDx Project Settings] にプロジェクト設定のサマリが表示されます。

SDx 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。[Debug] 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。[Release] ビルドでは、同じ ELF ファイルが Debug コンフィギュレー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。ビルド コンフィギュレーションに対して [Estimate Performance] をオンにし、そのビルド コンフィギュレーションを使用してアプリケーションのパフォーマンス見積もりを実行するモードを実行するには、別の設定および手順が必要となります。詳細は、パフォーマンスの見積もり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

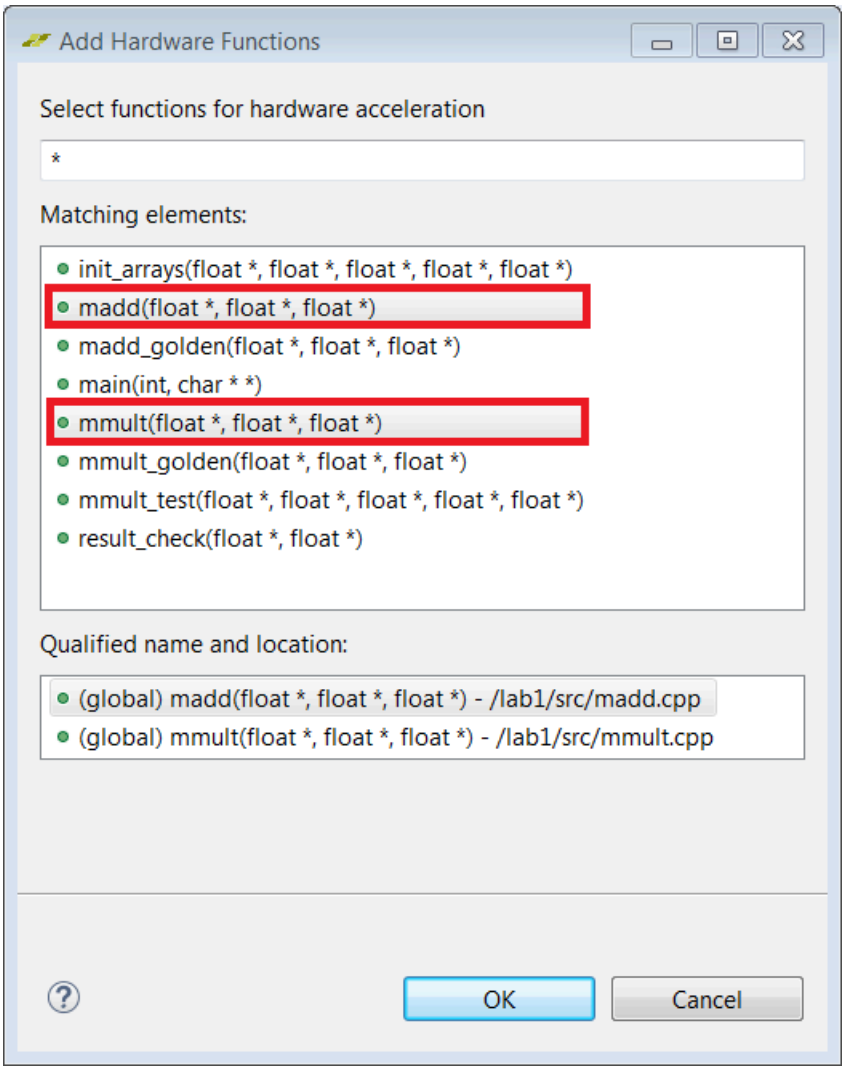
このアプリケーションには、2 つのハードウェア関数が含まれます。1 つは `mmult` で、2 つの行列を乗算して行列積を算出します。もう 1 つは `madd` で、2 つの行列を加算して行列和を算出します。これらのハードウェア関数がまとめられ、行列の乗加算関数を計算します。`mmult` と `madd` の両方の関数をハードウェアでインプリメントされるように指定します。

テンプレートからプロジェクトを作成すると、ハードウェア関数が自動的に指定されます。ハードウェア関数が削除されたり、指定されていない場合は、次の手順でハードウェア関数を追加します。

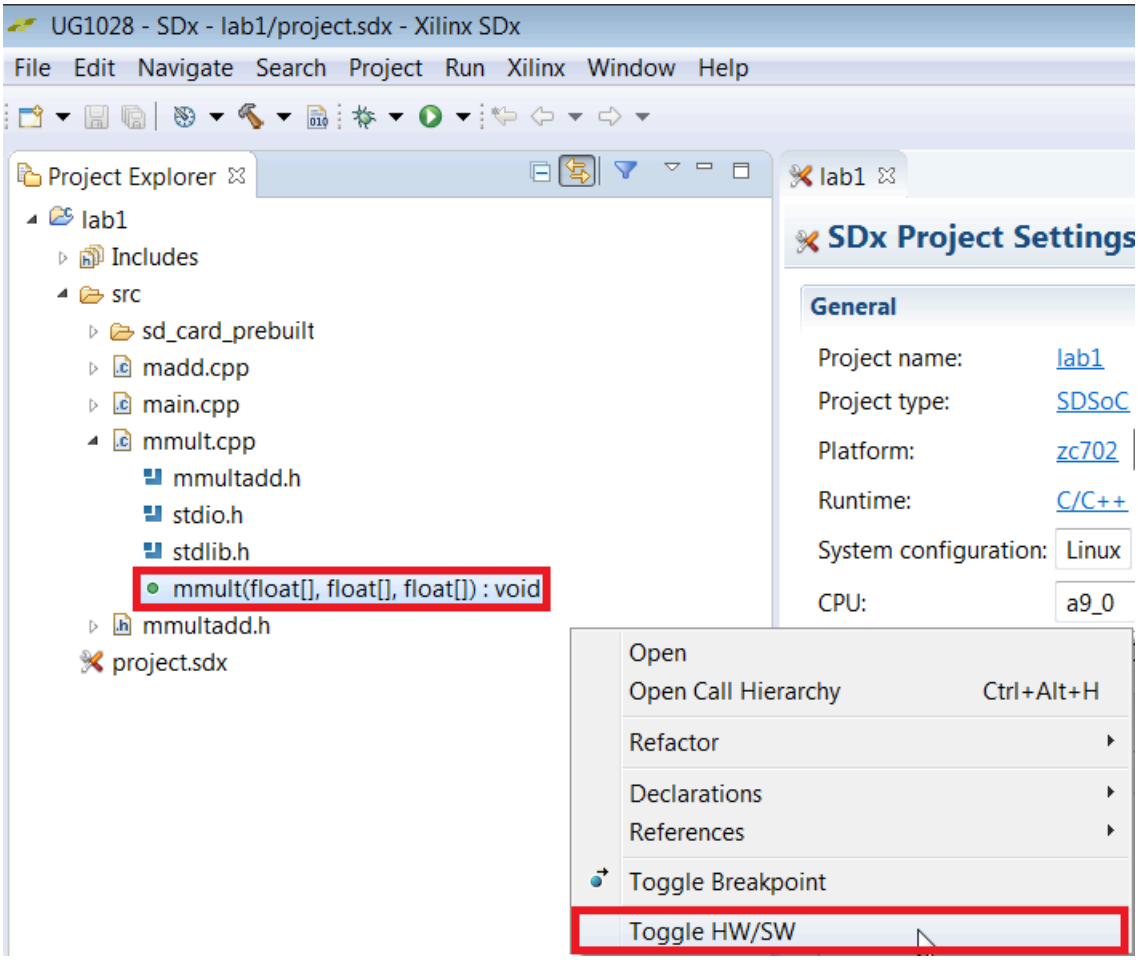
注記: この演習では、ハードウェア用に関数をマークする必要はありません。行列乗算と加算のテンプレート コードは既にハードウェア用にマークされています。`madd` および `mmult` 関数がハードウェア関数としてマークされていない場合は、次の手順でそれらをハードウェア関数としてマークできます。

1. [SDx Project Settings] では、プロジェクトの値を設定できます。[lab1] タブをクリックし (タブが開いていない場合は、[Project Explorer] ビューで project.sdx ファイルをダブルクリック)、[HW functions] パネルで [Add HW Functions] 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 ボックスを開きます。

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 [HW 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、[Project Explorer] に `mmult(float[], float[], float[]): void [H]` 関数が表示されます。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

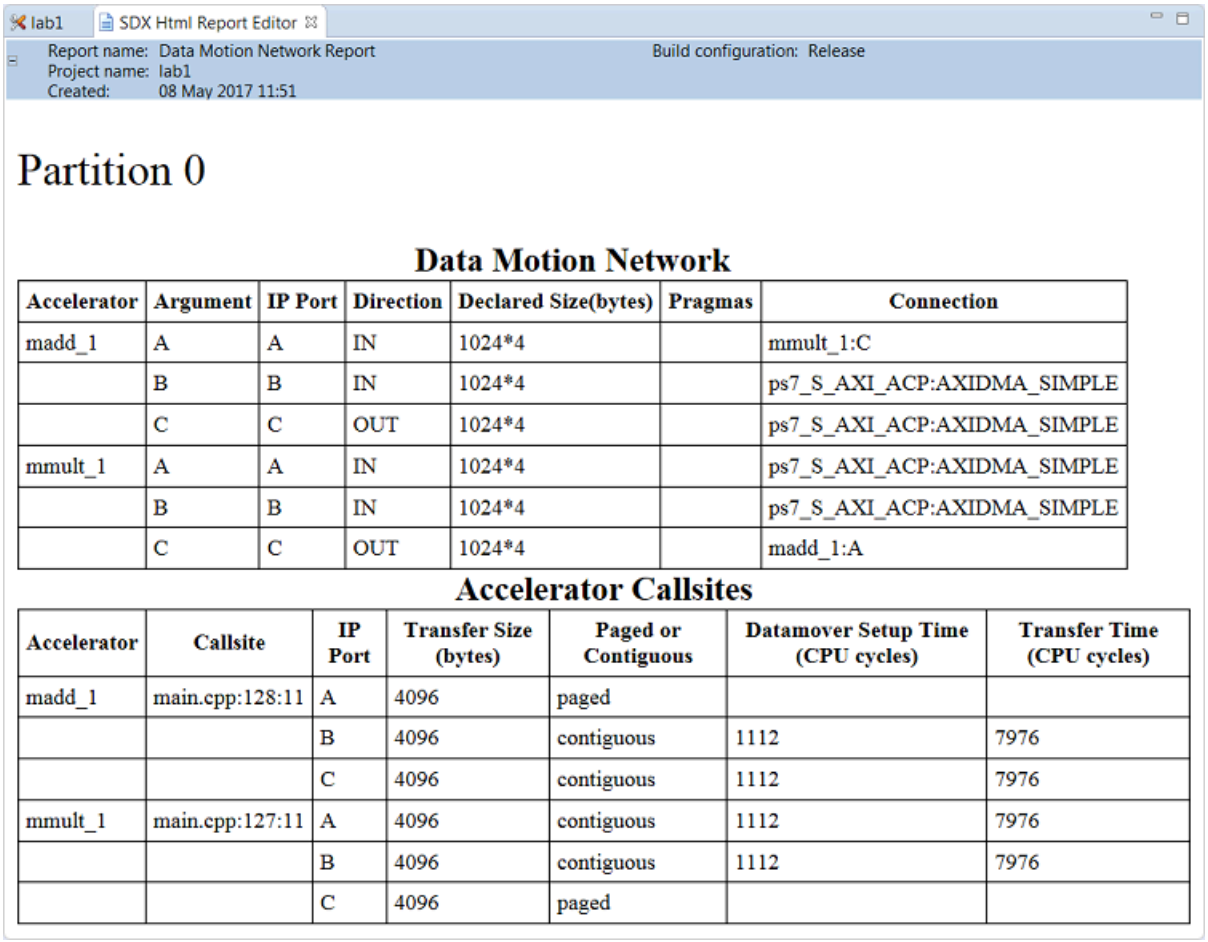
プロジェクトをビルドして実行ファイル、ビットストリーム、SD カード ブート イメージを生成するには、次の手順に従います。

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. [SDx Project Settings] ウィンドウの [Project Explorer] の下の [Reports] タブで [Data Motion Network Report] をダブルクリックして開きます。

   このレポートには、SDx 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、アプリケーション コードの最適化を参照してください。


lab1 SDX Html Report Editor

Report name: Data Motion Network Report
Project name: lab1
Created: 08 May 2017 11:51
Build configuration: Release

Partition 0

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_1	A	A	IN	1024*4		mmult_1:C
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
mmult_1	A	A	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		madd_1:A

Accelerator Callsites

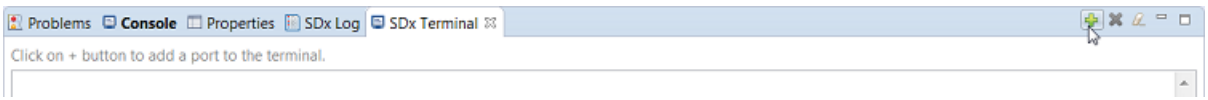
Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time (CPU cycles)	Transfer Time (CPU cycles)
madd_1	main.cpp:128:11	A	4096	paged		
		B	4096	contiguous	1112	7976
		C	4096	contiguous	1112	7976
mmult_1	main.cpp:127:11	A	4096	contiguous	1112	7976
		B	4096	contiguous	1112	7976
		C	4096	paged		

3. `lab1/Release/_sds/swstubs/mmult.cpp` を開き、SDx システム コンパイラにより元の `mmult` 関数が `cf_send_i` および `cf_wait` 関数を使用した FPGA に対して入出力転送を実行する `_p0_mmult_1_noasync` 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/Release/_sds/swstubs/main.cpp` 内の `_p0_mmult_1_noasync` に置き換えられます。SDx 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

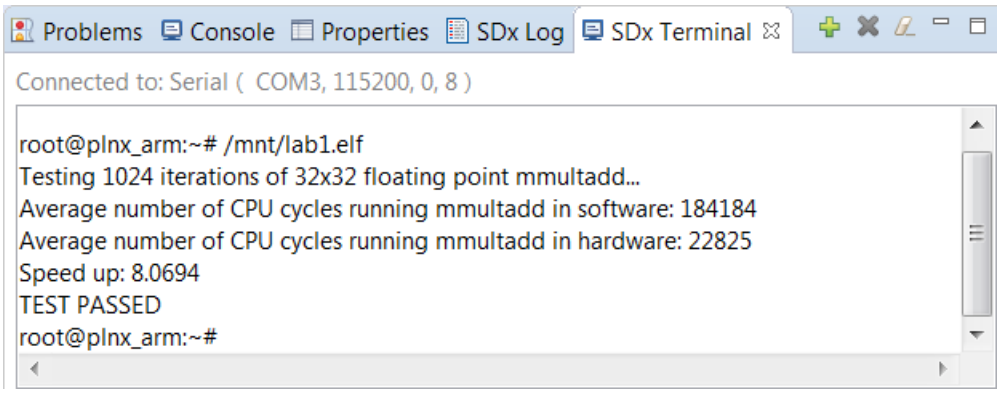
ZC702 ボードでプロジェクトを実行する手順は、次のとおりです。

1. [Project Explorer] タブで `lab1/Release` 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. [SDx Terminal] タブでシリアル ターミナルからボードに接続します。または、[Baud Rate]: 115200、[Data bits]: 8、[Stop bits]: 1、[Parity]: None、[Flow Control]: None に設定した Putty/Teraterm を介して接続します。 アイコンをクリックして設定を開きます。


Problems Console Properties SDx Log SDx Terminal
Click on + button to add a port to the terminal.

4. [Connect to serial port] ウィンドウの設定はそのままで [OK] をクリックします。

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.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。
- SDx™ 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。
- `Release/_sds` フォルダー内の `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 `Release/_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 8 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 4 千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 2 万 2 千サイクルかかります。

- sdscc, sds++, xocc, arm-linux-gnueabihf-gcc, arm-linux-gnueabihf-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。
  - xocc: OpenCL 言語ソースをコンパイルするのに使用されます。
  - arm-linux-gnueabihf-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-linux-gnueabihf-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。

XILINX
ALL PROGRAMMABLE™

第 3 章

パフォーマンスの見積もり

演習 2: パフォーマンスの見積もり

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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDSoC プロジェクトを作成する際に、推奨されている [Matrix Multiplication and Addition] テンプレートが見つからない場合は、ボードと使用可能なテンプレートの 1 つを選択してください。たとえば、より小型の Zynq-7000 デバイスを含む MicroZed ボードなどのボードの場合、使用可能なテンプレートに [Matrix Multiplication and Addition (area reduced)] アプリケーションがリストされます。このチュートリアルの学習目標は、アプリケーションが存在していれば、別のアプリケーションを使用しても達成できます。アプリケーションは、ボードで実装済みアプリケーションを実行して、ソフトウェア ランタイム データの収集するために必ず必要です。設定情報の詳細は、ご使用のボードの資料を参照してください。

チュートリアルの目標

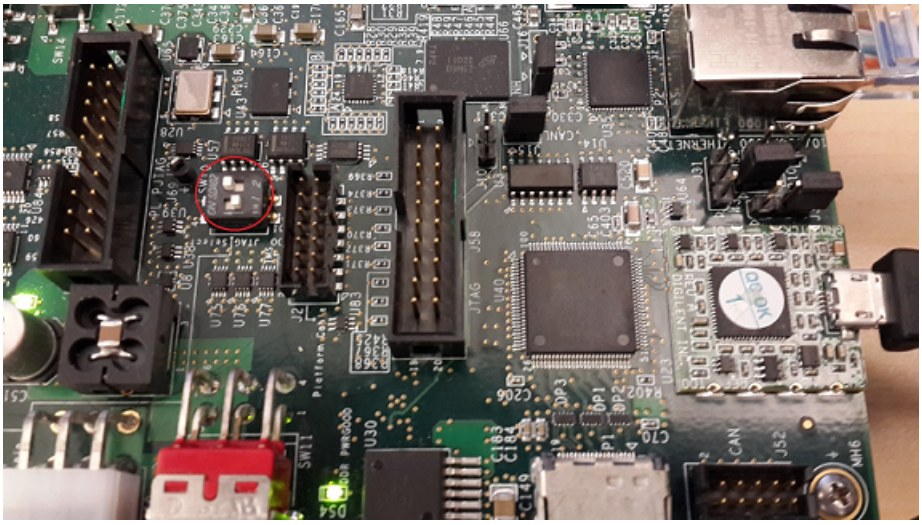
チュートリアルを終了すると、SDSoC 環境を使用して、ハードウェアにインプリメントされた関数の選択に基づいてスピードアップを見積もることができるようになります。

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 SDx IDE 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。

