

はじめての インテル® SoC FPGA 演習マニュアル (Atlas-SoC / DE10-Nano ボード版)

Ver.20.1

はじめての インテル® SoC FPGA 演習マニュアル (Atlas-SoC / DE10-Nano ボード版)

目次

本書をお読みにする前に	4
1. 概要	5
1-1. 使用環境	6
2. ボードの設定	7
2-1. ボードレイアウト	7
2-2. 電源およびケーブルの接続	7
2-3. SW10 の設定	7
3. 演習 1: ハードウェア演習	8
3-1. ステップ 1 : ハードウェア演習デザイン・プロジェクトのオープン	9
3-2. ステップ 2 : HPS コンポーネントの追加	15
3-3. ステップ 3 : HPS ペリフェラルの設定 (MAC、UART、I2C、SDIO、USB)	21
3-4. ステップ 4 : HPS クロックの設定	26
3-5. ステップ 5 : SDRAM の設定	28
3-6. ステップ 6 : HPS のクロックとエクスポート信号の設定	34
3-7. ステップ 7 : HPS コンポーネントと他のコンポーネントの接続	36
3-8. ステップ 8 : リセットの接続とベースアドレスの割り当て	38
3-9. ステップ 9 : Platform Designer システムの確認	39
3-10. ステップ 10 : Platform Designer システムの生成	41
3-11. ステップ 11 : ピン・アサインメントの設定と Quartus® Prime プロジェクトのコンパイル	45
3-12. ステップ 12 : 出力ファイルの確認	49
4. 演習 2: ソフトウェア演習(1) Preloader の生成	50
4-1. ステップ 1 : Embedded Command Shell の起動	51
4-2. ステップ 2 : bsp プロジェクトの生成	52
4-3. ステップ 3 : Preloader ビルド環境の起動	53
4-4. ステップ 4 : Preloader のビルド	55
5. 演習 3: ソフトウェア演習(2) ベアメタル・アプリケーション	59
5-1. FPGA デザインのダウンロード	60
5-2. Hello World サンプル・アプリケーションの実行	63
5-3. LED Blink サンプル・アプリケーションの実行	72

はじめての インテル® SoC FPGA 演習マニュアル (Atlas-SoC / DE10-Nano ボード版)

5-4. 演習 2 で作成した Preloader による初期化 (オプション演習)	79
5-5. システム・ヘッダーファイルによるアドレスの解決 (オプション演習)	81
6. 演習 4: Linux アプリケーション演習 (オプション演習)	84
6-1. microSD カードの準備	84
6-2. Linux 起動とログイン	86
6-3. Linux での IP アドレスとパスワードの設定	88
6-4. ホスト PC 側のネットワーク設定	89
6-5. Arm® DS の起動と Linux サンプル・アプリケーションのインポートおよびビルド	92
6-6. リモート・システム・エクスプローラー (RSE) の設定	99
6-7. Linux アプリケーションの実行・デバッグ	103
7. 今後の参考資料について	109
改版履歴	110

本書をお読みになる前に

この資料の内容は 2020 年 12 月現在のものであります。

この資料で紹介しているソフトウェアやハードウェア、操作手順などは、指定バージョンやデバイス等以外でも共通のものもありますが、一部については共通にならないものもありますので、ご注意ください。

文書中の記号

① Note	補足情報などを記載しています。
② Point	重要なポイントを記載しています。
📖 参考	理解を深めるため、参考となる資料やサイトを紹介しています。
⚠ 注記	この資料の中では具体的には触れませんが、必要となる知識や情報を記載しています。
🚫 禁止	注意点や、してはいけないことを記載しています。

文中の表記

<u>下線</u>	クリックする事で、資料中の別の章や、外部のサイトにジャンプします。
太字斜体	画面の操作をする際の、メニューやウィンドウなどに表示されている文字を示しています。
xxxxxxx□	入力するコマンド文字列を示しています。
網掛け	使用するツールを示しています。

1. 概要

この演習では、Cyclone® V SoC FPGA 評価キット「DE0-Nano-SoC Kit / Atlas-SoC Kit」(以下、Atlas-SoC ボード)、または「DE10-Nano Kit」(以下、DE10-Nano ボード)を使用して、Cyclone® V SoC のハードウェア、ソフトウェアそれぞれの開発方法について解説します。

この演習を実行することにより、インテル® SoC FPGA の開発環境である インテル® Quartus® Prime 開発ソフトウェアやシステム構成ツールである Platform Designer (旧: Qsys システム統合ツール)、およびソフトウェア開発環境である インテル® SoC FPGA エンベデッド開発スイート (以下、SoC EDS) の基本的な操作を学ぶことができます。

演習は、以下の 4 つで構成されています。

- 演習 1: ハードウェア演習
- 演習 2: ソフトウェア演習 (1)
- 演習 3: ソフトウェア演習 (2)
- 演習 4: Linux アプリケーション演習 (オプション演習)

演習 1 では、Quartus® Prime を使用して Arm® プロセッサを含むハードウェアを構成し、簡易的な SoC システムを設計します。

演習 2 では、SoC EDS ツールを使用して 28nm 世代のブートローダーである Preloader の生成を行います。

演習 3 では、Arm® Development Studio for Intel® SoC FPGA Edition (以下、Arm® DS) を利用したソフトウェア開発およびベアメタル・アプリケーションのデバッグを実施します。

演習 4 では、SD カードイメージを使用し、SoC デバイス上で Linux を動作させ、その上でアプリケーションを Arm® DS を使用して実行・デバッグします。

① Note:

演習 4 は、弊社開催の「SoC スタートアップ・トライアル・セミナー」では、時間の都合上実施しないオプション演習となります。

① Note:

SoC EDS が標準採用する統合開発環境ツールは、v20.1 から Arm® Development Studio (Arm® DS) に変更となりました。v19.4 以前の環境向けは旧製品の Arm® Development Studio 5 (DS-5™) となります。

1-1. 使用環境

この演習では、以下のソフトウェアを使用します。

- インテル® Quartus® Prime Standard Edition v20.1 (Lite Edition でも可能)

また Device データとして Cyclone® V を登録しておく必要があります。

ダウンロードとインストール方法については以下のサイトをご参照ください。

[インテル® Quartus® Prime 開発ソフトウェアおよび ModelSim® - Intel® FPGA Edition のダウンロード方法](#)

[インテル® Quartus® Prime 開発ソフトウェアおよび ModelSim® - Intel® FPGA Edition のインストール方法](#)

- インテル® SoC FPGA エンベデッド開発スイート Standard Edition v20.1 (以下、SoC EDS)

インストール方法に関しては以下のサイトをご参照下さい。

[インテル® SoC FPGA エンベデッド・デベロップメント・スイート \(SoC EDS\) のインストール方法](#)

- 演習データ (SoC-Trial_Seminer_Lab_data_atlas_de10nano_v20.1_r2.exe)

演習データの .exe ファイルをダブルクリックすると、デフォルトでは次の場所に展開されます:

C:\¥lab¥soc_lab¥cv_soc_lab

本資料では演習データを上記の場所に展開したものと説明しています。

- ホスト PC の OS : Windows® 10 Enterprise

この演習では、Windows® 10 Enterprise (バージョン 1803) を使用して動作の確認を行っております。

注記:

Windows® 10 環境にて SoC EDS v20.1std を利用する場合、bsp-create-settings と sopc-create-header-files というツールを実行する際には事前にエラーの対策が必要となります。SoC EDS のセットアップを行った後に、以下の参考情報サイトに記載の内容についてもご確認ください。