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. DIP スイッチ (上の図の赤丸) 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。
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、SDx IDE がボードと通信できるようにします。

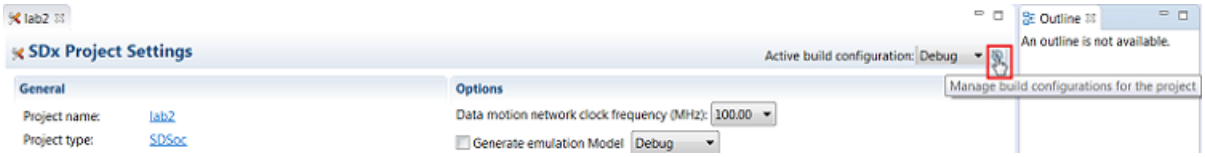
重要: ボードのジャンパーが SD ブートまたは JTAG ブートに設定されていることを確認します。このようにしておかないと、ボードが QSPI ブートなどのその他のモードでパワーアップし、QSPI デバイスまたはその他のブート デバイスからこの演習に関係のないものが読み込まれてしまいます。

パフォーマンス見積もり用のプロジェクトのセットアップ

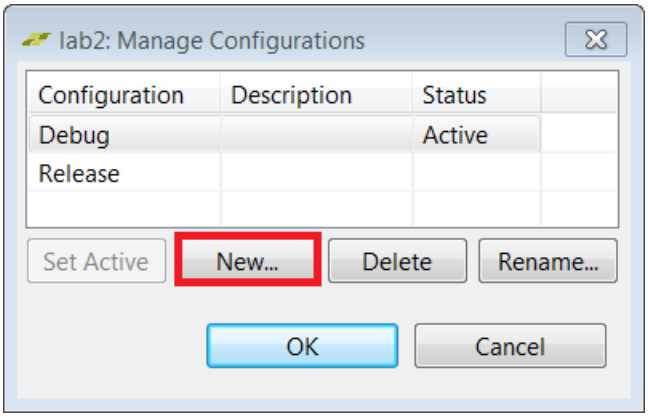
ビルド コンフィギュレーションにプロジェクトを作成して [Estimate Performance] オプションを作成する手順は、次のとおりです。

1. [Matrix Multiplication and Addition] デザイン テンプレートを使用して、プラットフォームに [zc702]、システム コンフィギュレーションに [Standalone] を選択して新しい SDx™ IDE 2017.1 プロジェクト (lab2) を作成します。
2. [lab2] タブをクリックして [SDx Project Settings] を開きます。タブが表示されていない場合は、[Project Explorer] タブの [lab2] プロジェクトの下の `project.sdx` ファイルをダブルクリックします。
3. [HW functions] パネルで `madd` および `mmult` 関数が既にハードウェア用にマークされていることを確認します。SDx 環境のテンプレート プロジェクトには、ハードウェア関数をマークするプロセスを自動化するための情報が含まれています。
4. [HW functions] パネルに関数がリストされていない場合は、[Add HW 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。[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。
5. パフォーマンス見積もりは、どのビルド コンフィギュレーションを使用しても実行できます。アクティブ コンフィギュレーションに [Debug] または [Release] コンフィギュレーションを選択する代わりに、アクティブ コンフィギュレーションの隣の [Manage build configuration for the project] アイコンをクリックする

こともできます。



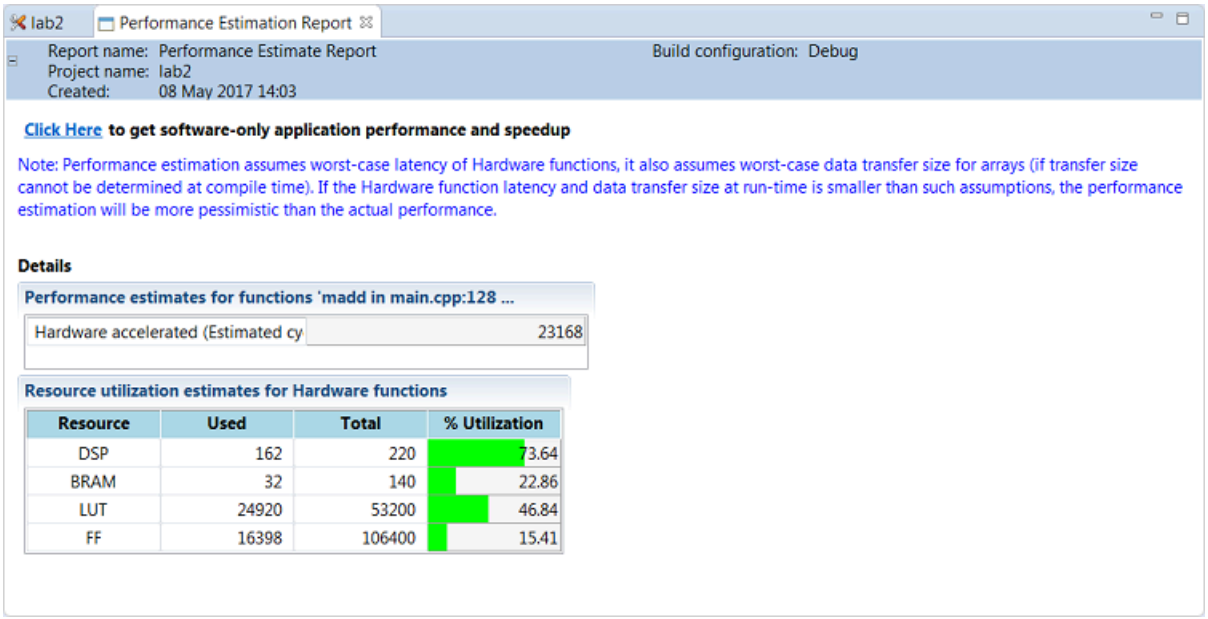
6. 使用可能なコンフィギュレーションを選択したり、新しいコンフィギュレーションを作成したりできます。新しいコンフィギュレーションは既存のコンフィギュレーションを基に作成したり、最初から作成したりできます。[Debug] ビルド コンフィギュレーションまたは [Debug] からコピーした別のビルド コンフィギュレーションを使用すると、GCC を使用して -O0 でコードがコンパイルされるので、ソフトウェア パフォーマンスがかなり低下します。この演習では、[Debug] を選択します。



7. [SDx Project Settings] の [Options] パネルで [Estimate Performance] をオンにします。これによりパフォーマンス見積もりフローがオンになります。
8. [Build] ツールバー ボタンには、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。[Estimate Performance] オプションをオンにしている場合は、パフォーマンスの見積もりも実行されます。ツールバーの [Build] ボタンをクリックします。

   SDx IDE 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。



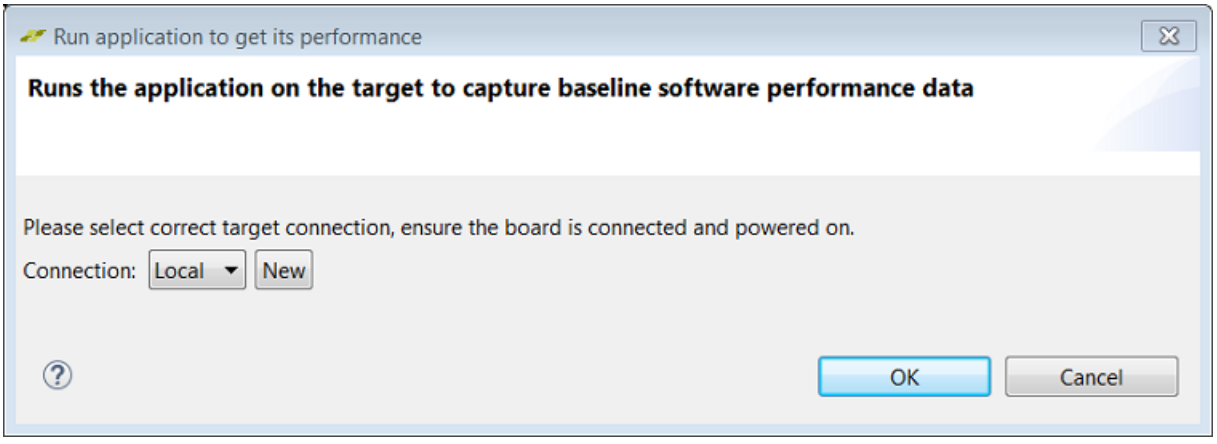
ソフトウェアとハードウェアのパフォーマンス比較

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見積もりレポートを生成するには、次の手順に従います。

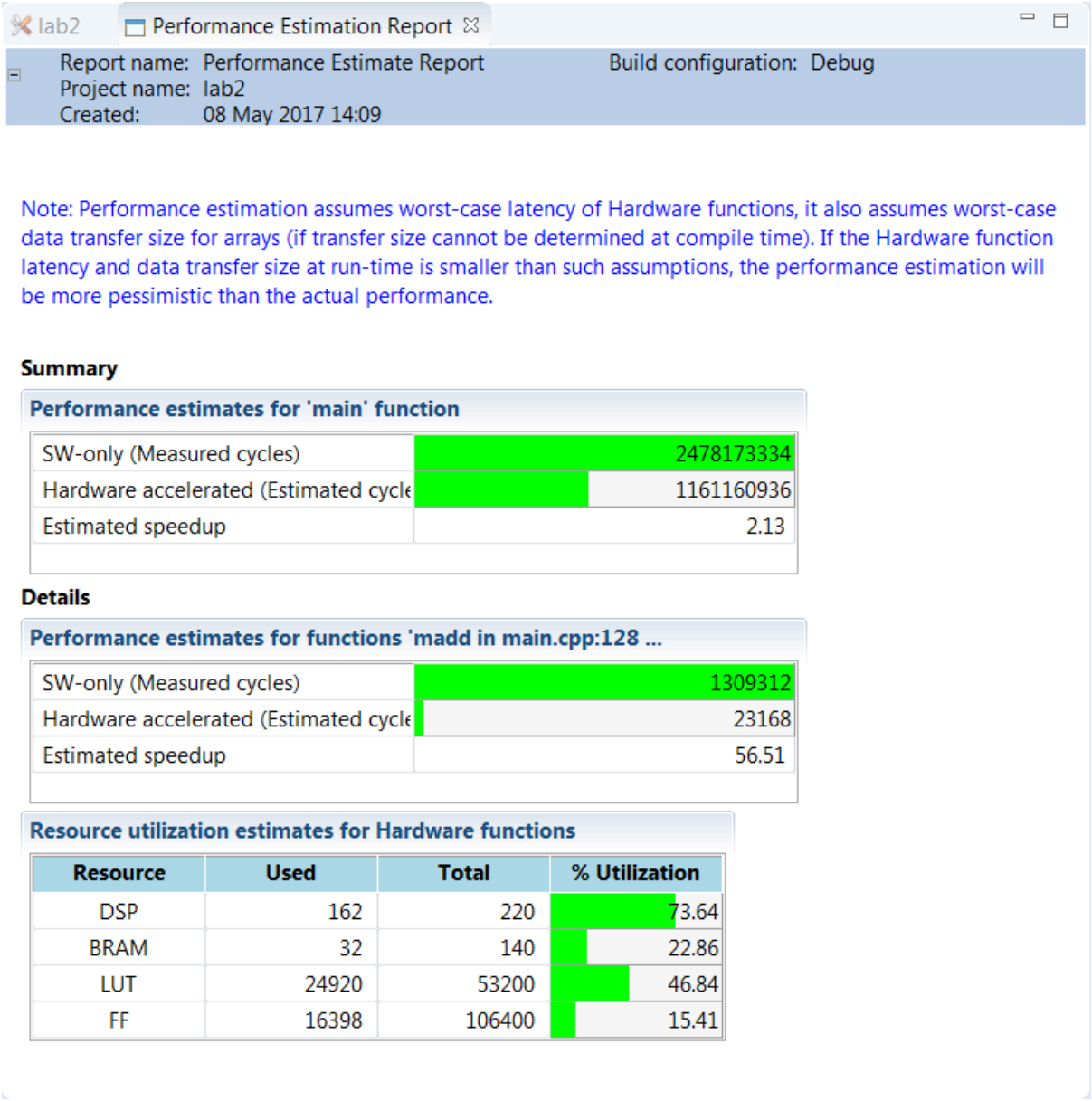
1. ビルドが終了したら、[SDSoC Report Viewer] タブが開きます。
2. [Click Here] リンクをクリックして、ボードのアプリケーションを起動します。

   [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、パフォーマンス見積もりレポートを表示するのに使用されます。


lab2
Performance Estimation Report
Report name: Performance Estimate Report
Project name: lab2
Created: 08 May 2017 14:09
Build configuration: Debug

Note: Performance estimation assumes worst-case latency of Hardware functions, it also assumes worst-case data transfer size for arrays (if transfer size cannot be determined at compile time). If the Hardware function latency and data transfer size at run-time is smaller than such assumptions, the performance estimation will be more pessimistic than the actual performance.

Summary

Performance estimates for 'main' function

SW-only (Measured cycles)	2478173334
Hardware accelerated (Estimated cycle	1161160936
Estimated speedup	2.13

Details

Performance estimates for functions 'madd in main.cpp:128 ...

SW-only (Measured cycles)	1309312
Hardware accelerated (Estimated cycle	23168
Estimated speedup	56.51

Resource utilization estimates for Hardware functions

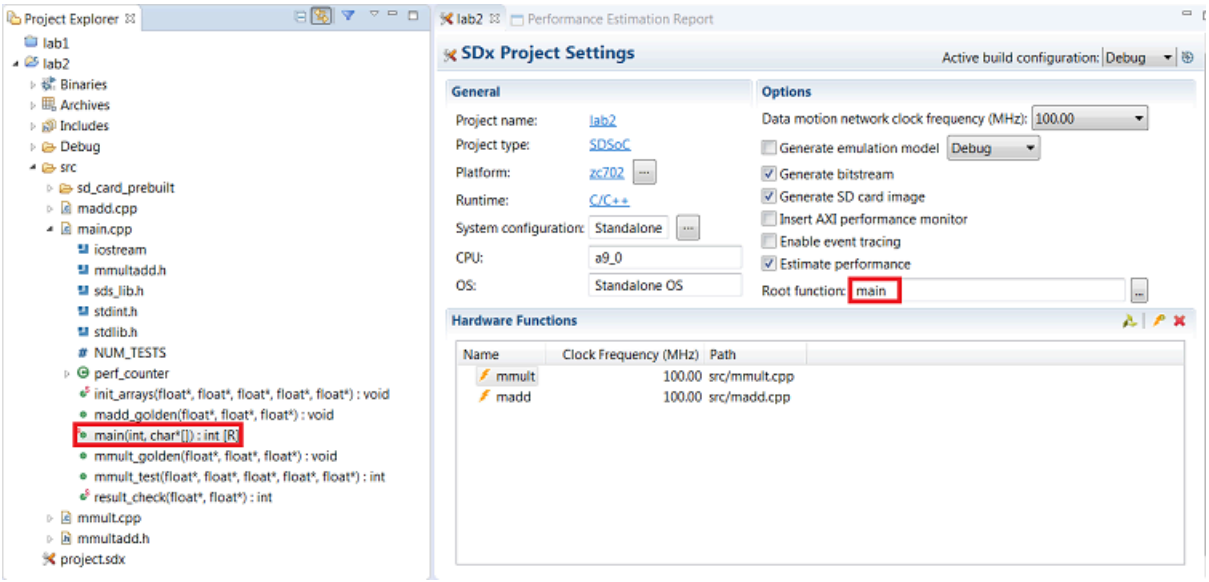
Resource	Used	Total	% Utilization
DSP	162	220	73.64
BRAM	32	140	22.86
LUT	24920	53200	46.84
FF	16398	106400	15.41

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、スピードアップ、およびリソース見積もりレポートのサマリには、最上位関数 (`perf root`) のスピードアップの見積もりが示されます。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、この比較から除外する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、パフォーマンス見積もり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. [SDx Project Settings] ウィンドウで [Root function] フィールドの参照ボタンをクリックし、見積もりフローのルートを変更します。

   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。[SDx Project Settings] で [Estimate performance] をオンにし、見積もり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の見積もりを取得します。

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、プラットフォームを [zc702]、システム コンフィギュレーションを [Linux SMP] に指定して、新しい SDx™ IDE プロジェクト (lab2_linux) を作成します。
2. [lab2_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 lab2_linux プロジェクトの `project.sdx` をダブルクリックします。[HW functions] パネルで `madd` および `mmult` 関数が既にハードウェア用にマークされていることを確認します。SDx 環境のテンプレート プロジェクトには、ハードウェア関数をマークするプロセスを自動化するための情報が含まれています。
3. [HW functions] パネルに関数がリストされていない場合は、[Add HW Functions] 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。[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。
4. [SDx Project Settings] の [Options] パネルで [Estimate performance] をオンにします。これにより、現在のビルド コンフィギュレーションでパフォーマンス見積もりフローがオンになります。
5. [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。

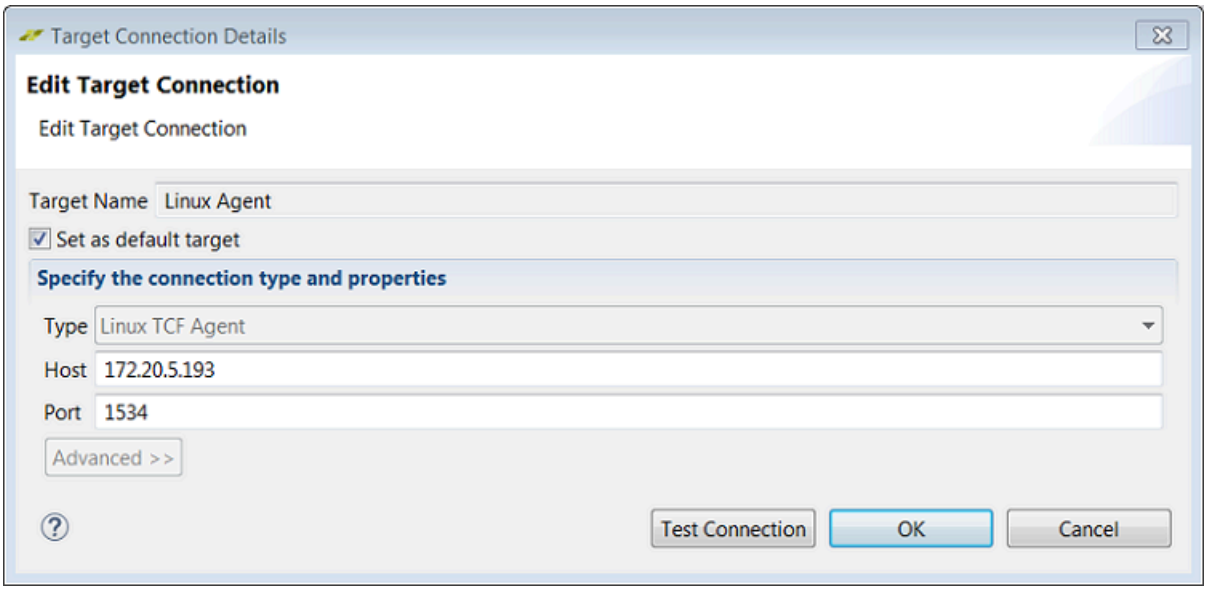
[Build] アイコンをクリックするとプロジェクトがビルドされ、[Estimate performance] がオンになっているとパフォーマンス見積もりフローが実行されます。[Build] をクリックします。

SDx IDE 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. この演習には、ボードに接続されたイーサネット ケーブルも必要です。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。まず、ビルド コンフィギュレーションの下の `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。シリアル ターミナルも接続されていることを確認します。
7. 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`」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。プロンプトのターミナル ウィンドウで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. SDx IDE 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。

    SDx IDE 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。

XILINX
ALL PROGRAMMABLE™

第 4 章

アプリケーション コードの最適化

演習 3: アプリケーション コードの最適化

このチュートリアルでは、SDx 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。

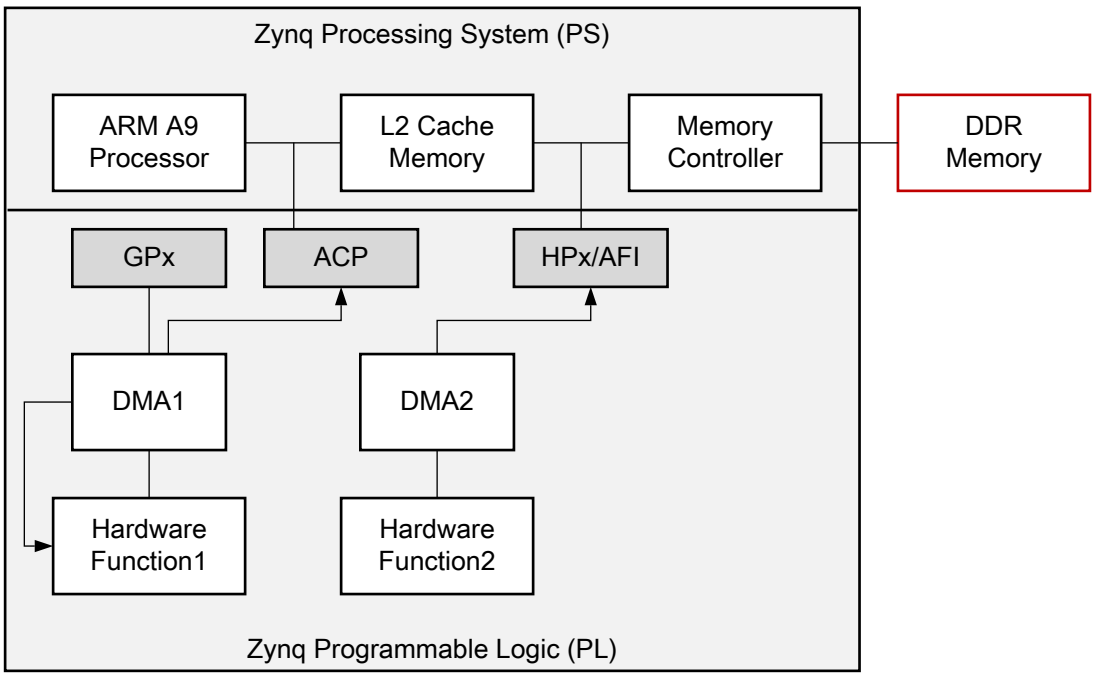
注記: この演習は、ZC702 ボードがなくても終了できます。SDSoC 環境プロジェクトを作成する際に、推奨されている [Matrix Multiplication and Addition] テンプレートが見つからない場合は、ボードと使用可能なアプリケーションの 1 つを選択してください。たとえば、より小型の Zynq-7000 デバイスを含む MicroZed ボードなどのボードの場合、使用可能なテンプレートに [Matrix Multiplication and Addition (area reduced)] アプリケーションがリストされます。このチュートリアルでは、ボード上でアプリケーションを実行することはありません。ZC702 の手順に従うことで、学習目標が達成できるようになっています。

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 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 sds++ 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。sds++ コンパイラでは、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	ハードウェア関数には、PS L2 キャッシュを介して DDR にキャッシュ コヒーレンシ アクセスがあります。
AFI (HPx)	ハードウェア関数には、PS メモリ コントローラーを介して DDR に非キャッシュ コヒーレンシ アクセスがあります。
GP	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。
MIG	ハードウェア関数は、MIG IP メモリ コントローラーを介して PL から DDR にアクセスします。

チュートリアルの目標

このチュートリアル (lab3) 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

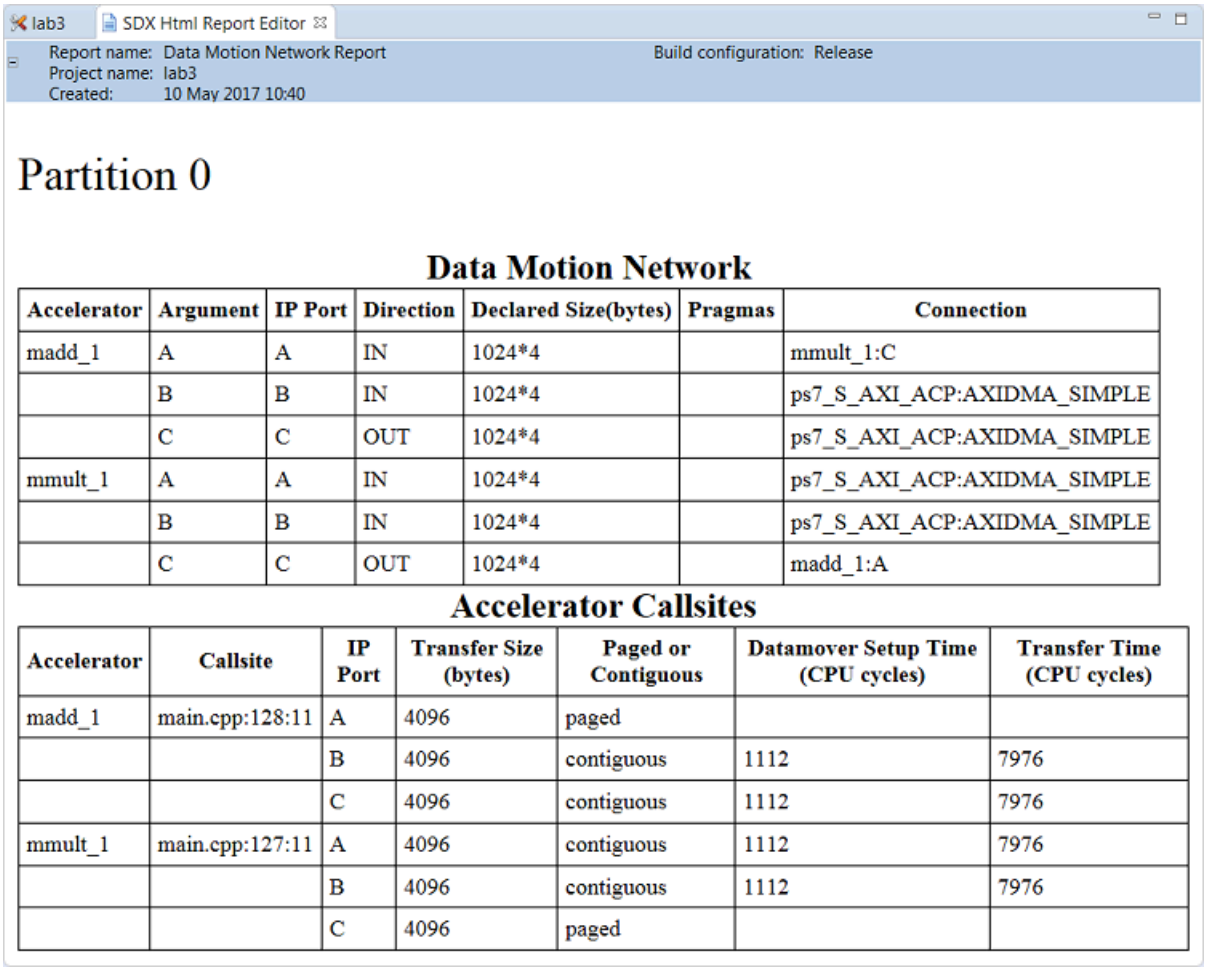
1. [Matrix Multiplication and Addition] デザイン テンプレートを使用して、[ZC702 platform] および [Linux] システム コンフィギュレーションで新しい SDx™ IDE プロジェクト (lab3) を作成します。
2. [lab3] タブをクリックして [SDx Project Settings] を開きます。タブが表示されていない場合は、[Project Explorer] タブの [lab3] プロジェクトの下の `project.sdx` ファイルをダブルクリックします。
3. [HW functions] パネルには、`madd` および `mmult` 関数がハードウェア アクセラレーション用にマークされた関数のリストに表示されます。
4. 最高のランタイム パフォーマンスにするには、[Active build configuration] オプションをクリックして [Release] を選択して [Release] コンフィギュレーションに切り替えます。または、[Build] アイコンから [Release] を選択するか、プロジェクトを右クリックして [Build Configurations] → [Set Active] → [Release] をクリックします。[Release] ビルド コンフィギュレーションでは、[Debug] 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。

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. SDx 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. [lab3] タブをクリックして [SDx Project Settings] を開きます。
   b. [Generate bitstream] 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。
3. ビルドが終了したら、[Reports] パネルで [Data Motion Network Report] をクリックして、データ モーション ネットワーク レポートを確認します。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。



lab3 | SDX Html Report Editor

Report name: Data Motion Network Report
Project name: lab3
Created: 10 May 2017 10:40
Build configuration: Release

Partition 0

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_1	A	A	IN	1024*4		mmult_1:C
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
mmult_1	A	A	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		madd_1:A

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time (CPU cycles)	Transfer Time (CPU cycles)
madd_1	main.cpp:128:11	A	4096	paged		
		B	4096	contiguous	1112	7976
		C	4096	contiguous	1112	7976
mmult_1	main.cpp:127:11	A	4096	contiguous	1112	7976
		B	4096	contiguous	1112	7976
		C	4096	paged		

4. `sys_port` プラグマを追加します。
   a. [Project Explorer] タブで `src` フォルダーの下の `mmultadd.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。

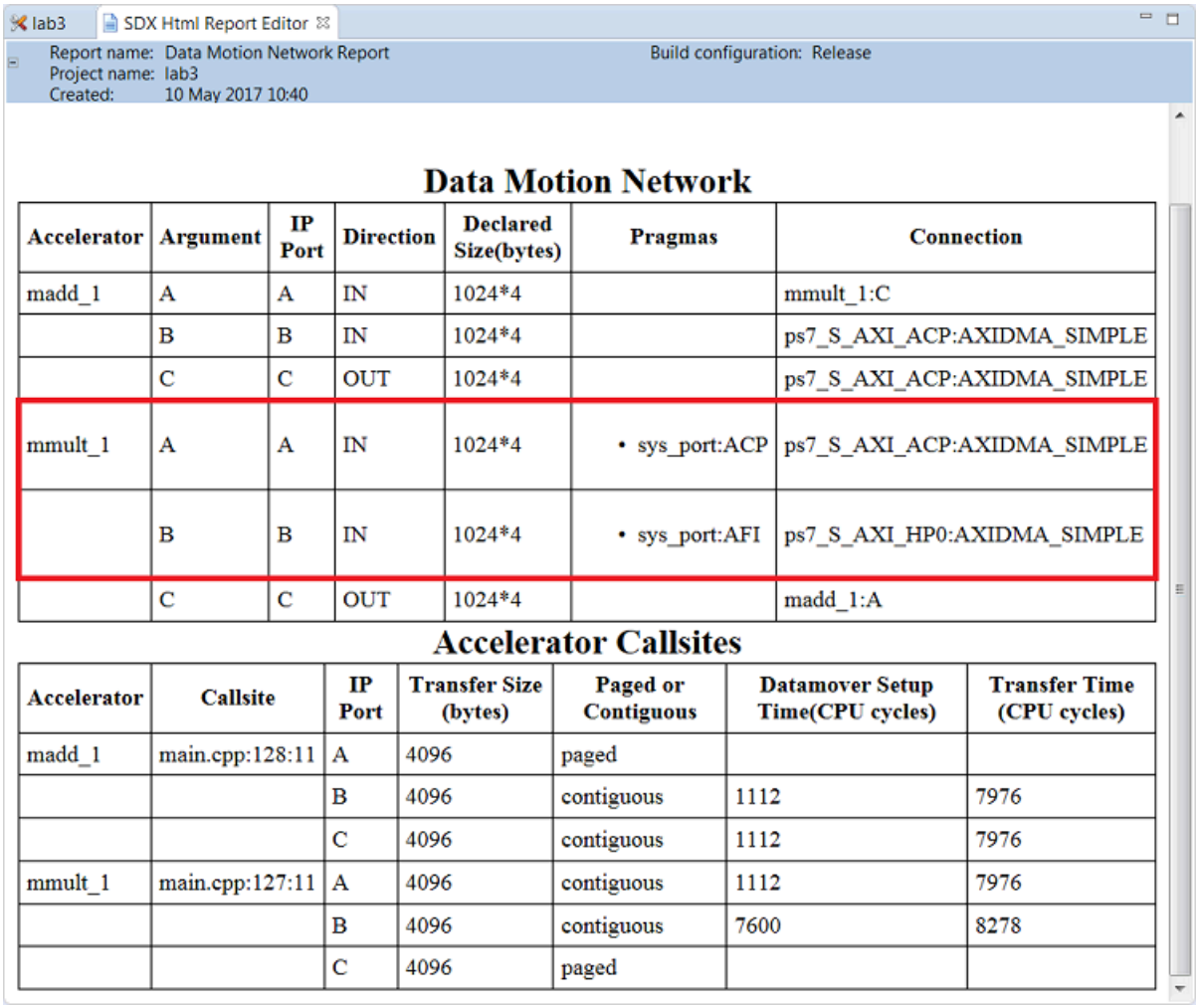
```
#pragma SDS data sys_port(A:ACP, B:AFI)
```

```
/**
 * Design principles to achieve best performance
 *
 * 1. Declare sequential access to stream data into accelerators via a hardware FIFO
 *    interface.  Otherwise, the default RAM interface requires all data to arrive
 *    before starting HLS accelerator
 */
#pragma SDS data access_pattern(A:SEQUENTIAL, B:SEQUENTIAL, C:SEQUENTIAL)
#pragma SDS data sys_port(A:ACP, B:AFI)
void mmult (float A[N*N], float B[N*N], float C[N*N]);

#pragma SDS data access_pattern(A:SEQUENTIAL, B:SEQUENTIAL, C:SEQUENTIAL)
void madd(float A[N*N], float B[N*N], float C[N*N]);

#endif /* _MMULTADD_H_ */
```

   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。
7. ビルドが終了したら、データ モーション ネットワーク レポート (`data_motion.html` ファイル) を表示しているタブをクリックします。
8. [Data Motion Network Report] ペイン内をクリックし、コンテキスト メニューから [Refresh] をクリックします。

   [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。



Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_1	A	A	IN	1024*4		mmult_1:C
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
mmult_1	A	A	IN	1024*4	• sys_port:ACP	ps7_S_AXI_ACP:AXIDMA_SIMPLE
	B	B	IN	1024*4	• sys_port:AFI	ps7_S_AXI_HP0:AXIDMA_SIMPLE
	C	C	OUT	1024*4		madd_1:A

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time (CPU cycles)
madd_1	main.cpp:128:11	A	4096	paged		
		B	4096	contiguous	1112	7976
		C	4096	contiguous	1112	7976
mmult_1	main.cpp:127:11	A	4096	contiguous	1112	7976
		B	4096	contiguous	7600	8278
		C	4096	paged		

9. プラグマ `#pragma SDS data sys_port(A:ACP, B:AFI)` を削除してファイルを保存します。

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。SDx IDE 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. `src` フォルダーからソース 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。

```
168
169     std::cout << "TEST " << (test_passed ? "FAILED" : "PASSED") << std::endl
170
171     sds_free(A);
172     sds_free(B);
```

Missing ';'
Press 'F2' for focus

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。

```
169     std::cou << "TEST " << (test_passed ? "FAILED" : "PASSED") << std::endl;
170
171     sds_f
172     sds_free(B);
```

Write occurrence of 'cou'
Press 'F2' for focus

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。

```
169     std::cout << "TEST " << (test_passed ? "FAILED" : "PASSED") << std::endl;
170
171
172     sds_free(B);
```

Press 'F2' for focus

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。

```
148 int main(int argc, char* argv[]){
149     int test_passed = 0;
150     // float *A, *B, *C, *D, *D_sw;
```

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。`Release/_sds/reports/sds_main.log` および `Release/_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。

Problems | Console | Properties | SDx Log | SDx Terminal

CDT Build Console [lab3]

```
C:/tutorials/2016.3_sdx/lab3/src/main.cpp:158:11: error: use of undeclared identifier 'A'
     if (!A || !B || !C || !D || !D_sw) {
          ^
C:/tutorials/2016.3_sdx/lab3/src/main.cpp:158:17: error: use of undeclared identifier 'B'
     if (!A || !B || !C || !D || !D_sw) {
                ^
C:/tutorials/2016.3_sdx/lab3/src/main.cpp:158:23: error: use of undeclared identifier 'C'
     if (!A || !B || !C || !D || !D_sw) {
                     ^
C:/tutorials/2016.3_sdx/lab3/src/main.cpp:158:29: error: use of undeclared identifier 'D'
     if (!A || !B || !C || !D || !D_sw) {
                          ^
C:/tutorials/2016.3_sdx/lab3/src/main.cpp:158:35: error: use of undeclared identifier 'D_sw'
     if (!A || !B || !C || !D || !D_sw) {
```

10. 変数が宣言されている行をコメントアウトします。

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動的に処理されます。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隣接するメモリを取得する必要があります。

演習 1 では、`mult_add` テンプレートを使用してシンプル DMA を使用できるようにしています。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 が自動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、lab3 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 `lab3/src` の下の `mmultadd.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(A:AXIDMA_SG, B:AXIDMA_SIMPLE, 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] の [Reports] タブで [Data Motion Report] 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。