■ 参考: アルティマ技術サポート「[SoC EDS 環境で bsp-create-settings が実行エラーになるトラブルの回避策](#)」

■ 参考: アルティマ技術サポート「[SoC EDS 環境で sopc-create-header-files が実行エラーになるトラブルの回避策](#)」

※本演習では該当の 2 つのツールを使用するため、上記両方に対して対策を行う必要があります。

2. ボードの設定

このセクションでは、演習 1、2、3 を実施するために必要なボードのセットアップに関して解説します。

2-1. ボードレイアウト

本演習で使用する Atlas-SoC ボードのレイアウト図を以下に示します。

DE10-Nano ボードも基本的には同じです。

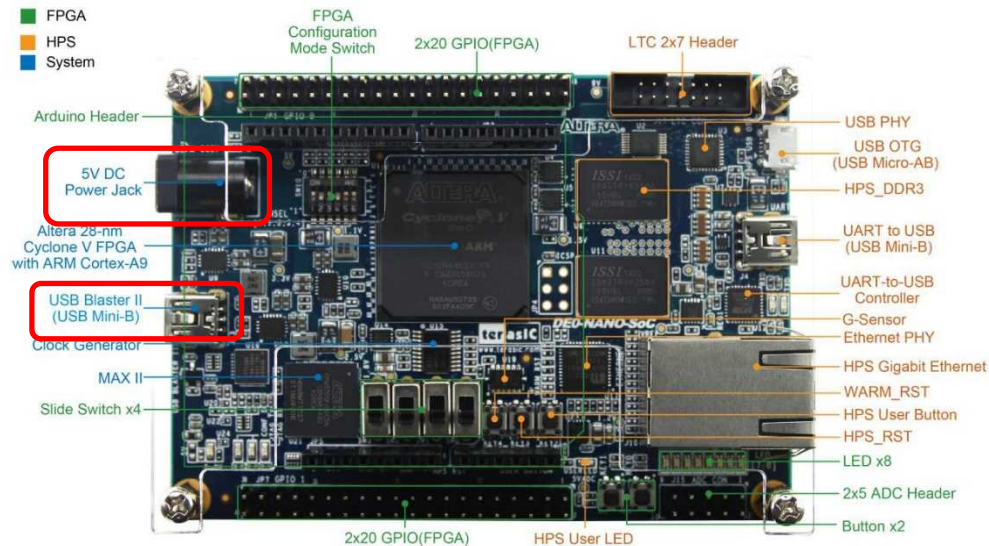


図 2-1. Atlas-SoC ボードレイアウト図

2-2. 電源およびケーブルの接続

AC アダプターの接続や各種ケーブルは以下の通り接続してください。

- 電源(AC アダプター)を DC 入力 (J14) に接続します。
- Mini USB ケーブルで作業用 PC とオンボード USB-Blaster™ II コネクタ (J13) を接続します。

2-3. SW10 の設定

SW10 (MSEL 設定スイッチ) が以下の通り設定されていることを確認します。

この設定により、FPGA は FPPx32 モードとなります。

表 2-1. SW10 の設定

ボード・リファレンス	信号名	設定
SW10. 1	MSEL0	ON ("0")
SW10. 2	MSEL1	OFF ("1")
SW10. 3	MSEL2	ON ("0")
SW10. 4	MSEL3	OFF ("1")
SW10. 5	MSEL4	ON ("0")
SW10. 6	N/A	N/A

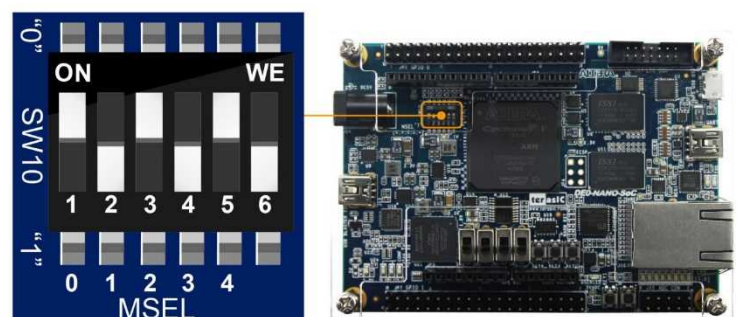


図 2-2. ジャンパーの設定

3. 演習 1: ハードウェア演習

このセクションでは、Quartus® Prime および Platform Designer を使用し、以下に示す Arm® プロセッサを含むハードウェアの設計を行います。

インテル® SoC FPGA では Cyclone® V に限らず、Quartus® Prime に含まれている Platform Designer というツールを使用してシステムを構成します。この Platform Designer では Hard Processor System (以下、HPS) のブロックをはじめ、FPGA 側に実装することのできるコンポーネント群が用意されており、所望のコンポーネントのみを実装することでリソースの最適化を図ることができます。また作成したシステムはペリフェラルが対応していれば、簡単に他のデバイスに移植できますので、それ自体も設計資産として活用していただくことが可能です。

本演習では演習時間を短縮するため、あらかじめ Platform Designer システム内にいくつかのコンポーネントとクロックソース・コンポーネントが実装してあります。このため、HPS ブロック（太枠で囲われた青色のブロック）の追加と既存コンポーネントの接続を実施します。演習内容は以下の通りです。

演習内容:

- HPS コンポーネントを既存の Platform Designer システムへ追加
- HPS インターフェイスと他のパラメーターの設定
- 既存コンポーネントと HPS との接続
- Platform Designer システムの生成

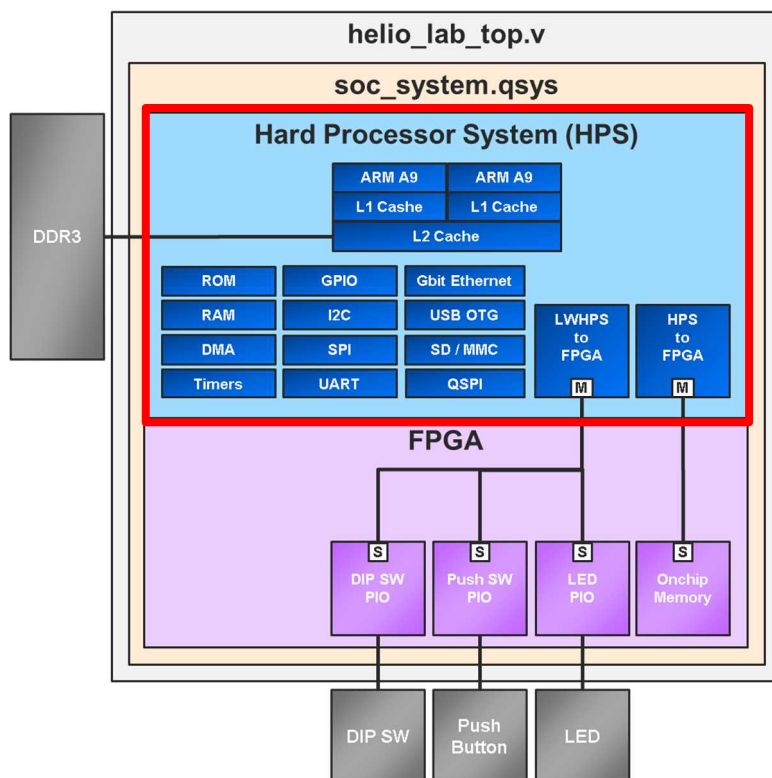


図 3-1. 演習 1 で設計する SoC システムのブロック図

3-1. ステップ 1 : ハードウェア演習デザイン・プロジェクトのオープン

演習を進めるにあたり、本演習マニュアルの各ステップに記載されている全ての説明をよく読み慎重に作業を進めてください。

本資料では作業ディレクトリーを C:\lab\soc_lab フォルダとして説明をします。作業フォルダを変更された場合は設定した環境に合わせて読み直してください。