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size (bytes)	Pragmas	Connection
madd_1	A	A	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SG
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4		ps7_S_AXI_ACP:AXIDMA_SIMPLE
mmult_1	A	A	IN	1024*4	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	B	B	IN	1024*4	• data_mover:AXIDMA_SIMPLE	ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4	• data_mover:AXIFIFO	ps7_M_AXI_GP0:AXIFIFO

6. 手順 2 で入力したプラグマ `#pragma SDS data data_mover(A:AXIDMA_SG, B:AXIDMA_SIMPLE, C:AXIFIFO)` を削除し、ファイルを保存します。

sds_alloc() を malloc() に変更

この演習では、lab3 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。

1. [Project Explorer] タブで src フォルダーの下の `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 `Release/_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。

Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
madd_1	A	A	IN	1024*4		mmult_1:C
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SG
	C	C	OUT	1024*4		ps7_S_AXI_ACP:AXIDMA_SG
mmult_1	A	A	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SG
	B	B	IN	1024*4		ps7_S_AXI_ACP:AXIDMA_SG
	C	C	OUT	1024*4		madd_1:A

8. 手順 2 の変更をすべて取り消して、ファイルを保存します。

転送されるデータ量を制御するプラグマの追加

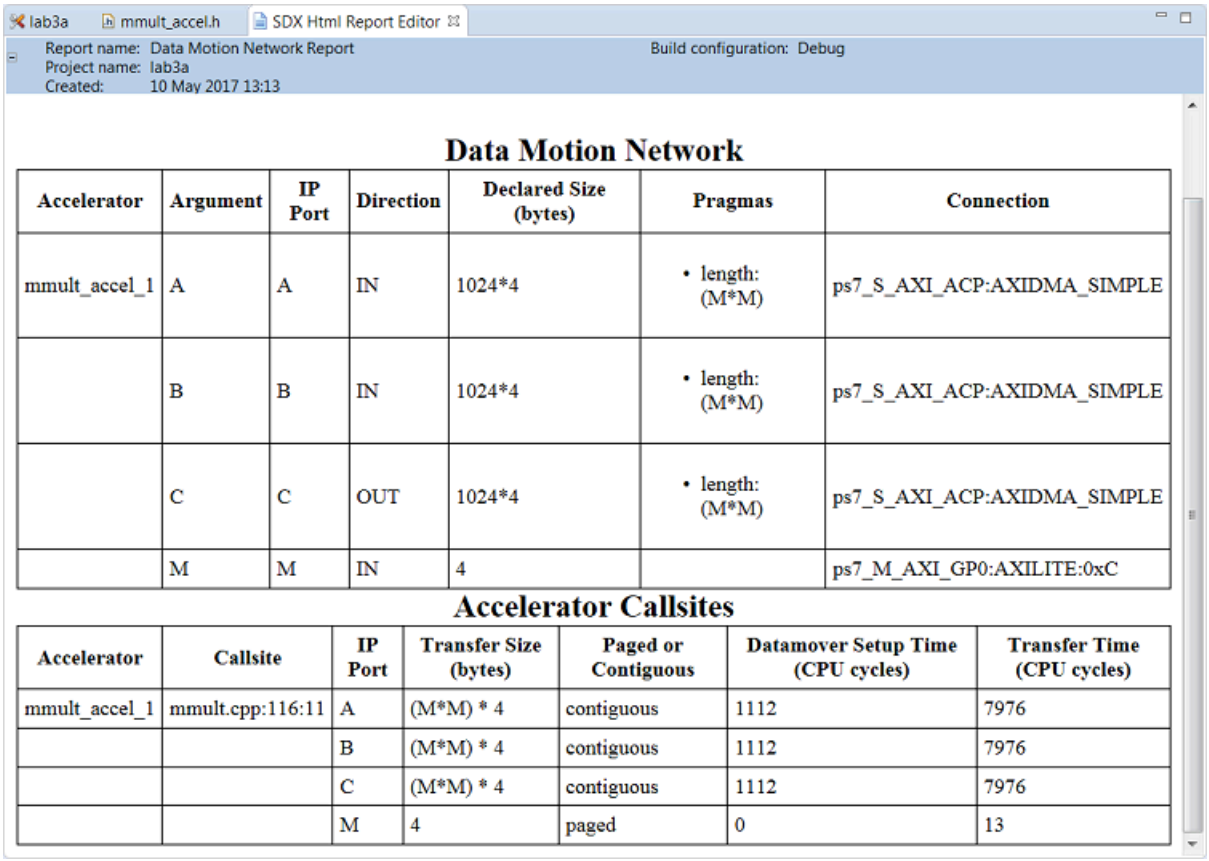
この演習では、別のテンプレートを使用してコピー プラグマを使用する方法を示します。このテンプレートでは、M と呼ばれる追加のパラメーターが行列乗算関数に渡されます。このパラメーターを使用すると、最大 32*32 までの任意サイズ M*M の正方行列 2 つを乗算する行列乗算関数を使用できます。この行列の最上位の割り当てにより、最大 32x32 までのサイズの行列が作成されます。M パラメーターは、行列乗算関数で乗算する行列のサイズを指定します。data copy プラグマは、最大の行列サイズではなく、実際の行列サイズに相当するデータ量を転送することを指定します。

1. SDx 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] → [Xilinx SDx 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 lab3a) を入力します。
   c. [zc702] および [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. mmult_accel ファンクションはハードウェア アクセラレーション用にマークされていることに注意してください。
2. [Options] パネルで [Generate bitstream] 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プロジェクトを設定します。
3. data copy プラグマがコードに含まれています。data copy プラグマを確認するには、[Project Explorer] タブで mmult_accel.h (src フォルダーの下) をダブルクリックしてソース エディター ビューで開きます。

   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 M を使用しています。

```
#pragma SDS data copy(A[0:M*M], B[0:M*M], C[0:M*M])
#pragma SDS data access_pattern(A:SEQUENTIAL, B:SEQUENTIAL, C:SEQUENTIAL)
void mmult_accel (float A[N*N],
                  float B[N*N],
                  float C[N*N],
                  int M);
```

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。
5. ビルドが完了したら、[Project Explorer] の [Reports] タブで [Data Motion Network Report] をダブルクリックして開きます。
6. 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。



Data Motion Network

Accelerator	Argument	IP Port	Direction	Declared Size (bytes)	Pragmas	Connection
mmult_accel_1	A	A	IN	1024*4	• length: (M*M)	ps7_S_AXI_ACP:AXIDMA_SIMPLE
	B	B	IN	1024*4	• length: (M*M)	ps7_S_AXI_ACP:AXIDMA_SIMPLE
	C	C	OUT	1024*4	• length: (M*M)	ps7_S_AXI_ACP:AXIDMA_SIMPLE
	M	M	IN	4		ps7_M_AXI_GP0:AXILITE:0xC

Accelerator Callsites

Accelerator	Callsite	IP Port	Transfer Size (bytes)	Paged or Contiguous	Datamover Setup Time (CPU cycles)	Transfer Time (CPU cycles)
mmult_accel_1	mmult.cpp:116:11	A	(M*M) * 4	contiguous	1112	7976
		B	(M*M) * 4	contiguous	1112	7976
		C	(M*M) * 4	contiguous	1112	7976
		M	4	paged	0	13

XILINX
ALL PROGRAMMABLE™

第 5 章

アクセラレータの最適化

演習 4: 指示子を使用したアクセラレータの最適化

この演習では、プロジェクト内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。詳細は、『SDSoC 環境最適化ガイド』(UG1235)を参照してください。

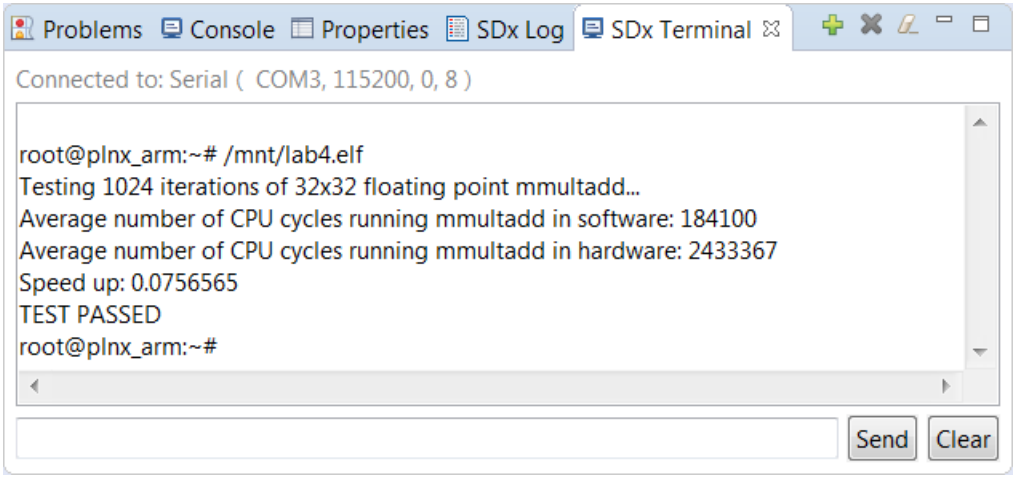
1. [Matrix Multiplication and Addition] デザイン テンプレートを使用して、[ZC702 Platform] および [Linux] システム コンフィギュレーションで新しい SDx™ 環境プロジェクト (lab4) を作成します。
2. [lab4] タブをクリックして [SDx Project Settings] を開きます。タブが表示されていない場合は、[Project Explorer] タブの [lab4] プロジェクトの下の `project.sdx` ファイルをダブルクリックします。
3. [HW functions] パネルには、`madd` および `mmult` 関数がハードウェア アクセラレーション用にマークされた関数のリストに表示されます。
4. 最高のランタイム パフォーマンスにするには、[Active Build Configuration] オプションをクリックして [Release] を選択して [Release] コンフィギュレーションに切り替えます。または、[Build] アイコンから [Release] を選択するか、プロジェクトを右クリックして [Build Configuration] → [Set Active] → [Release] をクリックします。[Release] ビルド コンフィギュレーションでは、[Debug] 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。
5. [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。
6. `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。

7. これらのプラグマの行をコメントアウトします。

```
53 void mmult (float A[N*N], float B[N*N], float C[N*N])
54 {
55      float Abuf[N][N], Bbuf[N][N];
56 // #pragma HLS array_partition variable=Abuf block factor=16 dim=2
57 // #pragma HLS array_partition variable=Bbuf block factor=16 dim=1
58
59      for(int i=0; i<N; i++) {
60           for(int j=0; j<N; j++) {
61 // #pragma HLS PIPELINE
62                Abuf[i][j] = A[i * N + j];
63                Bbuf[i][j] = B[i * N + j];
64           }
65      }
66
67      for (int i = 0; i < N; i++) {
68           for (int j = 0; j < N; j++) {
69 // #pragma HLS PIPELINE
70                float result = 0;
71                for (int k = 0; k < N; k++) {
72                     float term = Abuf[i][k] * Bbuf[k][j];
73                     result += term;
74                }
75                C[i * N + j] = result;
76           }
77      }
78 }
```

8. ファイルを保存します。
9. 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。
10. ビルドが終了したら、`lab4/Release/sd_card` を SD カードにコピーします。
11. SD カードを ZC702 ボードに挿入し、ボードに電源を投入します。
12. SDx IDE 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。+ アイコンをクリックして設定を開きます。
13. 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/lab4.elf`」と入力します。

    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 (lab1 の結果と比較すると)、`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。



演習 5: タスク レベルのパイプライン

この演習では、SDx IDE 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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDSoC 環境プロジェクトを作成する際に、ボードを選択してください。このチュートリアルでは、ZC702 のアプリケーション用に作成されたソース ファイルを追加するように記述されています。ボードにより小型の Zynq-7000 デバイスが含まれる場合は、ソース ファイルを追加した後に `mmult_accel.cpp` ファイルを編集して、リソース使用率を減らしてください。たとえば、アクセラレータ ソース ファイルの場合は、`#pragma_HLS_array_partition` で `block factor=16` と設定されていますが、`block factor=8` と設定します。

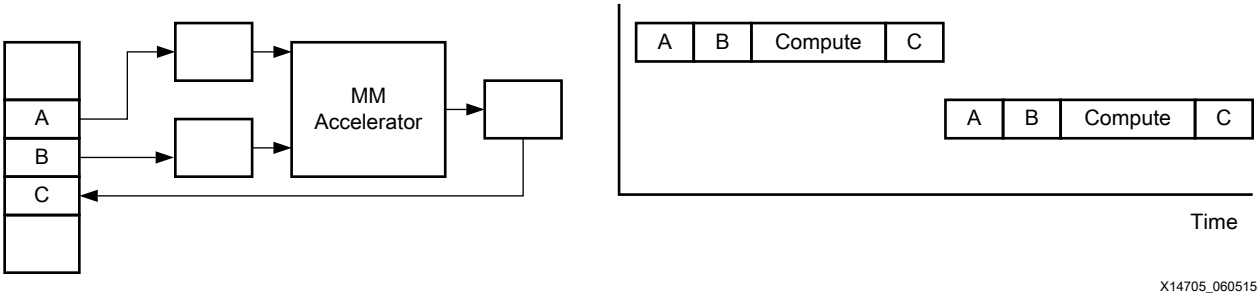
タスクのパイプライン処理

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。

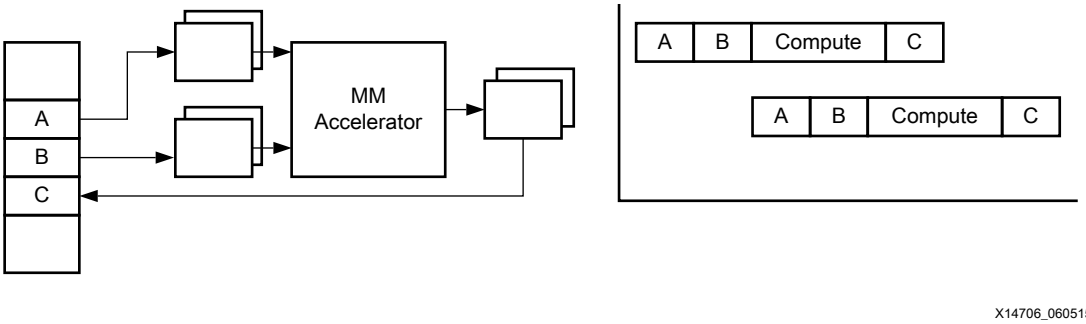
次の図の左側に行列乗算デザイン、右側に順次実行される連続した 2 つの呼び出しに対するこれらのイベントのタイムチャートを示します。

図 2: 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 3: 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。

行列乗算サンプルでのタスクのパイプライン処理

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。

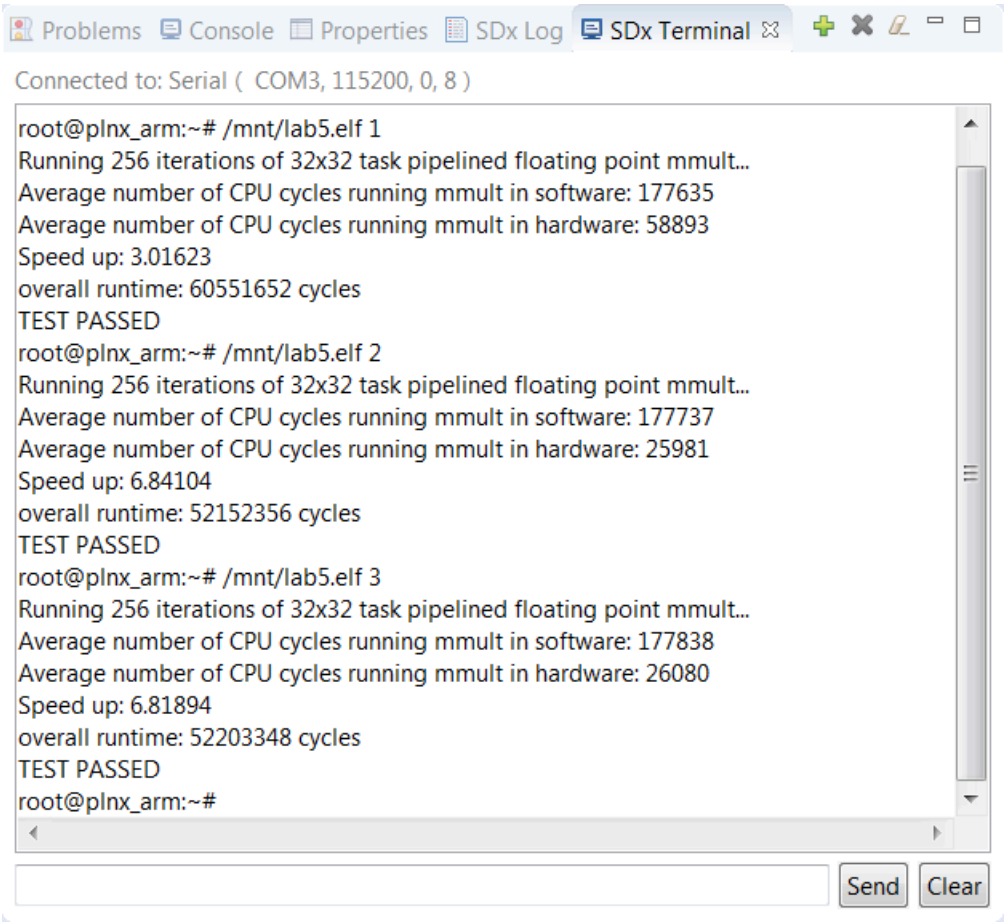
- SDx IDE 環境を使用してタスク レベルのパイプライン処理を実行することにより、アプリケーションを実行時間を短縮するよう最適化
- アクセラレータでの計算と入力および出力通信を同時に実行するアクセラレータへのパイプライン呼び出しのパフォーマンスに対する影響を確認

行列乗算サンプルでのタスクのパイプライン処理

SDx IDE 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] → [Xilinx SDx Project] をクリックし、新しい SDx プロジェクト (lab5) を作成します。[Project name] に「`lab5`」と入力し、[Platform] に [zc702]、[System Configuration] 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>/SDx/2017.0/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 (例: `./lab5/src`) にコピーします。
4. SDx で [lab5] をクリックして、コンテキスト メニューから [Refresh] をクリックします。これで前の段階でコピーしたすべてのソースがプロジェクトに追加されます。
5. ビルド コンフィギュレーションを [Release] に変更します。
6. `mmult_accel.cpp` ファイルの `mmult_accel` 関数を、[SDx Project Settings] で [Add HW Functions] 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。
7. プロジェクトをビルドします。