では、はじめましょう。

- ____ 1. インストールされている Quartus® Prime 20.1 Standard Edition (Lite Edition でも可能) 開発ソフトウェア内の、Quartus® Prime を起動します。デフォルトのままであれば下記にあります。

Windows スタート ⇒ Intel FPGA 20.1.1.720 Standard Edition / Lite Edition ⇒ Quartus (Quartus Prime 20.1)

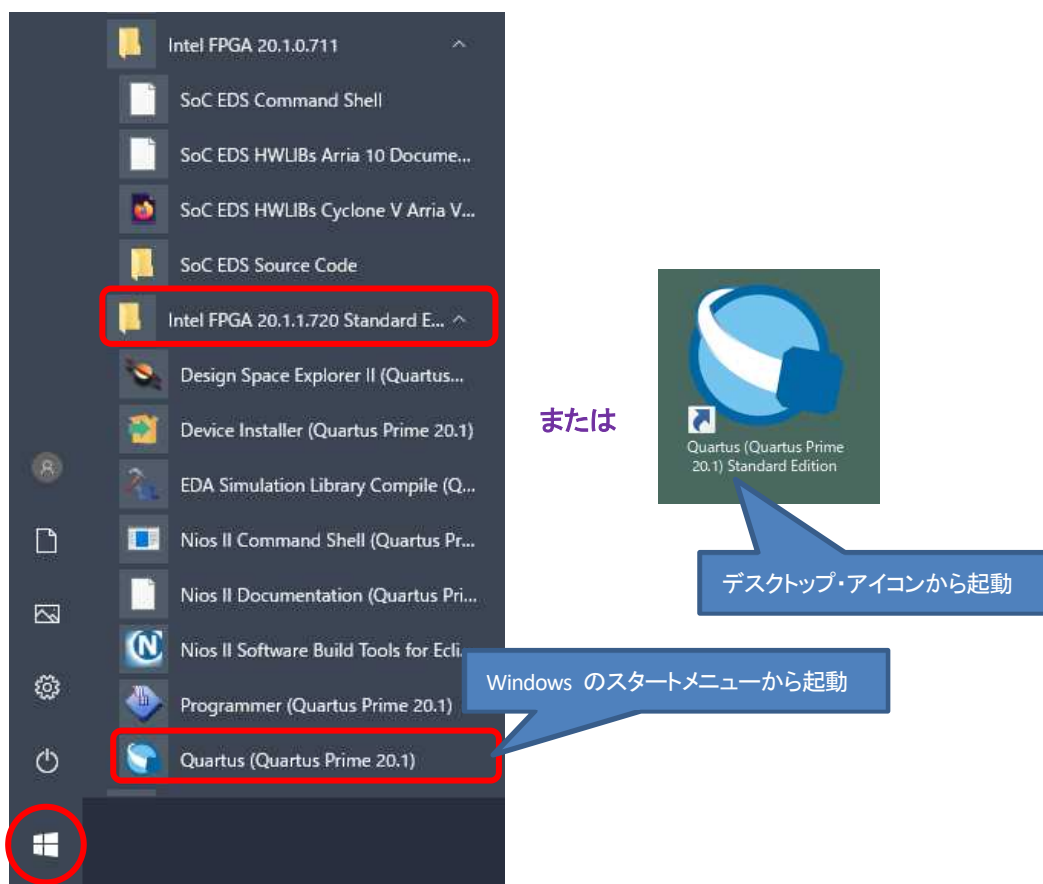


図 3-2. Quartus® Prime の起動

2. Quartus® Prime メニューバーから、**File** ⇒ **Open Project** を選択し、C:\lab\soc_lab\cv_soc_lab にある **soc_system.qpf** を選択します。この qpf ファイルは Quartus® Prime でのプロジェクト・ファイルとなっています。

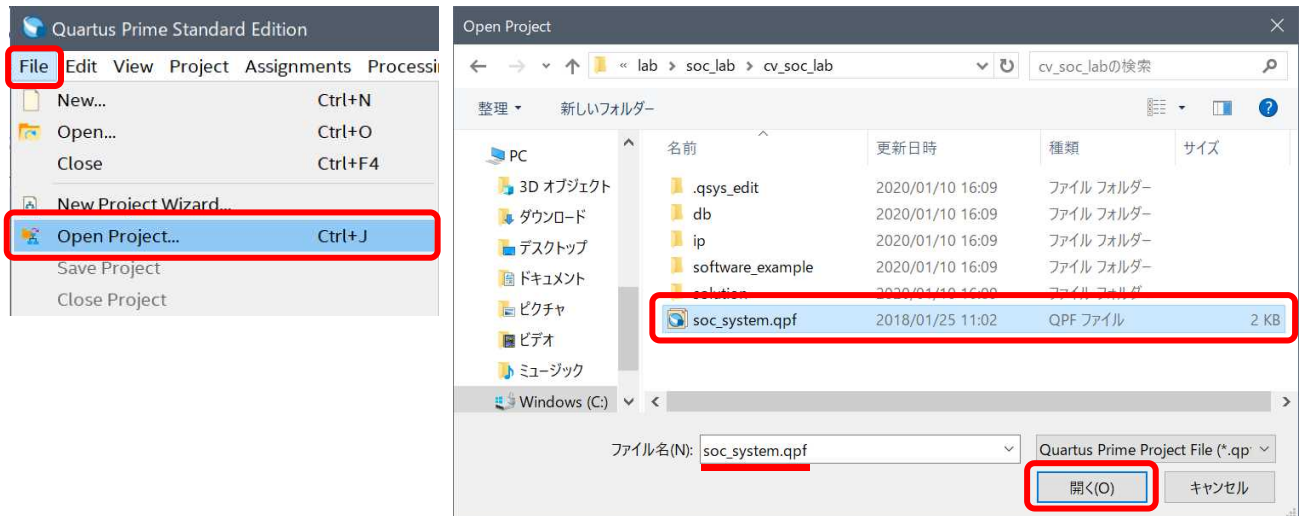


図 3-3. Quartus® Prime プロジェクトのオープン

3. ボードの選択を行います。図を参考に、使用するボードを設定してください。

- DE0-Nano-SoC / Atlas-SoC ボードの場合 : **atlas** を選択
- DE10-Nano ボードの場合 : **DE10-Nano** を選択

この設定を行うことにより、今回使用するボードに合わせ、あらかじめ設定済みのピンの配置や使用するデバイスなどの情報を使用することができるようになります。

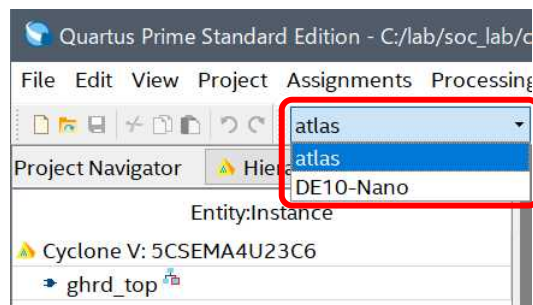


図 3-4. 使用ボードの選択

4. Quartus® Prime の **Tools** ⇒ **Platform Designer** を起動します。または、ツールバーにある Platform Designer のアイコン  をクリックし、Platform Designer を起動します。

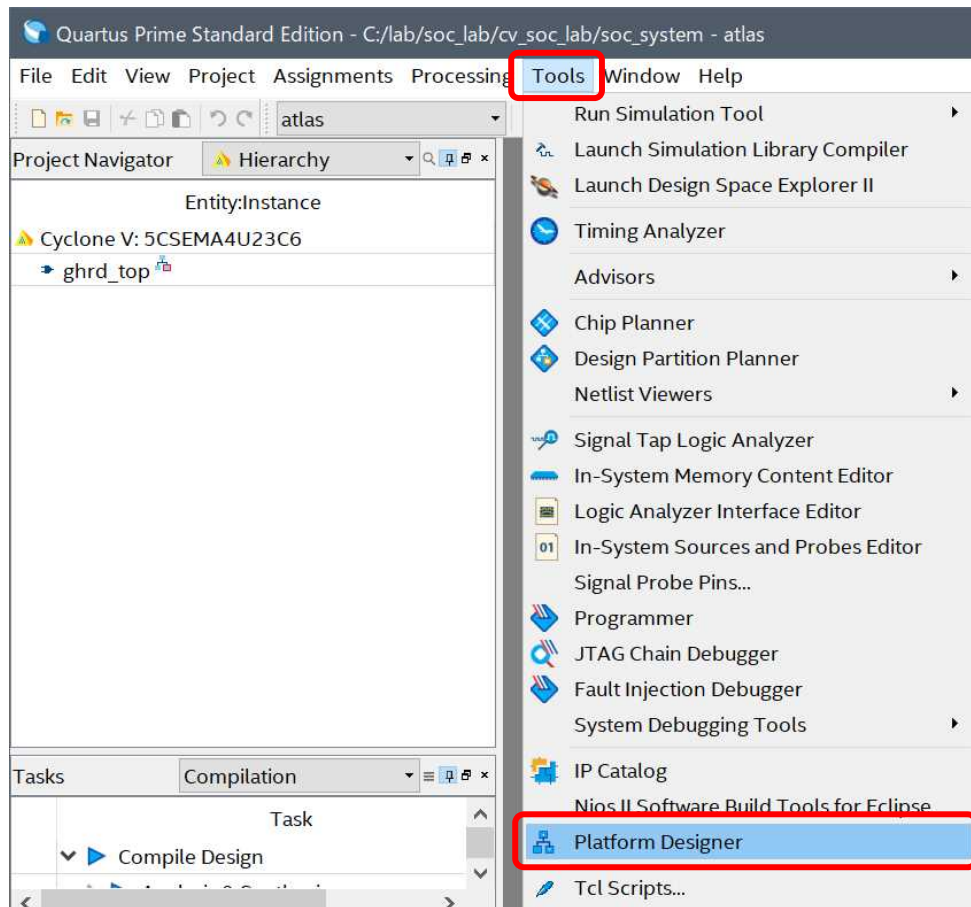


図 3-5. Platform Designer の起動

5. Soc_system.qsys ファイルを開きます。

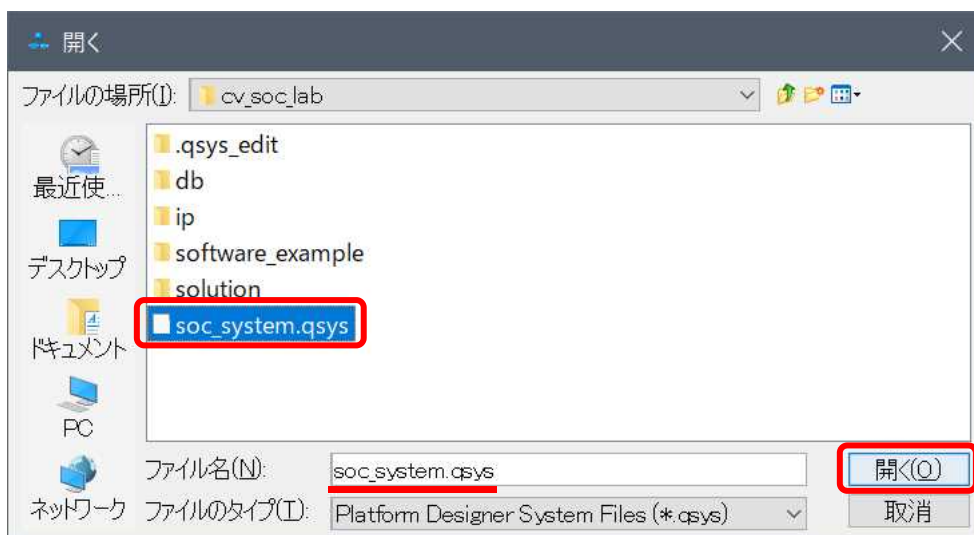


図 3-6. Platform Designer ファイルのオープン

まずは簡単に Platform Designer の使い方について説明します。

Platform Designer では主に IP Catalog と System Contents、そして Message Window の 3 つの画面があります。

IP Catalog には Platform Designer で使用できるコンポーネントがラインナップされています。この中から実装したいコンポーネントを System Contents に追加します。そして System Contents 内のコンポーネント同士を接続し、システムを作成します。

HPS と呼ばれるチップ内のハードマクロ化された部分に関してもソフト・コンポーネントとして IP Catalog 上にラインナップされており、このコンポーネントを Platform Designer システムに実装することで SoC デバイスの HPS 側が使用できるようになります。

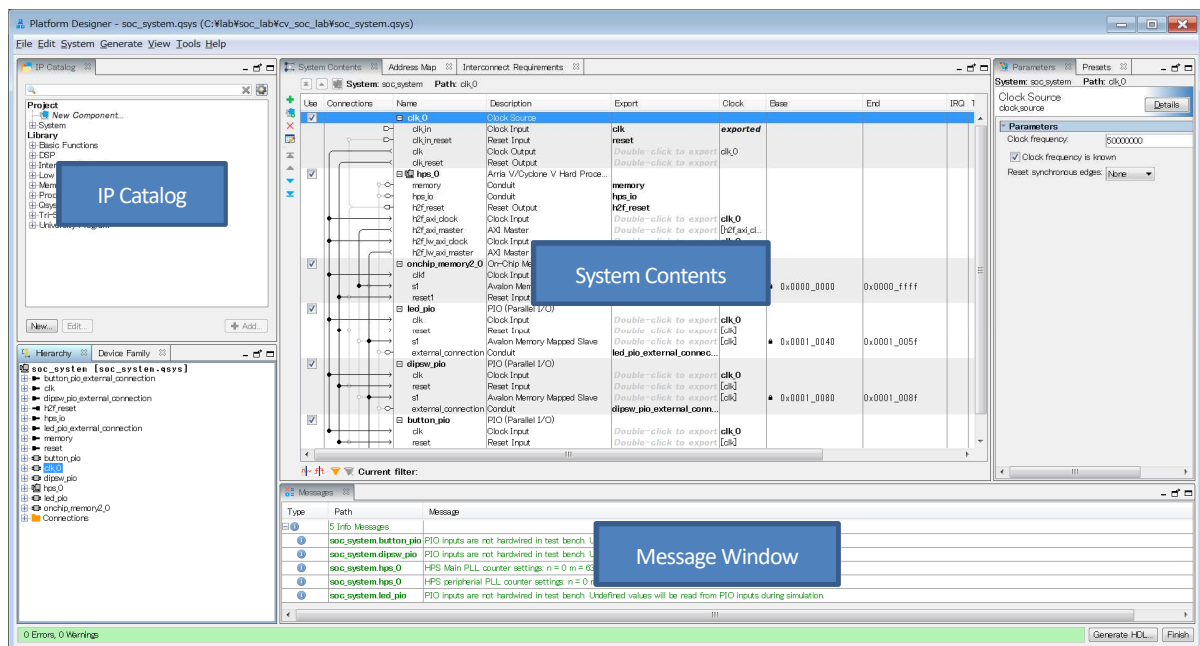


図 3-7. Platform Designer 画面

オープンした Platform Designer システムは以下のコンポーネント（白色）が実装済みとなっています。このシステムに対して HPS ブロック（青色）の追加と設定、そして実装済みコンポーネントの接続を行います。