8. 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。



XILINX
ALL PROGRAMMABLE

第 6 章

デバッグ

演習 6: デバッグ

このチュートリアルでは、SDx IDE でインタラクティブ 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、SDx IDE 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。

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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDx プロジェクトを作成する際に、推奨されている [Matrix Multiplication and Addition] テンプレートが見つからない場合は、ボードと使用可能なアプリケーションの 1 つを選択してください。たとえば、より小型の Zynq-7000 デバイスを含む MicroZed ボードなどのボードの場合、使用可能なテンプレートに [Matrix Multiplication and Addition (area reduced)] アプリケーションがリストされます。このチュートリアルの学習目標は、どのアプリケーションでも達成できるようになっています。

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。

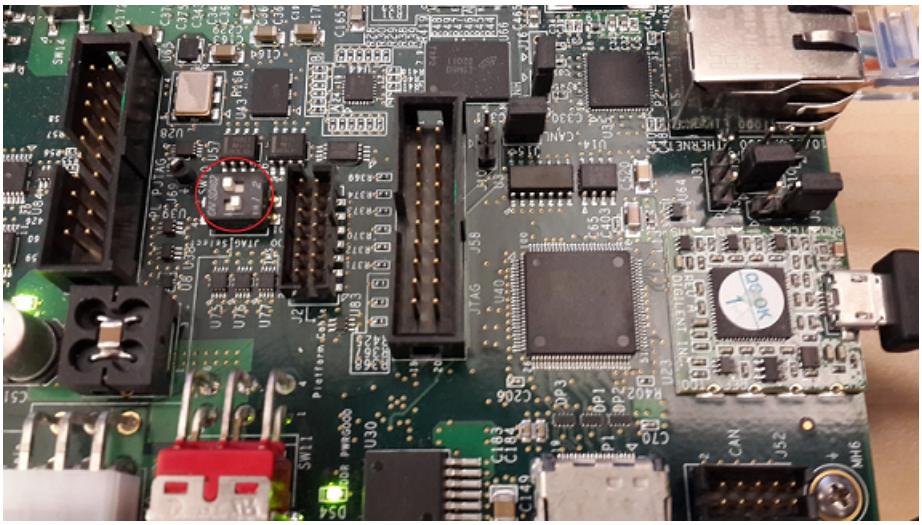
- SDx IDE を使用して、スタンドアロン アプリケーションをダウンロードして実行
- オプションで SDx IDE でソース コードをステップ スルーして、さまざまなレジスタおよびメモリを確認(ARM A9 で実行するコードに制限され、ハードウェア関数に変換されたコードには適用されない)

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 SDx IDE 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。
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. DIP スイッチ (上の図の赤丸) 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。
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、SDx IDE 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 SDx™ プロジェクト (lab6) を作成します。

SDx IDE 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. SDx IDE が起動します。
2. [File] → [New] → [Xilinx SDx Project] をクリックします。
3. [Project name] フィールドにプロジェクト名を指定します (例: `lab6`)。[Next] をクリックします。
4. [Platform] から [zc702] を選択します。[Next] をクリックします。
5. [System Configuration] ドロップダウン リストから [Standalone] を選択します。[Next] をクリックします。
6. [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。
7. [lab6] タブをクリックし (タブが表示されていない場合は [Project Explorer] ビューで `project.sdx` ファイルをダブルクリック)、[SDx Project Settings] の [HW functions] パネルで `mmult` および `madd` 関数がハードウェア関数としてマークされていることを確認します。
8. ハードウェア関数としてマークされていない場合は、[Add HW Functions] ボタン をクリックしてダイアログ ボックスを開き、ハードウェア関数としてマークします。[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 [HW functions] セクションに追加します。

9. [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。

   SDx 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。

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] ビューで lab6 プロジェクトの Debug フォルダーに含まれる ELF (`.elf`) ファイルをクリックし、ツールバーの [Debug] ボタンをクリックするか、[Debug] ボタンのプルダウン メニューから [Debug As] → [Launch on Hardware (SDSoC Debugger)] 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、SDx IDE 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 SDx ログが表示されます。

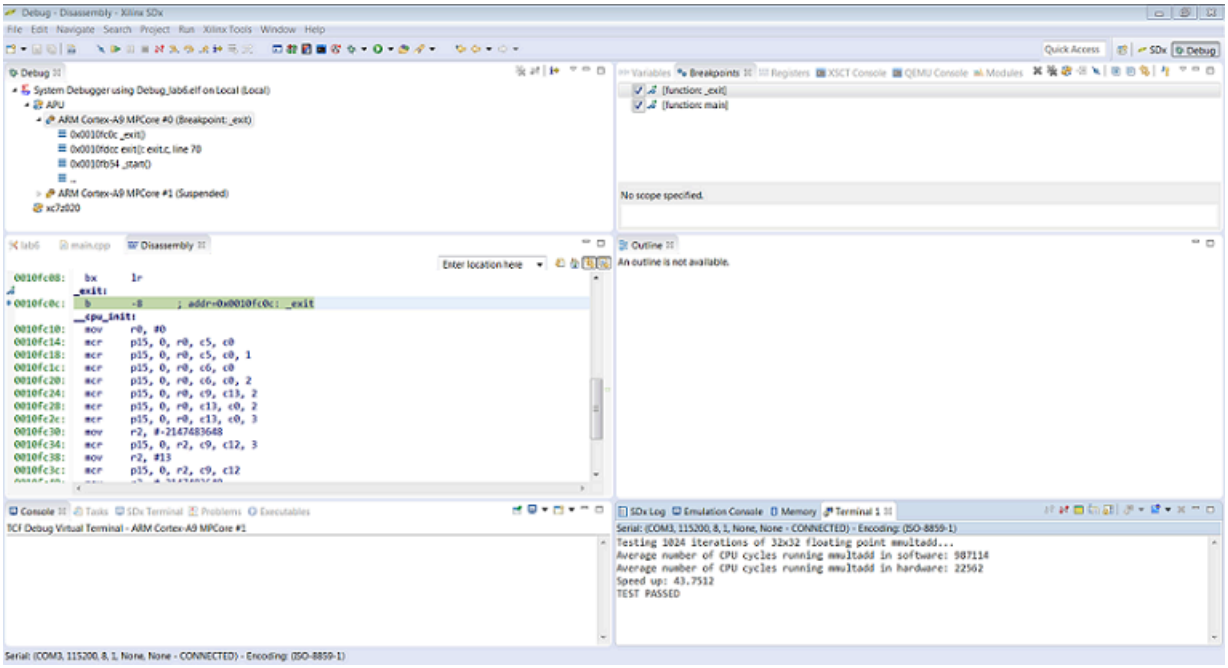
3. アプリケーションを実行する前に、シリアル ターミナルをボードに接続して、プログラムからの出力が表示されるようにする必要があります。この例では、[Window] → [Show View] → [Other] をクリックし、[Terminal] → [Terminal] をクリックして起動した SDSoC 環境ターミナルを使用します。[Debug] パースペクティブの下部の [Terminal] タブをクリックし、[Connect] アイコン をクリックして、ターミナルをボード (既に電源投入済みのはず) に接続します。[Connection Type]: Serial、[Port]: COM<n>、[Baud Rate]: 115200 ボーに設定します。

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、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. lab6 の [Debug] ビュー (`Debug_lab6.elf` を使用した System Debugger) でデバッグ階層を右クリックし、[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。

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. プロジェクトを作成し (例: lab6_linux)、[Platform] に [zc702] を、[System Configuration] に [Linux] を指定します。アプリケーション テンプレートのリストから [Matrix Multiplication and Addition] を選択します。

   詳細は、新規プロジェクトの作成を参照してください。

2. [SDx Project Settings] の [HW functions] の表で、`mmult` および `madd` 関数がハードウェア インプリメンテーション用にマークされます。

   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。アクティブ ビルド コンフィギュレーションに [Debug] を使用します

   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。(SDSoC インストールでのディスク使用量を最小限にするため、インポートされるプロジェクトに含まれるファイル数がユーザーのビルドしたプロジェクトよりも少ないことがありますが、チュートリアルを終了するのに必要なファイルは含まれています)。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] をオンにしてプロジェクト ファイル フォルダーの `lab6_linux.zip` ファイル (`<path to install>/SDx/2017.1/docs/labs/lab6_linux.zip`) を指定します。[Open] 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 lab6_linux プロジェクトを展開表示し、`project.sdx` をダブルクリックして [SDx Project Settings] を表示します。[Options] パネルで [Generate bitstream] をオフにして、[Generate SD card image] はオンにままにします。lab6_linux 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、lab6_linux 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。[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。

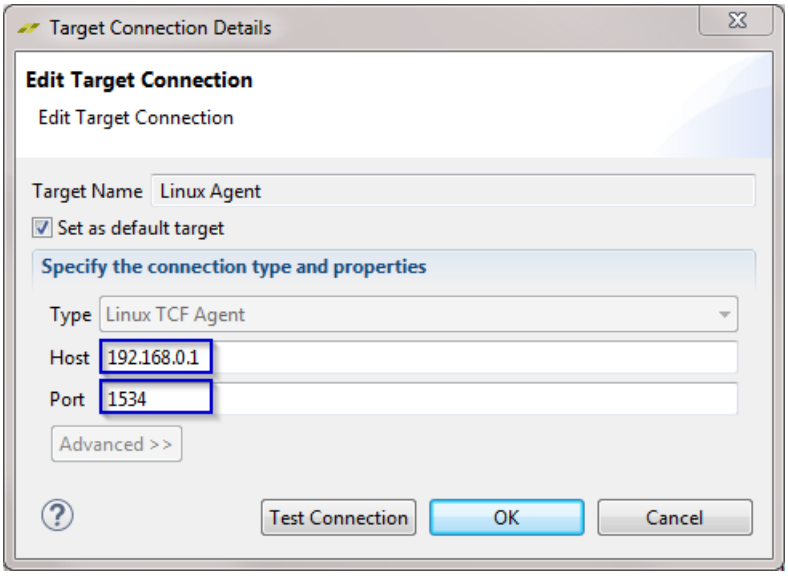
   - SD カードを挿入せずにボードに電源を投入します。
   - [Terminal] タブの [Settings] アイコン をクリックし、コンフィギュレーションを設定し、[OK] をクリックします。
   - ターミナルに接続されていることが示されます。赤い [Disconnect] アイコン をクリックしてボードからターミナルの接続を解除して、ボードの電源を切ります。

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コンピュ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。ターミナル プロンプトが表示されたら、`ifconfig eth0 192.168.0.1` と入力して IP アドレスを設定します。コンピューターは、イーサネット アダプターが ZC702 ボードと同じサブ ネットワークにあるように設定する必要があります。Windows ホスト システムで [コントロール パネル] → [ネットワークとインターネット] → [ネットワ

ークtと共有センター] → [アダプター設定の変更] をクリックし、イーサネット アダプターの [ローカル エリア接続] を右クリックして [プロパティ] を選択します。[ネットワーク] タブで [インターネット プロトコル バージョン 4 (TCP/IPv4)] を選択し、[プロパティ] ボタンをクリックします。[全般] タブで [次の IP アドレスを使う] をオンにして「`192.168.0.11`」と入力します。[OK] をクリックします。

サブネットワークに既に `192.168.0.11` のデバイスが含まれる場合は、別のアドレスを選択してください。`192.168.0.x` で始まるアドレスを使用できます。

7. SDSoC 環境に戻り、[Target Connections]パネルで [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. [Project Explorer] ビューで ELF ファイルをクリックして選択し、ツールバーの [Debug] アイコンをクリック (または [Debug] アイコンのプルダウン メニューから [Debug As] → [Launch on Hardware (SDSoC Debugger)] 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またはステップ実行します。

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。

演習 7: ハードウェア デバッグ

この演習では、プロジェクトを作成し、トレースをオンにし、アプリケーションを実行し、トレース結果を表示して確認する方法を手順を追って説明します。このチュートリアルでは、ホスト PC が直接 Zynq-7000 ボードに接続されており、ボードはザイリンクス ZC702 ボードであると想定しています。このチュートリアルは、ほかのボードおよびコンフィギュレーションにも適用できますが、詳細な手順は少し違う可能性もあります。このチュートリアルでは、すでに SDx IDE をインストールして GUI を起動してワークスペースを選択したものと仮定して説明します。

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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDx プロジェクトを作成する際に、推奨されている [Matrix Multiplication] テンプレートが見つからない場合は、ボードと使用可能なテンプレートの 1 つを選択してください。たとえば、より小型の Zynq-7000 デバイスを含む MicroZed ボードなどのボードの場合、使用可能なテンプレートに [Matrix Multiplication (area reduced)] アプリケーションがリストされます。このチュートリアルの学習目標は、どのアプリケーションでも達成できるようになっています。

スタンドアロンまたはベアメタル プロジェクトのトレース

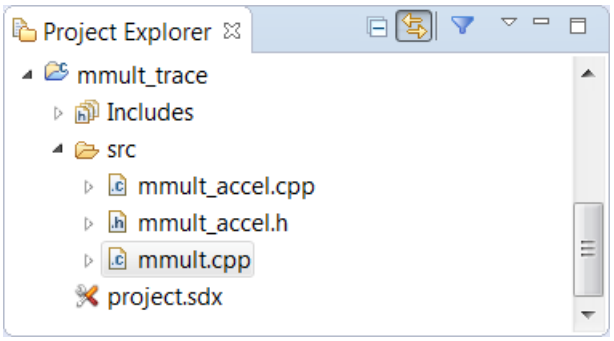
新規プロジェクトの作成方法、SDSoC トレース機能を有効にするようにプロジェクトを設定する方法、プロジェクトのビルド方法、ボード上でのアプリケーションの実行方法などを学ぶことができます。

新規プロジェクトの作成

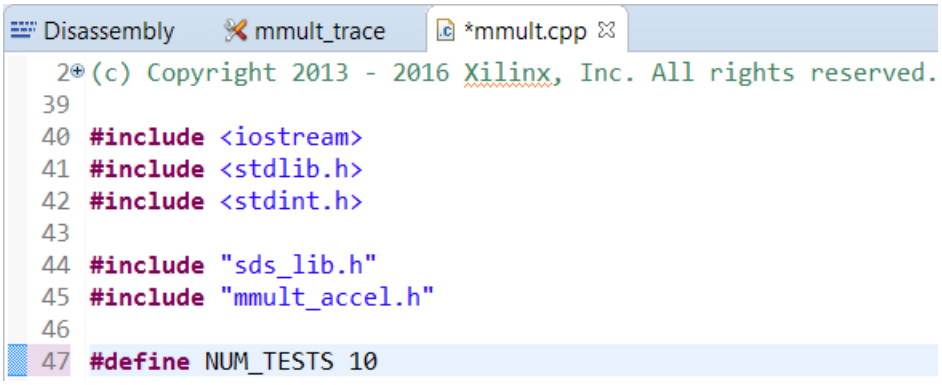
1. [File] → [New] → [Xilinx SDx Project] をクリックします。
2. New Project ウィザードで、[Project name] に `mmult_trace` と入力し、[Next] をクリックします。
3. [Choose Hardware Platform] ページで [zc702] を選択し、[Next] をクリックします。

   注記: ZC702 ボード以外を使用している場合は、適切なプラットフォームを選択します。

4. [System Configuration] に [Standalone OS] を選択します。
5. [Available Templates] で [Matrix Multiplication Data Size] を選択し、[Finish] をクリックします。
6. [Project Explorer] ビューで三角形のアイコン ▷ をクリックしてさまざまなフォルダーを展開表示し、`mmult.cpp` ファイルを開きます。