実装済みのコンポーネント（白色）:

- クロックソース
- オンチップ・メモリー
- LED / Button 制御用 PIO ペリフェラル
 - DIP スイッチ PIO
 - ボタン PIO
 - LED PIO

演習で追加するコンポーネント（青色）:

- HPS

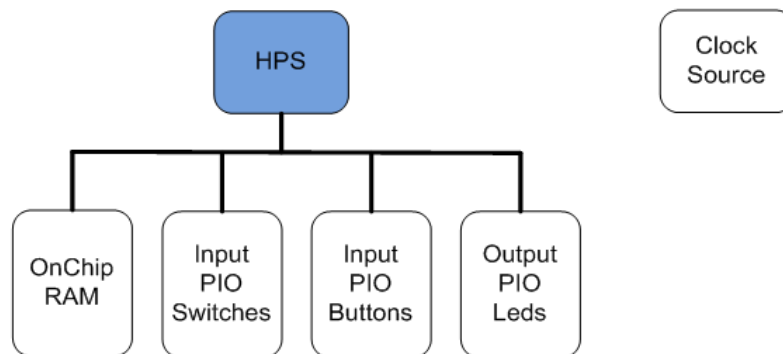


図 3-8. 設計する Platform Designer システム

Platform Designer では各 IP ごとに設定画面が用意されており、System Contents 内のコンポーネントをダブルクリックすると、そのコンポーネントの設定画面を開くことができます。

___ 6. Clock Source コンポーネント (clk_0) をダブルクリックして、**Clock Frequency** は開発ボード上の発振器と一致させるため、50 MHz に設定されていることを確認してください。

___ 7. **Clock frequency is known** がチェックされていることを確認してください。

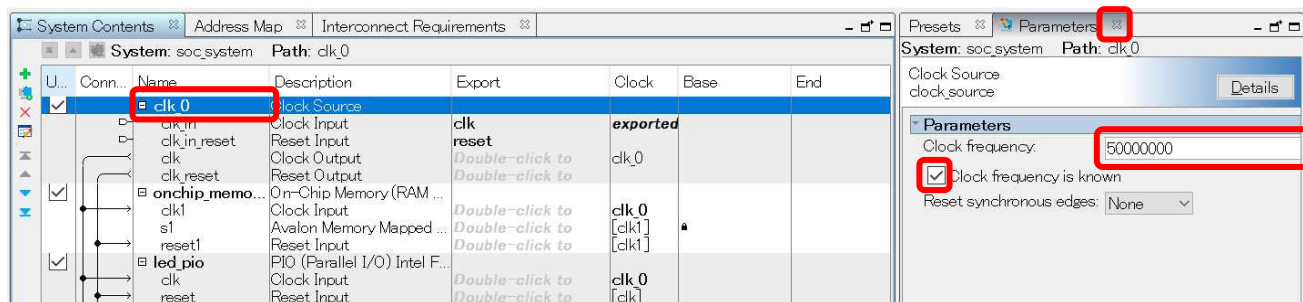


図 3-9. Clock Source の確認

___ 8. **Parameters** タブの [閉じる] (× マーク) をクリックし、**Parameters** タブを閉じます。

Platform Designer の各コンポーネントの設定は **Parameters** タブを閉じて Platform Designer を閉じない限り保持されます。

3-2. ステップ 2 : HPS コンポーネントの追加

HPS は、Dual-core Arm® Cortex™-A9 MPCore™ プロセッサと様々なペリフェラルから構成されています。また、以下に示す通り、インテル® SoC FPGA には、大きく分けて HPS 部と FPGA 部の 2 つのブロックから構成されます。

このステップでは、Platform Designer システムに HPS ブロックの追加と設定を行います。この Platform Designer システム上の HPS ブロックにおいて、HPS 部の設定を行うことができます。

HPS を設定するために使用する GUI には複数のタブ (FPGA interfaces、Peripheral Pins、HPS Clocks、SDRAM) が用意されており、それぞれについて設定を行います。

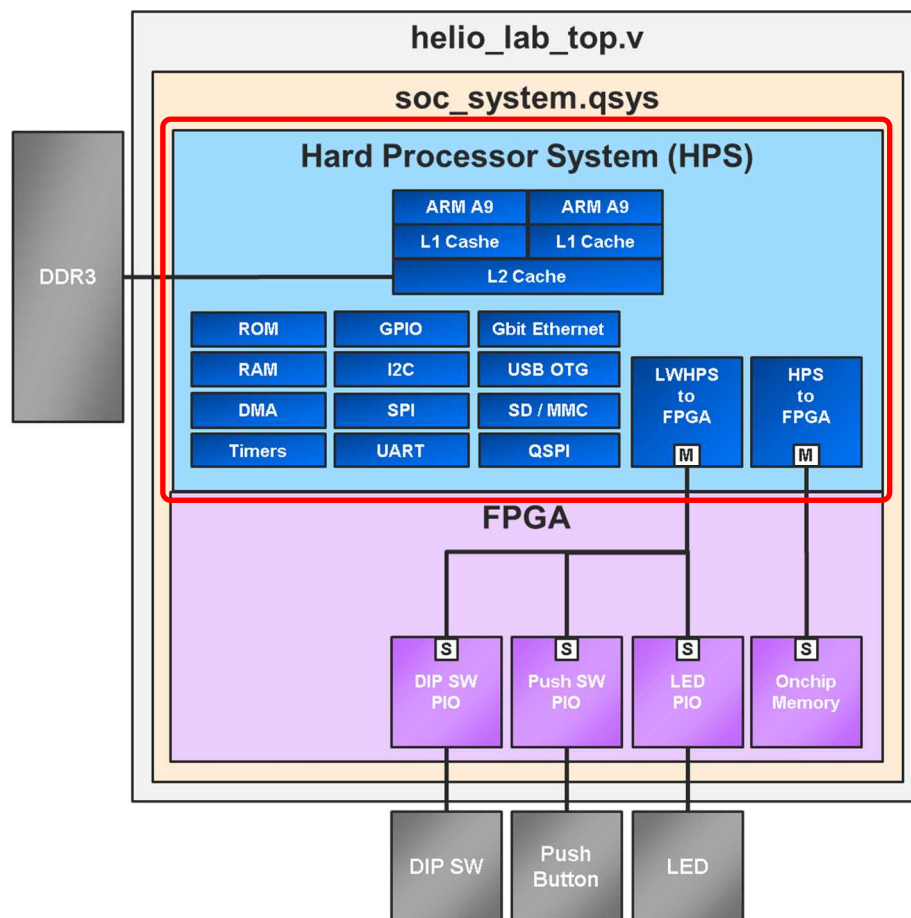


図 3-10. Platform Designer システムに追加する HPS ブロック

次ページより、Platform Designer システムに HPS ブロックを追加および各種設定を行います。

1. **IP Catalog** タブの下の 検索ボックスに、**processor** と入力します。

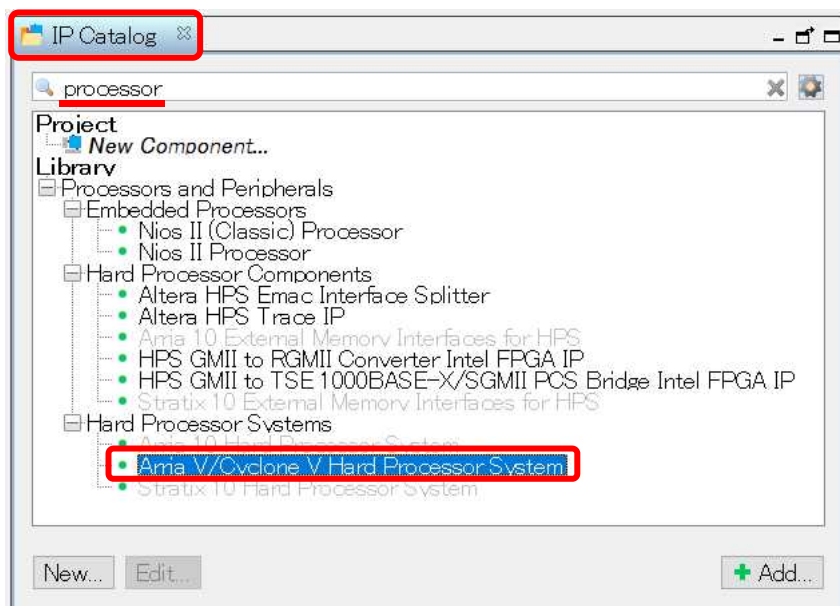


図 3-11. IP Catalog の検索ボックス

2. **Arria V/Cyclone V Hard Processor System** をダブルクリックします。

このコンポーネントが HPS コンポーネントを設定するブロックです。これから設定する HPS コンポーネントのダイアログボックスが表示されます。このウィンドウは初回のみ別ウィンドウとして起動します。[Finish] ボタンをクリック後、2 回目以降に再表示させる場合には、**System Contents** タブから HPS コンポーネントをダブルクリックしてください。

FPGA Interfaces タブではデバイス内部で接続される HPS と FPGA 間の信号の使用有無を設定することができます。設定次第で HPS 側のステータスを FPGA に通知したり、FPGA 側から HPS 側を制御したりすることができます。

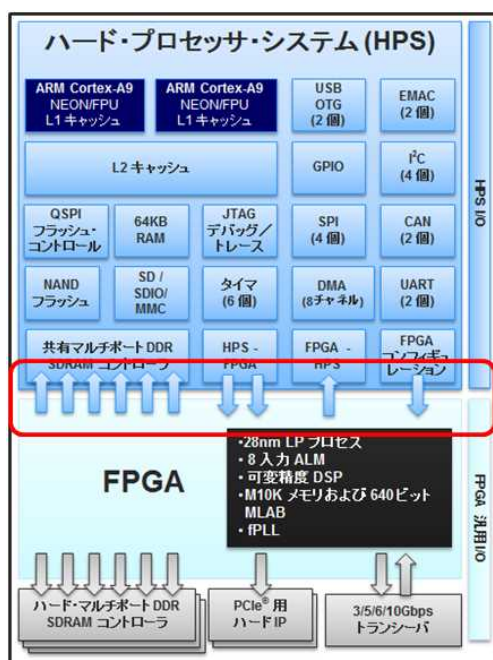


図 3-12. HPS のペリフェラルと FPGA との内部バス

3. **FPGA Interfaces** タブをクリックして、デフォルトで有効になっている **Enable MPU standby and event signals** のチェックを外して無効にします。

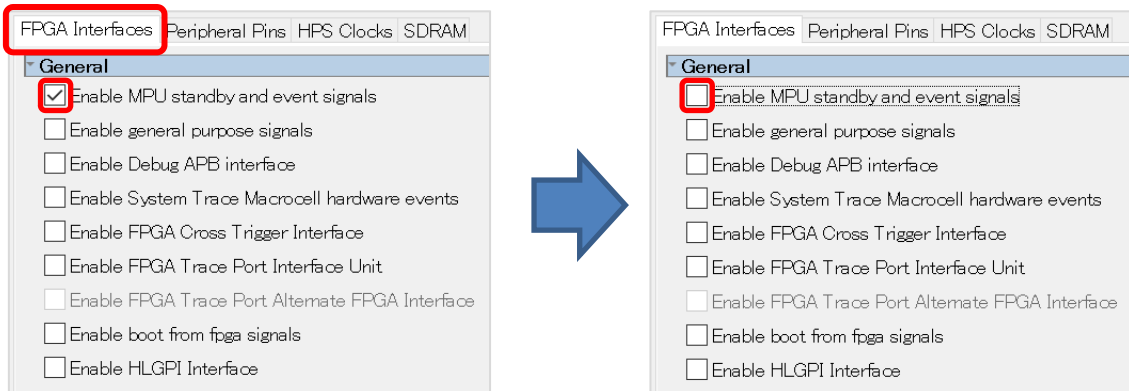


図 3-13. FPGA Interface タブの設定

① **Note:**

これは、マイクロ・プロセッサがスタンバイモードであるか、CPU がウェイクアップ可能かを示す内部信号です。恒久的に有効にするためこの入力信号を論理 High に接続する、もしくはプロセッサのイベントとして接続することもできます。

4. **Enable HLGPI Interface** のチェックが外れて無効（デフォルト）になっていることを確認します。

① **Note:**

これは、SDRAM インターフェイスで未使用のピン（14bit）を入力専用の汎用ピンとして使用する際のオプションです。この演習では、この信号は必要ありません。

次に HPS と FPGA 間のブリッジの設定を行います。

HPS と FPGA 間にはそれぞれがマスター、スレーブになるポートがあります。ポート数としては HPS から FPGA へ 2 系統、FPGA から HPS へ 1 系統です。HPS から FPGA への 2 系統ポートはそれぞれ HPS-to-FPGA interface、Lightweight HPS-to-FPGA interface です。FPGA から HPS への 1 系統のポートは FPGA-to-HPS です。すべてのポートについて、アクセスするパスに応じたバス幅の設定やポートの使用有無を設定することができます。

Arm® プロセッサや HPS 側の Master からアクセスする場合は、「ブリッジのアドレス + FPGA 側のコンポーネントのオフセットアドレス」のアドレスを指定することでアクセスすることができます。ブリッジのアドレスは下記の図のように