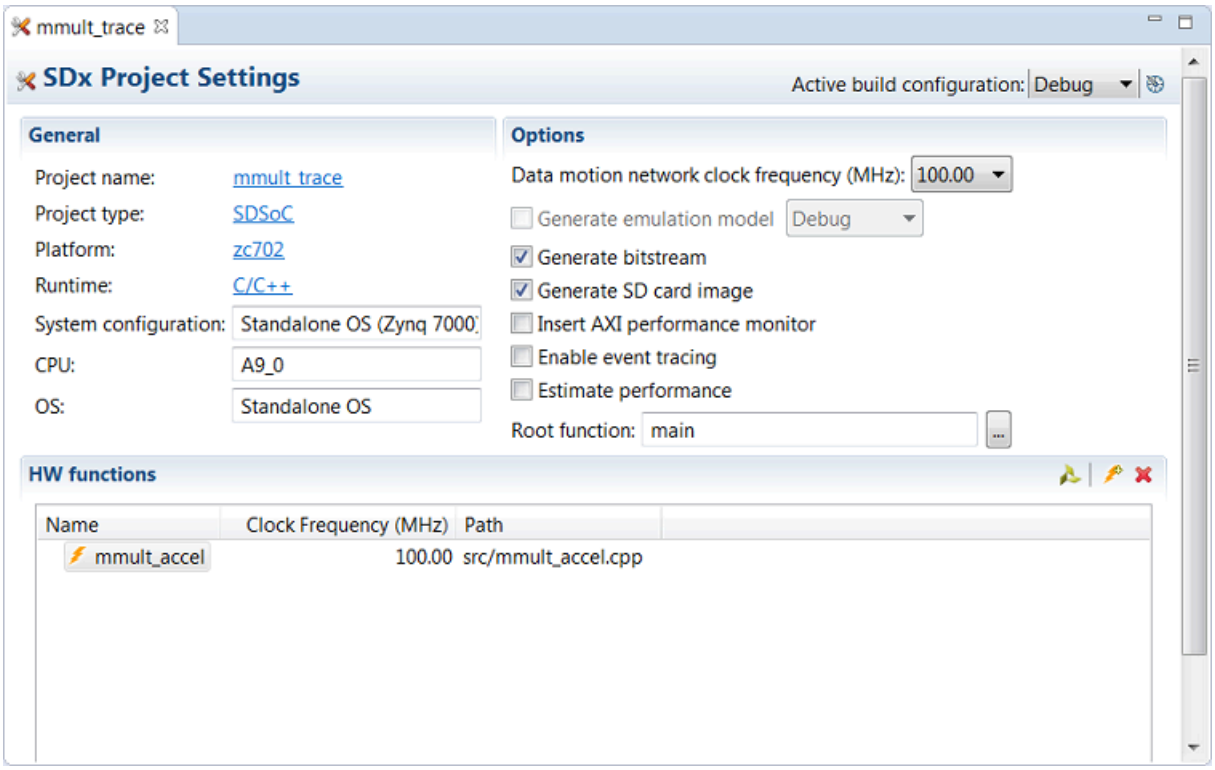


7. テスト シンボル NUM_TESTS の数値を 256 から 10 に変更し、ファイルを保存して閉じます。



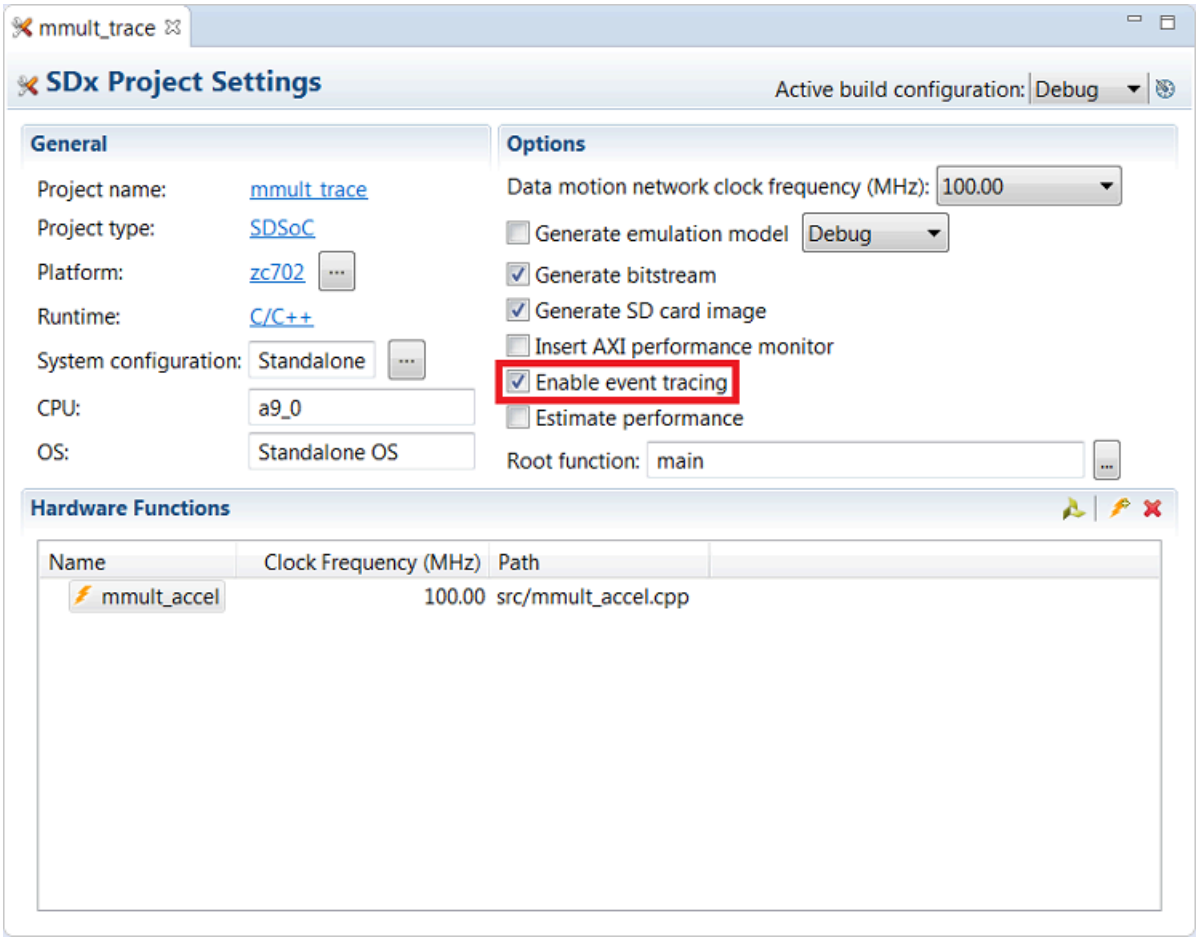
8. [SDx Project Settings] ([mmult_trace] タブ) の [HW functions] セクションで mmult_accel がハードウ

ェアにインプリメントされるようマークされていることを確認します。



[Options] セクションでトレース機能をオンにしてプロジェクト設定を変更

1. [Project Settings] ウィンドウで [Enable event tracing] をオンにします。

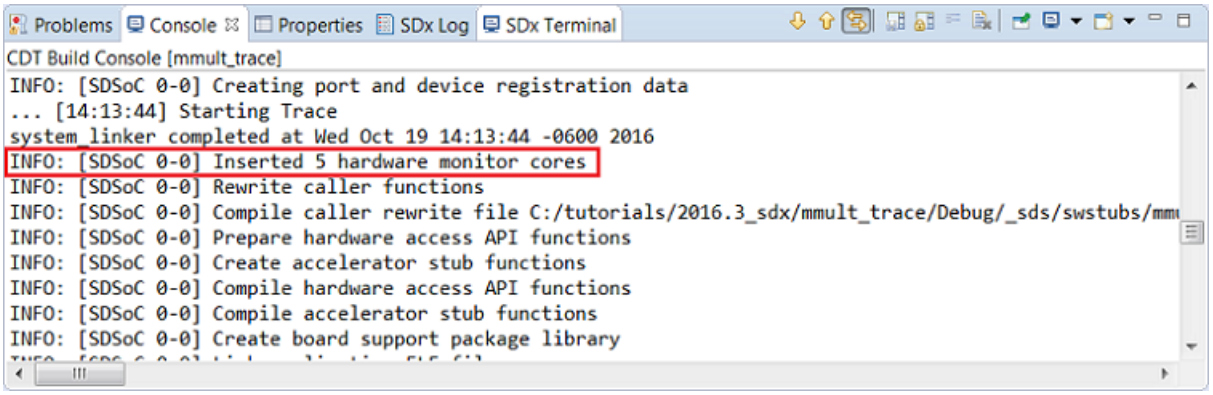


プロジェクトのビルド

1. [Build] ボタンをクリックし、プロジェクトをビルドします。これには多少時間がかかります。

重要: ビルド プロセスが完了するまでに、約 15 ～ 20 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 (SDx インストールでのディスク使用量を最小限にするため、インポートされるプロジェクトに含まれるファイル数がユーザーのビルドしたプロジェクトよりも少ないことがありますが、チュートリアルを終了するのに必要なファイルは含まれています)。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] をオンにしてプロジェクト ファイル フォルダーの `lab7a_mmult_trace.zip` ファイル (`<path to install>/SDx/2017.1/docs/labs/lab7a_mmult_trace.zip`) を指定します。[Open] をクリックします。[Finish] をクリックします。[Project Explorer] で `project.sdx` ファイルをダブルクリックして、[SDx Project Settings] を開きます。[Generate bitstream] をオフにします。

Vivado HLS ですべてのハードウェア関数がインプリメントされ、Vivado IP インテグレーター デザインが作成されると、コンソールに「`Inserted # hardware monitor cores`」というメッセージが表示されます。このメッセージは、デザインでトレース機能がイネーブルになったことと、自動挿入されたハードウェア モニター コアの数を示します。



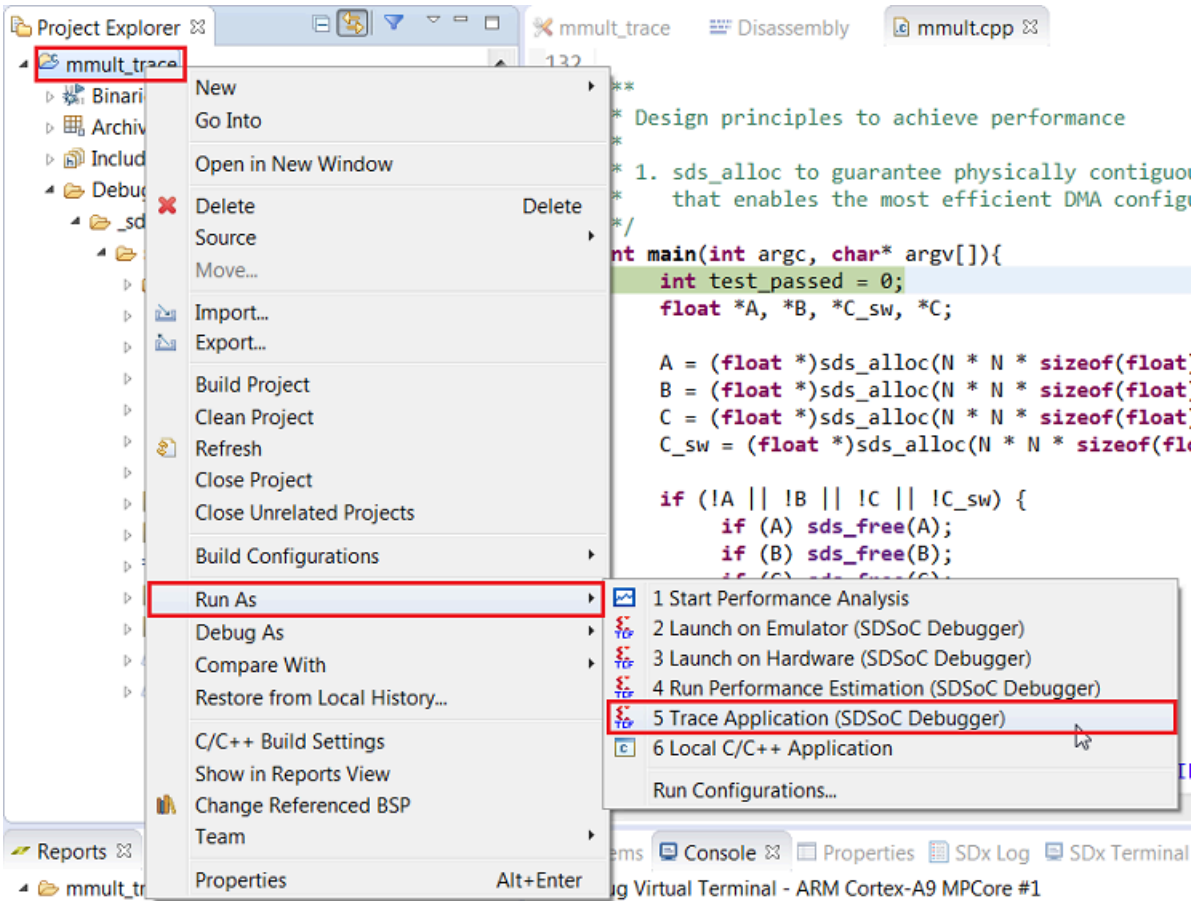
ボードでのアプリケーションの実行

1. ビルドが終了したら、[Project Explorer] ビューでプロジェクトを右クリックし、[Run As] → [Trace Application (SDSoC Debugger)] をクリックします。

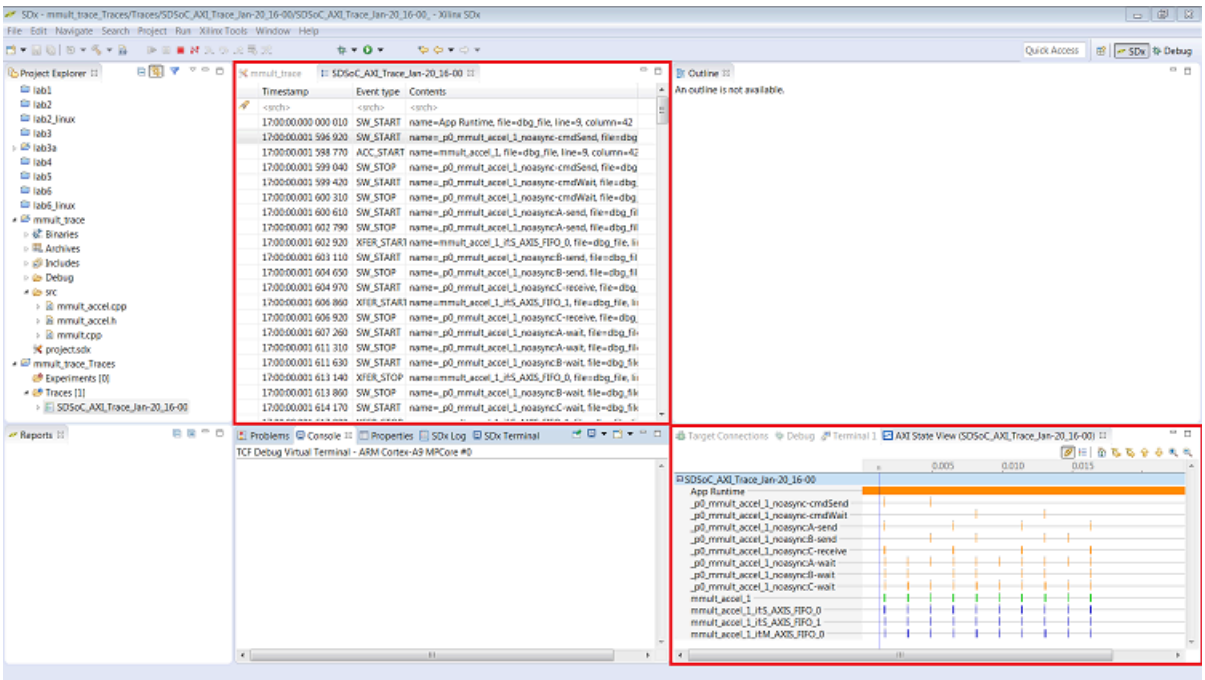
   注記: [Debug As] をオンにするとブレークポイントが有効になるので、オンにしないようにしてください。プログラムの実行がブレークポイントで停止すると、ソフトウェアは停止しますが、ハードウェアとタイムスタンプに使用されるトレース タイマーは実行し続けるので、タイミングが正確なものになりません。

   [Trace Application (SDSoC Debugger)] をクリックすると、ボードにビットストリームおよびアプリケーション ELF がダウンロードされ、アプリケーションが開始されて、トレース データの収集が開始されてアプリケーションが終了するまで収集されます。アプリケーションが終了するかトレース データの収集中にエラーが発生すると、収集されたデータが表示されます。

   注記: トレース データが正しく収集されるためには、アプリケーションが正常に終了する必要があります。ハードウェアまたはソフトウェアが停止したり、Linux カーネルがクラッシュするなど、アプリケーションが正常に終了しなかった場合、トレース データが正しく収集されない可能性があります。

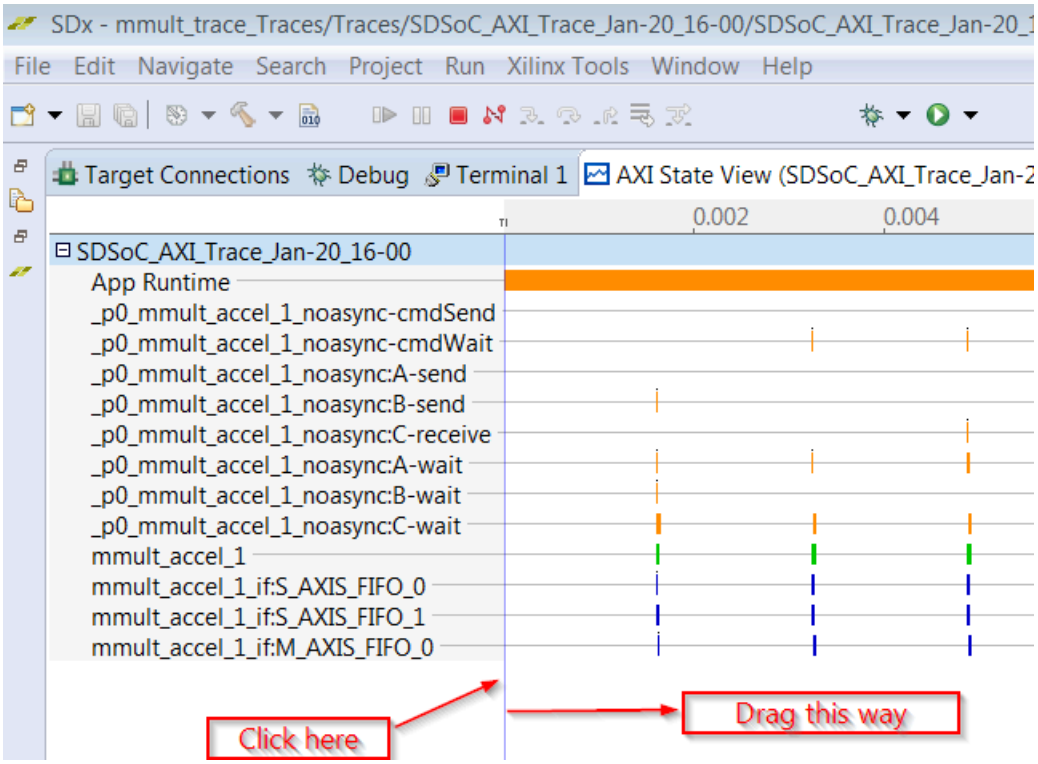


2. アプリケーションが終了すると、すべてのトレース データが収集され、表示されます。イベントのテキスト リスト (下の図で黄色で囲まれたエリア) とイベント タイムライン (下の図で紫色で囲まれたエリア) の 2 つのエリアがあります。これらのエリアは、同じ情報を示します。テキスト リストには、イベントが時間の降順に表示されます。イベント タイムラインは、デザインの各トレース ポイント (トレースされたモニター コアまたはソフトウェアの領域) が複数の軸で表示されます。

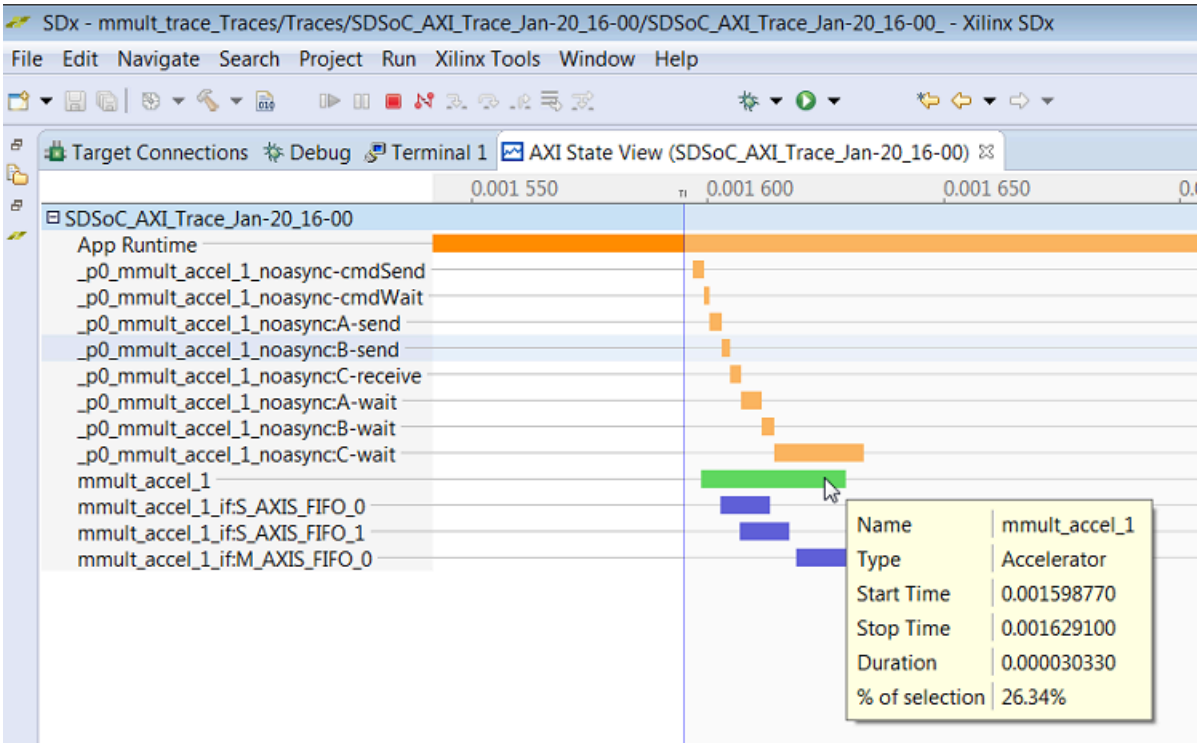


アプリケーションの 10 回の反復が繰り返されたイベント グループとして示されているのがわかります。オレンジのイベントはソフトウェア イベント、緑のイベントはアクセラレータ イベント、青のイベントはデータ転送イベントです。