HPS-to-FPGA interface が 0xC000_0000

Lightweight HPS-to-FPGA interface が 0xFF20_0000

と決まっています。

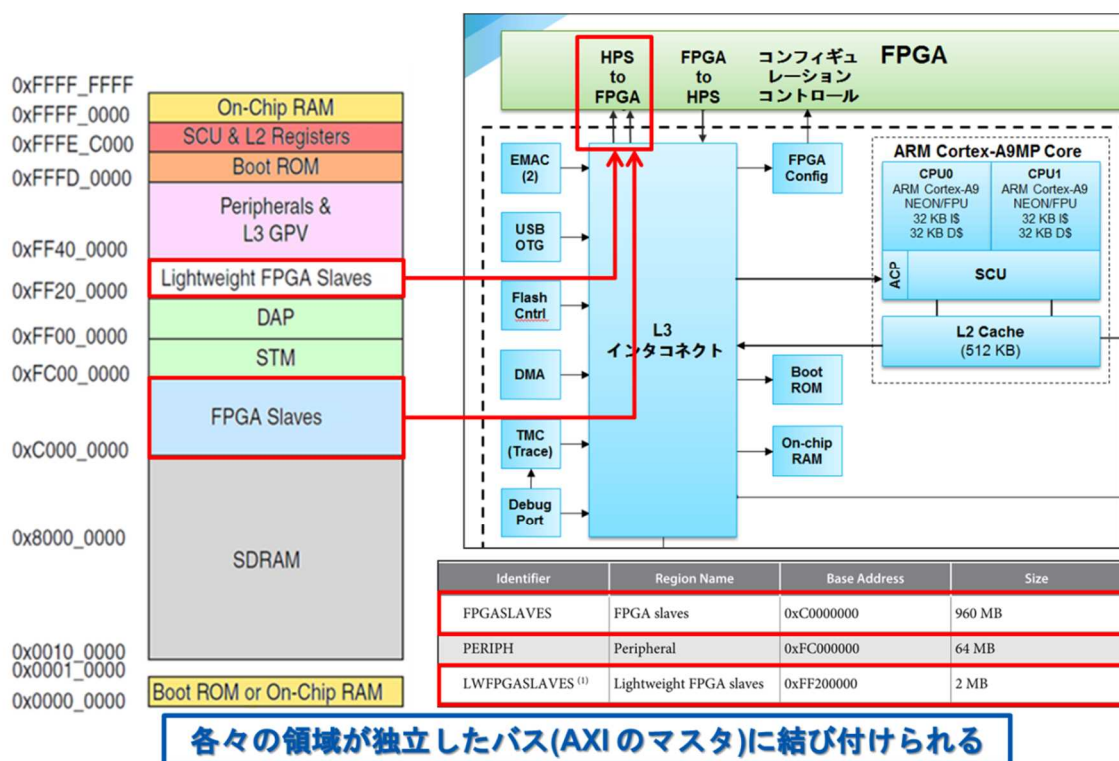


図 3-14. HPS と FPGA の内部バスと Arm から見たアドレスマップ

参考:

HPS – FPGA 間のインターフェイスに関しては、マクニカ・ホームページ技術情報にも資料がございますので、併せてご参照ください。

[SoC はじめてガイド – HPS-FPGA 間のアクセス方法](#)

次ページより設定を行います。

5. **AXI Bridges** セクションにて、**FPGA-to-HPS interface width** を **Unused**、**HPS-to-FPGA interface width** を **64-bit**、**Lightweight HPS-to-FPGA interface width** を **32-bit** に設定してください。



図 3-15. AXI Bridges の設定

Note:

FPGA-to-HPS interfaces を有効にすると、FPGA 内のマスターが HPS のペリフェラルにアクセスすることができます。この演習では使用しません。

HPS-to-FPGA interface を有効にすると、HPS がマスターとなり FPGA のペリフェラルにアクセスすることができます。HPS-to-FPGA interfaces は、32 / 64 / 128 bit 幅を選択できますが、この演習では中間の 64bit 幅を使用します。

6. **FPGA interface** ページを下にスクロールすると、**FPGA-to-HPS SDRAM interface**、**Resets** および **DMA Peripheral Request** セクションなどさらに多くのオプションがあります。

7. **FPGA to HPS SDRAM Interface** が表示されるまで **FPGA interface** ウィンドウをスクロールします。

8. **F2h_sdram0** インターフェイスをクリックし、**-** ボタンをクリックして、インターフェイスを削除します。

こちらは FPGA から HPS 側の SDRAM ヘダイレクトにアクセスできる広帯域ポートです。インターコネクトと ACP (アクセラレーター・コヒーレンシー・ポート) を介さないのが高速にアクセスできます。その反面、データのコヒーレンシーはユーザーがとる必要があります。

今回は使用しませんのでポートを削除します。

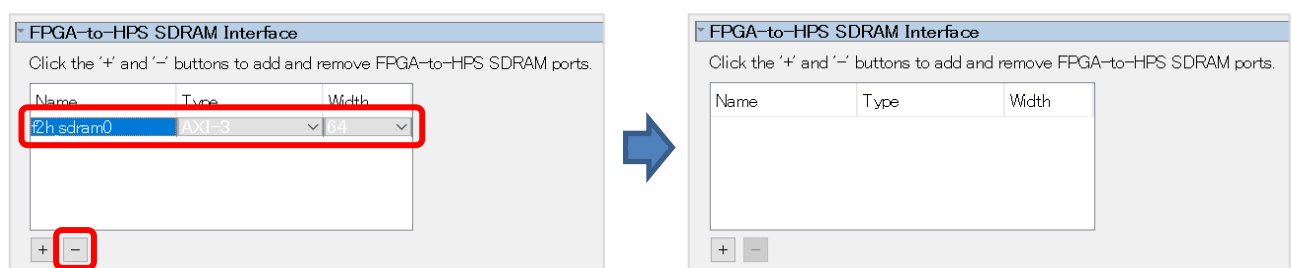


図 3-16. FPGA-to-HPS SDRAM インターフェイスの設定

- ___ 9. **Resets** セクションまでスクロールダウンします。
- ___ 10. **Resets** セクションでは、HPS リセットのためのすべてのオプションが**無効**になっていることを確認します。
- ___ 11. **DMA Peripheral Request** セクションでは、**Enabled** 列の下すべての行が **No** と表示されていることを確認します。

① **Note:**

DMA peripheral request を有効にすると、HPS 側の DMA コントローラーの Peripheral Request 信号を FPGA ファブリック側へ接続可能になります。

Peripheral Request 信号を利用した DMA 転送を行う場合を除き、通常は No をセットします。

- ___ 12. **Interrupts** セクションの、**Enable FPGA-to-HPS interrupts** オプションが**無効**になっていることを確認します。
今回は FPGA に実装したコンポーネントから Arm® プロセッサに対して割り込みは使用しません。

Resets / DMA / Interrupts の設定は以下の通りです（デフォルトから変更はありません）:

The screenshot shows a configuration window with three main sections:

- Resets:** Contains five checkboxes, all of which are unchecked:
 - Enable HPS-to-FPGA cold reset output
 - Enable HPS warm reset handshake signals
 - Enable FPGA-to-HPS debug reset request
 - Enable FPGA-to-HPS warm reset request
 - Enable FPGA-to-HPS cold reset request
- DMA Peripheral Request:** Contains a table with 6 rows (Peripheral Req. 0 to 5). The 'Enabled' column for all rows is set to 'No'.