3. イベント タイムラインのトレース ポイントの名前に省略記号 (...) が付いている場合、名前が表示されているグレーのエリアとタイムラインが表示されている白のエリアの境界 (適切な位置にカーソルを置くと境界線が赤色に変る) をクリックして右にドラッグすると、名前のエリアを広げることができます。



4. イベントのいずれかにカーソルを置くと、ツール ヒントに各トレースの詳細情報が表示されます。次の図に、最初のアクセラレータ イベントを示します。このイベントは、Vivado HLS でハードウェアにインプリメントするよう選択した `mmult_accel` 関数の開始/停止に対応します。開始時間は 0.000002070 秒 (2,070ns) で、停止時間は 0.000038110 秒 (38,110ns) です。イベントの期間 (この場合はアクセラレータの実行時間) は 0.000036040 秒 (36,040ns) です。

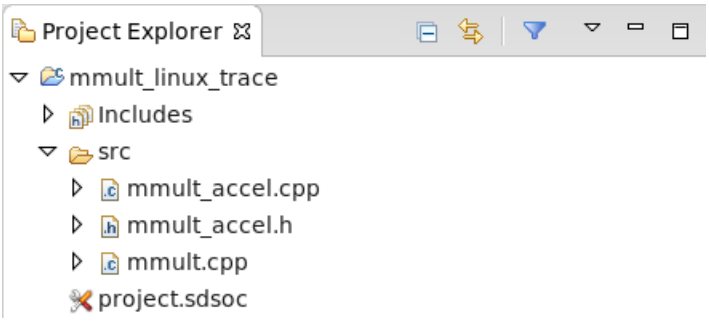


Linux プロジェクトのトレース

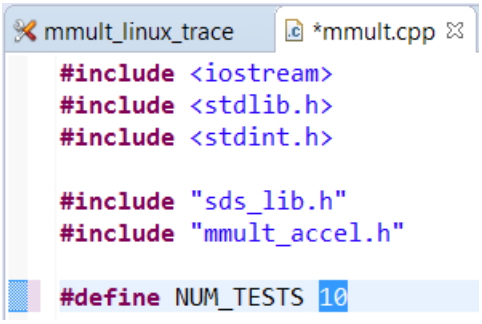
新規プロジェクトの作成方法、SDx トレース機能を有効にするようにプロジェクトを設定する方法、プロジェクトのビルド方法、ボード上でのアプリケーションの実行方法、トレース データの確認方法などを学ぶことができます。

1. 新しいプロジェクトを作成します。
    a. [File] → [New] → [Xilinx SDx Project] をクリックします。
    b. New Project ウィザードで、[Project name] に `mmult_linux_trace` と入力し、[Next] をクリックします。
    c. ハードウェア プラットフォームに [zc702] を選択します。[Next] をクリックします。
    d. [System Configuration] に [Linux] をクリックします。
    e. [Next] をクリックします。
    f. [Available Templates] で [Matrix Multiplication Data Size] を選択し、[Finish] をクリックします。

g. [Project Explorer] ビューで三角形のアイコン ▷ をクリックしてさまざまなフォルダーを展開表示し、src フォルダーの下の mmult.cpp ファイルを開きます。



h. テスト シンボル NUM_TESTS の数値を 256 から 10 に変更し、ファイルを保存して閉じます。

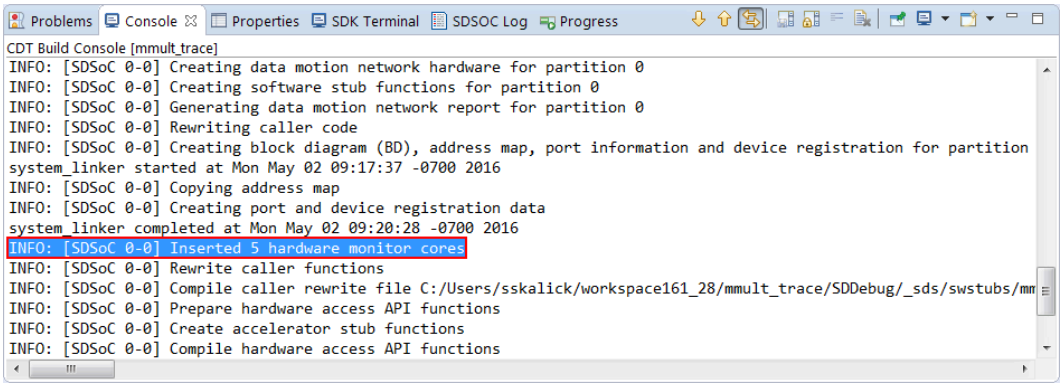


i. [SDx Project Settings] ([mmult_linux_trace] タブ) の [HW functions] セクションで mmult_accel がハードウェアにインプリメントされるようマークされていることを確認します。

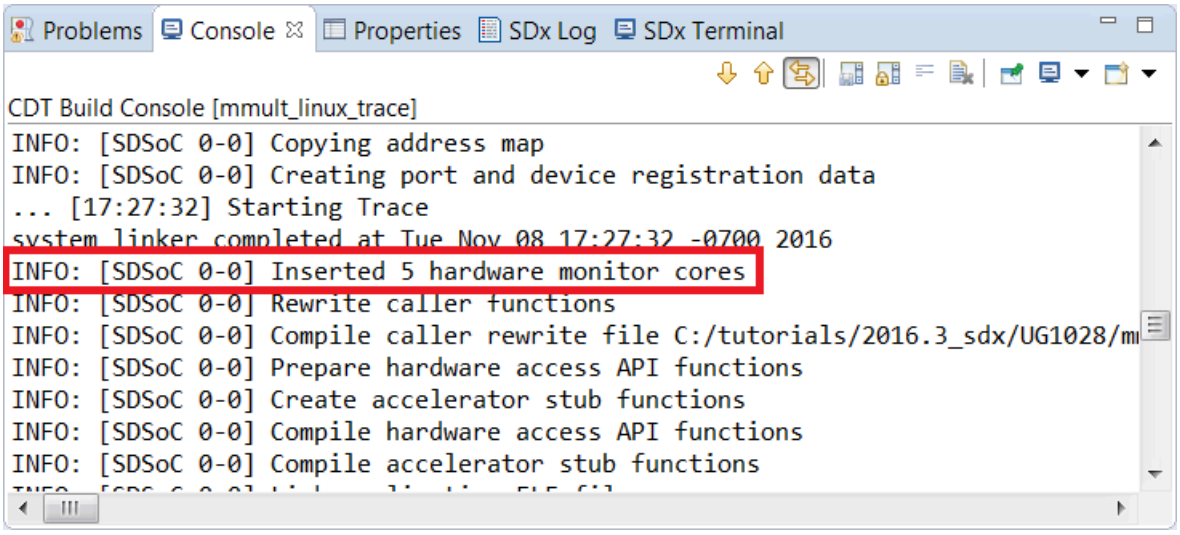
2. プロジェクト設定を変更し、SDx IDE でトレース機能をイネーブルにします。
   a. [SDx Project Settings] の [Options] セクションの [Enable event tracing] をオンにします。
3. プロジェクトをビルドします。
   a. [Build] ボタンをクリックし、プロジェクトをビルドします。これには多少時間がかかります。

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。(SDx インストールでのディスク使用量を最小限にするため、インポートされるプロジェクトに含まれるファイル数がユーザーのビルドしたプロジェクトよりも少ないことがありますが、チュートリアルを終了するのに必要なファイルは含まれています)。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 `lab7b_mmult_trace_linux.zip` ファイル (`<path to install>/SDSoC/2017.1/docs/labs/lab7b_mmult_trace_linux.zip`) を指定します。[Open] をクリックします。[Finish] をクリックします。[Project Explorer] で `project.sdx` ファイルをダブルクリックして、[SDx Project Settings] を開きます。[Generate bitstream] をオフにします。

Vivado HLS ですべてのハードウェア関数がインプリメントされ、Vivado IP インテグレーター デザインが作成されると、コンソールに「`Inserted # hardware monitor cores`」というメッセージが表示されます。このメッセージは、デザインでトレース機能がイネーブルになったことと、自動挿入されたハードウェア モニター コアの数を示します。

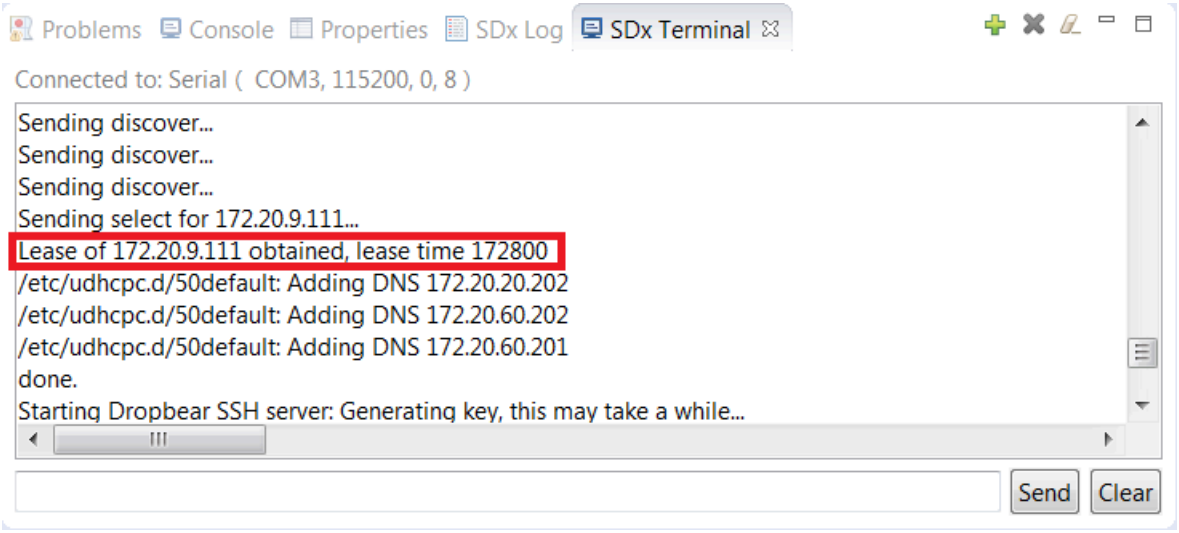


X16927-050316



4. ボード上でアプリケーションを実行します。
   a. ビルドが完了したら、sd_card ディレクトリのファイルを SD カードにコピーし、SD カードをボードの SD カード ソケットに挿入します。
   b. イーサネット ケーブルをボードに接続します (ネットワークに接続するか、PC に直接接続)。
   c. USB/UART ポートを PC に接続し、[SDx Terminal] タブで + ボタンをクリックしてシリアル コンソールを開きます。
   d. USB/JTAG ポートを PC に接続し、ボード上で Linux を起動します。

e. [SDx Terminal] ログから ZC702 ボードの IP アドレスを確認します。



f. Linux でのパフォーマンス予測フローの使用と同じように、[Target Connections] ビューから [Linux TCF Agent] を設定します。

g. [Project Explorer] ビューでプロジェクトを右クリックし、[Run As] → [Trace Application (SDSoC Debugger)] をクリックします。

注記: [Debug As] をクリックしないでください。[Debug As] をクリックすると、ブレークポイントがイネーブルになります。プログラムの実行がブレークポイントで停止すると、ソフトウェアは停止しますが、ハードウェアとタイムスタンプに使用されるトレース タイマーは実行し続けるので、タイミングが正確なものになりません。

[Trace Application (SDSoC Debugger)] をクリックすると、イーサネット TCF エージェント接続を介して ELF ファイルがダウンロードされ、アプリケーションが開始して、トレース データの収集が開始されてアプリケーションが終了するまで収集されます。アプリケーションが終了するかトレース データの収集中にエラーが発生すると、収集されたデータが表示されます。

注記: トレース データが正しく収集されるためには、アプリケーションが正常に終了する必要があります。ハードウェアまたはソフトウェアが停止したり、Linux カーネルがクラッシュするなど、アプリケーションが正常に終了しなかった場合、トレース データが正しく収集されない可能性があります。

5. トレース データを確認します。
   a. アプリケーションが終了すると、すべてのトレース データが収集され、表示されます。

トレースの表示

1. アプリケーションを実行してトレース データを収集すると、トレースのアーカイブが作成され、プロジェクトのビルド ディレクトリ `<build_config>/_sds/trace` に保存されます。
2. このトレース アーカイブを開くには、右クリックして [Import and Open AXI Trace] をクリックします。

`_sds/trace` フォルダーには、`metadata` および `sdsoc_trace.tcl` も含まれます。これらのファイルはビルド中に生成され、トレース データを抽出し、トレース可視化アーカイブを作成するために使用されます。これらのファイルを削除または変更すると、トレース データを収集できなくなり、クリーンアップおよびビルドを実行して再生成する必要があります。

XILINX
ALL PROGRAMMABLE™

第 7 章

エミュレーション

演習 8: エミュレーション

このチュートリアルでは、SDx IDE でエミュレーション機能を使用する方法を示します。

まず、デザインの OS とプラットフォームを指定し、プログラムでエミュレーションを実行します。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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDx プロジェクトを作成する際は、推奨されているエミュレーション アプリケーションのテンプレートが見つからない場合でも、ボードと使用可能なテンプレートの 1 つを選択してください。

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。

- SDx IDE を使用して、エミュレーション モードで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 SDx IDE 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

エミュレーションを実行するプロジェクトの作成

[Emulation Example] 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 SDx プロジェクト (lab8) を作成します。SDx IDE でプロジェクトを作成する手順は、次のとおりです。

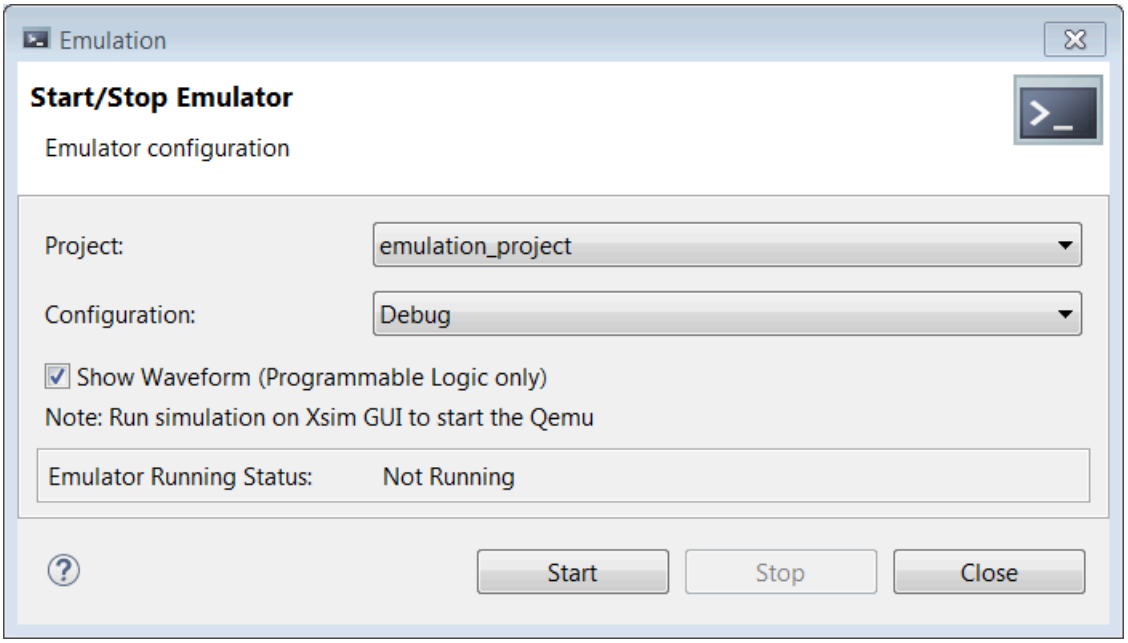
1. SDx IDE が起動します。
2. [File] → [New] → [Xilinx SDx Project] をクリックします。
3. [Project name] フィールドにプロジェクト名を指定します (例: lab8)。[Next] をクリックします。
4. [Platform] から [zc702] を選択します。[Next] をクリックします。
5. [System Configuration] ドロップダウン リストから [Linux] を選択します。[Next] をクリックします。
6. [Available Templates] から [Emulation Example] を選択し、[Finish] をクリックします。
7. [lab8] タブをクリックして SDx プロジェクト設定を選択します (タブが表示されていない場合は、[Project Explorer] で project.sdx ファイルをダブルクリックします)。[HW functions] パネルで、プロジ

ェクトが作成されたときに mmult_accel 関数がハードウェア関数としてマークされていることを確認します。

8. ハードウェア関数が削除されていたり、マークされていない場合は、[Add HW Functions] アイコンをクリックして表示されたダイアログ ボックス内でハードウェア関数を指定します。
9. [SDx Project Settings] の [Options] の下の [Generate emulation model] オプションを確認します。このプルダウン メニューには [Debug] および [Optimized] の 2 つのオプションがあります。[Debug] オプションを選択して、デバッグ情報のキャプチャを有効にします。デバッグ情報なしにしてエミュレーションを速めるには、[Optimized] プルダウン メニューの方を選択します。この演習では、デフォルト オプションの [Debug] を使用します。
10. [Generate emulation model] オプションを選択すると、ビットストリームと SD カード イメージの生成をオフにするかどうかを尋ねるダイアログ ボックスが表示されます。エミュレーション モードでは、ビットストリーム オプションおよび SD カード イメージは必要ないので、[Yes] をクリックします。

エミュレーターの開始

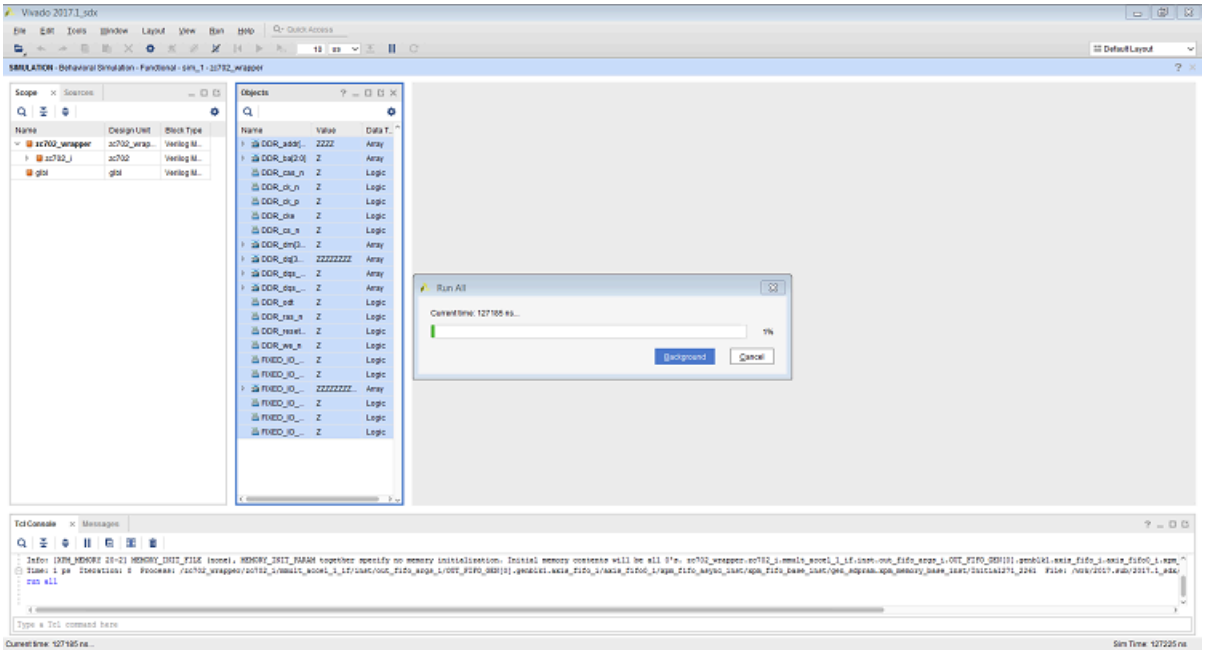
1. [Xilinx] → [Start/Stop Emulator] をクリックします。
2. [Emulation] ダイアログ ボックスが開きます。プロジェクトと設定を選択します。



3. 波形を表示するかどうかを選択します。波形を表示するようにすると、Vivado ツール セッションが開始されてシミュレーション ウィンドウが表示され、デザイン内のさまざまな信号の波形を確認できます。波形を表示しないようにするとエミュレーションが速く実行されます。[Show the Waveform] オプションを

オンにしたままにします。

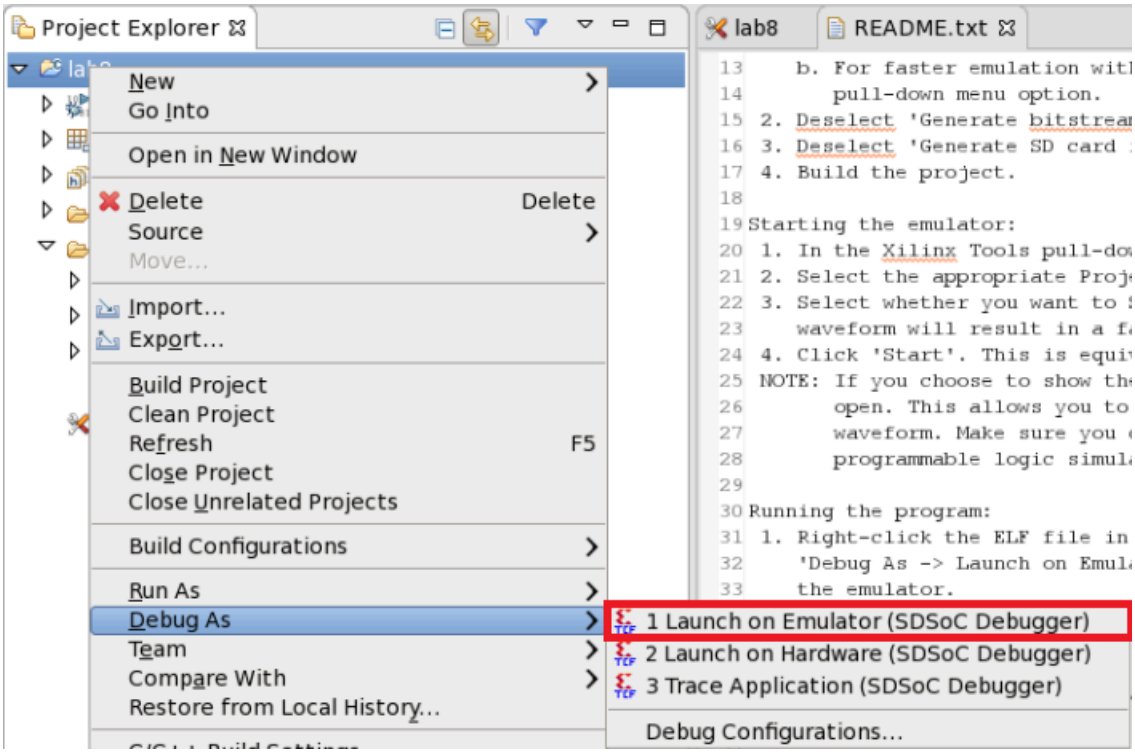
4. [Start] をクリックします。これは、ボードをオンにしたのと同様です。
5. 波形を表示するように選択すると、Vivado シミュレータ ウィンドウが表示されます。これにより、波形で表示する信号を選択できるようになります。信号を選択したら、[Run All] ボタンをクリックして、プログラマブル ロジック シミュレーションを開始します。



アプリケーションの実行

エミュレーターは、開始されるまで数秒かかります。エミュレーションを開始する方法は、次のとおりです。

1. [lab8] を右クリックして [Debug As] → [Launch on Emulator (SDSoC Debugger)] をクリックします。



2. [Confirm Perspective Switch] ダイアログ ボックスが表示されます。[Yes] をクリックしてパースペクティブを切り替えます。
3. デバッグに示すように、パースペクティブを [Debug] に切り替えると、コードを実際のハードウェアで実行しつつデバッグできます。

XILINX
ALL PROGRAMMABLE™

第 8 章

GitHub

演習 9: GitHub からのアプリケーションのインストール

このチュートリアルでは、ザイリンクス GitHub に含まれる SDSoC 環境の例をインストールする方法について説明します。

まず、SDx IDE を使用して GitHub の例をインストールします。アプリケーションをインストールしたら、デザインの OS およびプラットフォームを指定し、新しくインストールしたサンプル アプリケーションを選択します。

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。

注記: このチュートリアルは、ZC702 ボードがなくても終了できます。SDx プロジェクトを作成する際は、推奨されているエミュレーション アプリケーションのテンプレートが見つからない場合でも、ボードと使用可能なテンプレートの 1 つを選択してください。

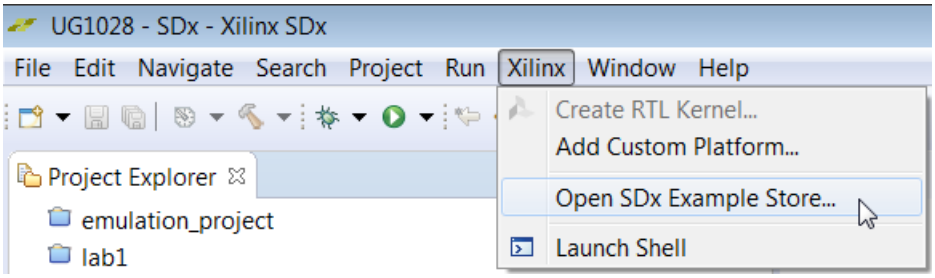
チュートリアルの目標

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。

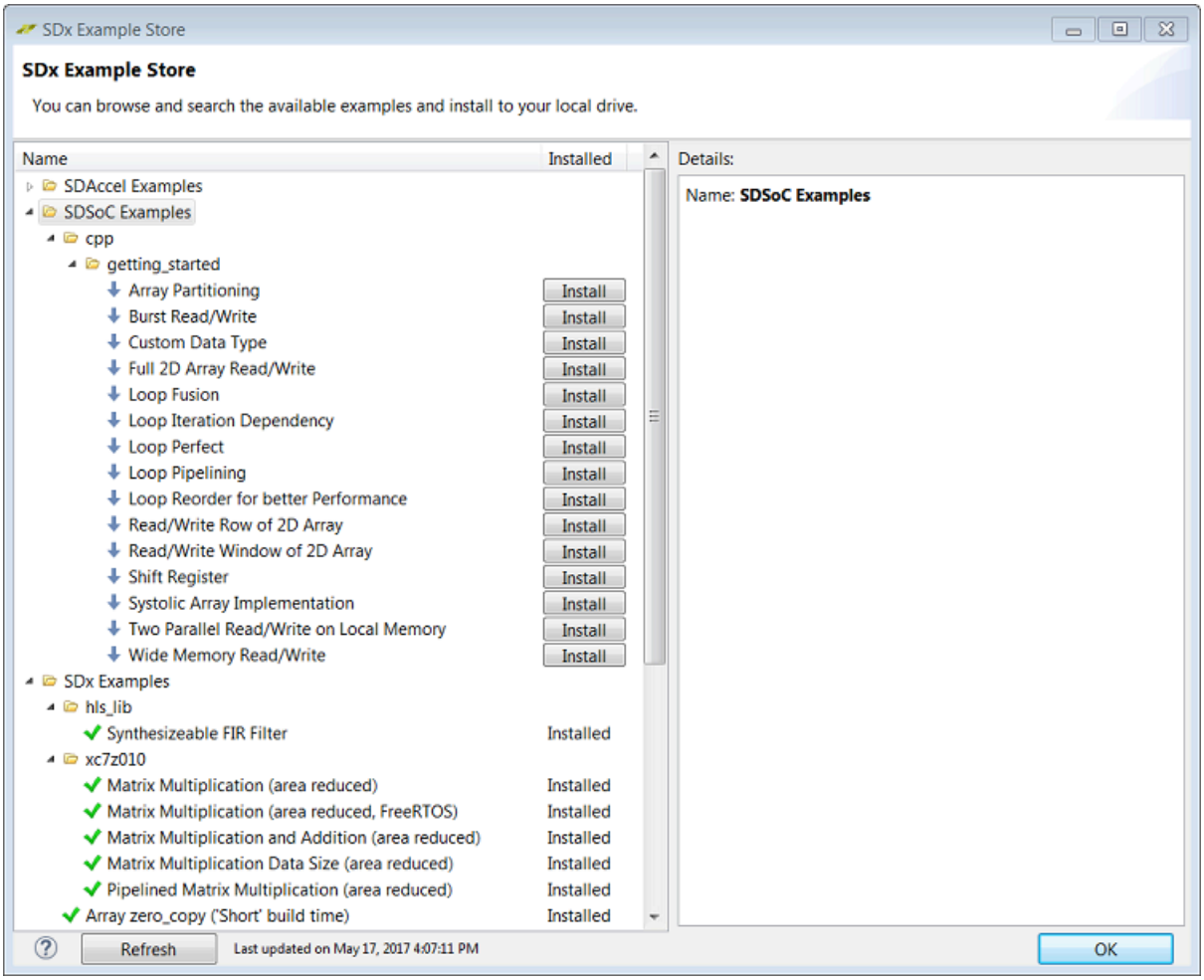
- SDx IDE を使用して、ザイリンクス GitHub から例をダウンロードしてインストールします。
- ターゲット プラットフォームでデザイン例を実行します。

GitHub からの例のダウンロードおよびインストール

1. SDx Example ストアから例をダウンロードしてインストールするには、[Xilinx] → [SDx Example Store] をクリックします。



2. [SDx Example Store] ダイアログ ボックスが開きます。次の図のように、[SDSoC Examples] フォルダーを展開します。



3. [Array Partitioning] の [Install] をクリックします。これでアプリケーションがインストールされます。
4. 必要な可能性のあるその他のデザインをインストールします。
5. [SDx Example Store] ダイアログ ボックスで [OK] をクリックします。例は、`<install_area>/Xilinx/SDx/20xx.x/examples` の下にインストールされます。

例を実行するプロジェクトの作成

1. [File] → [New] → [Xilinx SDx Project] をクリックします。
2. [Project name] フィールドにプロジェクト名を指定します (例: lab9)。[New] をクリックします。
3. [Platform] から [zc702] を選択します。[Next] をクリックします。
4. [System Configuration] ドロップダウン リストから [Linux] を選択します。[Next] をクリックします。
5. [Available Templates] から [Array Partitioning] を選択し、[Finish] をクリックします。
6. [lab9] タブをクリックして SDx プロジェクト設定を選択します (タブが表示されていない場合は、[Project Explorer] で project.sdx ファイルをダブルクリックします)。[HW functions] パネルで、プロジェクトが作成されたときに matmul_partition_accel 関数がハードウェア関数としてマークされていることを確認します。
7. ハードウェア関数が削除されていたり、マークされていない場合は、[Add HW Functions] アイコンをクリックして表示されたダイアログ ボックス内でハードウェア関数を指定します。
8. ツールバーの [Build] アイコンをクリックしてプロジェクトをビルドします。

アプリケーションの実行

ビルドが終わったら、前の章で説明したように、その他の例を実行する場合と同じ方法でアプリケーションを実行できるようになります。

XILINX
ALL PROGRAMMABLE™

付録 A

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、デザイン 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. 『SDx 環境リリース ノート、インストールおよびライセンス ガイド』(UG1238)
2. 『SDSoC 環境ユーザー ガイド』(UG1027)
3. 『SDSoC 環境最適化ガイド』(UG1235)
4. 『SDSoC 環境チュートリアル: 入門』(UG1028)
5. 『SDSoC 環境プラットフォーム開発ガイド』(UG1146)
6. SDSoC 開発環境ウェブ ページ
7. 『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)
8. 『ZC702 評価ボード (Zynq-7000 XC7Z020 All Programmable SoC 用) ユーザー ガイド』(UG850)
9. 『Vivado Design Suite ユーザー ガイド: 高位合成』(UG902)
10. 『PetaLinux ツール資料: ワークフロー チュートリアル』(UG1156)
11. Vivado® Design Suite の資料
12. 『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

本通知に基づいて貴殿または貴社 (本通知の被通知者が個人の場合には「貴殿」、法人その他の団体の場合には「貴社」。以下同じ) に開示される情報 (以下「本情報」といいます) は、ザイリンクスの製品を選択および使用することのためにのみ提供されます。適用される法律が許容する最大限の範囲で、(1) 本情報は「現状有姿」、およびすべて受領者の責任で (with all faults) という状態で提供され、ザイリンクスは、本通知をもって、明示、黙示、法定を問わず (商品性、非侵害、特定目的適合性の保証を含みますがこれらに限られません)、すべての保証および条件を負わない (否認する) ものとします。また、(2) ザイリンクスは、本情報 (貴殿または貴社による本情報の使用を含む) に関係し、起因し、関連する、いかなる種類・性質の損失または損害についても、責任を負わない (契約上、不法行為上 (過失の場合を含む)、その他のいかなる責任の法理によるかを問わない) ものとし、当該損失または損害には、直接、間接、特別、付随的、結果的な損失または損害 (第三者が起こした行為の結果被った、データ、利益、業務上の信用の損失、その他あらゆる種類の損失や損害を含みます) が含まれるものとし、それは、たとえ当該損害や損失が合理的に予見可能であったり、ザイリンクスがそれらの可能性について助言を受けていた場合であったとしても同様です。ザイリンクスは、本情報に含まれるいかなる誤りも訂正する義務を負わず、本情報または製品仕様のアップデートを貴殿または貴社に知らせる義務も負いません。事前の書面による同意のない限り、貴殿または貴社は本情報を再生産、変更、頒布、または公に展示してはなりません。一定の製品は、ザイリンクスの限定的保証の諸条件に従うこととなるので、https://japan.xilinx.com/legal.htm#tos で見られるザイリンクスの販売条件を参照してください。IP コアは、ザイリンクスが貴殿または貴社に付与したライセンスに含まれる保証と補助的条件に従うことになります。ザイリンクスの製品は、フェイルセーフとして、または、フェイルセーフの動作を要求するアプリケーションに使用するために、設計されたり意図されたりしていません。そのような重大なアプリケーションにザイリンクスの製品を使用する場合のリスクと責任は、貴殿または貴社が単独で負うものです。https://japan.xilinx.com/legal.htm#tos で見られるザイリンクスの販売条件を参照してください。

自動車用のアプリケーションの免責条項

オートモーティブ製品 (製品番号に「XA」が含まれる) は、ISO 26262 自動車用機能安全規格に従った安全コンセプトまたは余剰性の機能 (「セーフティ設計」) がない限り、エアバッグの展開における使用または車両の制御に影響するアプリケーション (「セーフティ アプリケーション」) における使用は保証されていません。顧客は、製品を組み込むすべてのシステムについて、その使用前または提供前に安全を目的として十分なテストを行うものとします。セーフティ設計なしにセーフティ アプリケーションで製品を使用するリスクはすべて顧客が負い、製品の責任の制限を規定する適用法令および規則にのみ従うものとします。

© Copyright 2017 Xilinx, Inc. Xilinx、Xilinx のロゴ、Artix、ISE、Kintex、Spartan、Virtex、Vivado、Zynq、およびこの文書に含まれるその他の指定されたブランドは、米国およびその他各国のザイリンクス社の商標です。OpenCL および OpenCL のロゴは Apple Inc. の商標であり、Khronos による許可を受けて使用されています。PCI、PCIe、および PCI Express は PCI-SIG の商標であり、ライセンスに基づいて使用されています。すべてのその他の商標は、それぞれの保有者に帰属します。

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。