Peripheral Req.	Enabled
0	No
1	No
2	No
3	No
4	No
5	No
- Interrupts:** Contains a checkbox for 'Enable FPGA-to-HPS Interrupts' which is unchecked. Below it is a sub-section **HPS-to-FPGA** with 20 checkboxes, all of which are unchecked:
 - Enable CAN interrupts
 - Enable clock peripheral interrupts
 - Enable CTI interrupts
 - Enable DMA interrupts
 - Enable EMAC interrupts (for EMAC0 and EMAC1)
 - Enable FPGA manager interrupt
 - Enable GPIO interrupts
 - Enable I2C-EMAC interrupts (for I2C2 and I2C3)
 - Enable I2C peripheral interrupts (for I2C0 and I2C1)
 - Enable L4 timer interrupts
 - Enable NAND interrupt
 - Enable OSC timer interrupts
 - Enable Quad SPI interrupt
 - Enable SD/MMC interrupt
 - Enable SPI master interrupts
 - Enable SPI slave interrupts
 - Enable UART interrupts
 - Enable USB interrupts
 - Enable watchdog interrupts

図 3-17. Resets / DMA / Interrupts の設定

3-3. ステップ 3 : HPS ペリフェラルの設定 (MAC、UART、I2C、SDIO、USB)

Peripheral Pins タブは HPS 内部にハードコードされている HPS ペリフェラルを有効にするタブです。

HPS のピンの多くは最大 4 つのペリフェラルで共有されています。しかしながら使用できるのは 1 つのペリフェラルのみです。そのため、有効にするペリフェラル同士でピンが競合しないように、ピンの割り当てを指定する必要があります。ピンの割り当ては最大 3 通りのパラメーター (HPS I/O Set 0 ~ 3) から選択することができます。

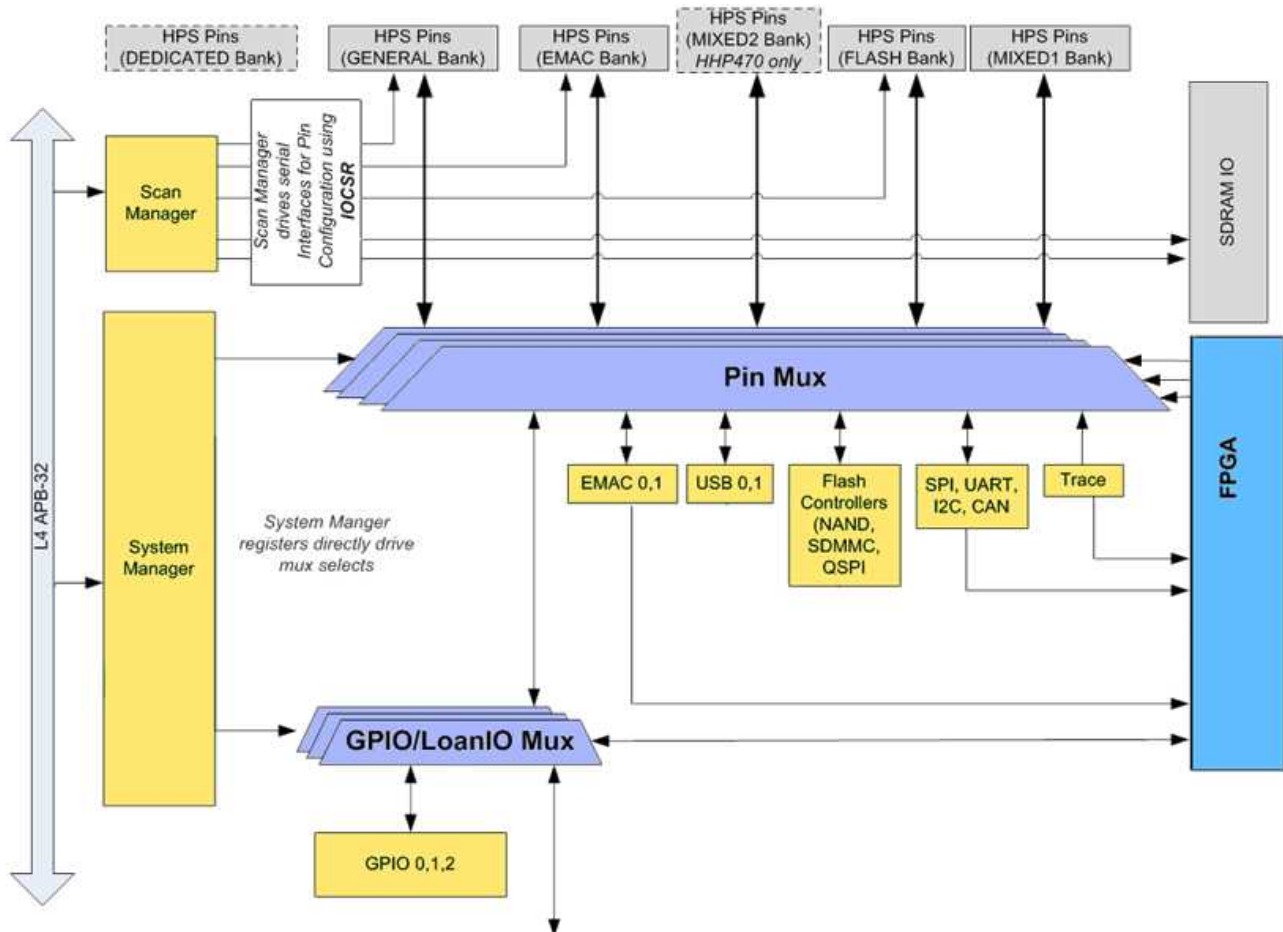


図 3-18. HPS I/O のピン・マルチプレクサー

- ___ 1. **Peripheral Pins** タブを選択します。

- ___ 2. **Ethernet Media Access Controller** の **EMAC1 pin** を **HPS I/O Set 0** に設定します。
- ___ 3. **Ethernet Media Access Controller** の **EMAC1 mode** を **RGMII** に設定します。

- ___ 4. **SD/MMC Controller** の **SDIO pin** を **HPS I/O Set 0** に設定します。
- ___ 5. **SD/MMC Controller** の **SDIO mode** を **4-bit Data** に設定します。

- ___ 6. **USB Controllers** の **USB1 pin** を **HPS I/O Set 0** に設定します。
- ___ 7. **USB Controllers** の **USB1 PHY interface mode** を **SDR with PHY clock output mode** に設定します。

- ___ 8. **SPI Controllers** の **SPIM1 pin** を **HPS I/O Set 0** に設定します。
- ___ 9. **SPI Controllers** の **SPIM1 mode** を **Single Slave Select** に設定します。

- ___ 10. **UART Controllers** の **UART0 pin** を **HPS I/O Set 0** に設定します。
- ___ 11. **UART Controllers** の **UART0 mode** を **No Flow Control** に設定します。

- ___ 12. **I2C Controllers** の **I2C0 pin** を **HPS I/O Set 0** に設定します。
- ___ 13. **I2C Controllers** の **I2C0 mode** を **I2C** に設定します。

- ___ 14. **I2C Controllers** の **I2C1 pin** を **HPS I/O Set 0** に設定します。
- ___ 15. **I2C Controllers** の **I2C1 mode** を **I2C** に設定します。

設定後のパラメーターは次ページを参照してください。

The diagram illustrates the configuration of HPS peripherals in the Quartus II software. The left panel shows the initial state with 'Peripheral Pins' highlighted. The right panel shows the configuration after selecting 'HPS I/O Set 0' for various peripherals. A blue arrow indicates the transition from the initial state to the configured state.

Peripheral	Initial State (Left)	Configured State (Right)
Ethernet Media Access Controller	EMAC0 pin: Unused EMAC0 mode: N/A EMAC1 pin: Unused EMAC1 mode: N/A	EMAC0 pin: Unused EMAC0 mode: N/A EMAC1 pin: HPS I/O Set 0 EMAC1 mode: RGMII
NAND Flash Controller	NAND pin: Unused NAND mode: N/A	NAND pin: Unused NAND mode: N/A
Quad SPI Flash Controller	QSPI pin: Unused QSPI mode: N/A	QSPI pin: Unused QSPI mode: N/A
SD/MMC Controller	SDIO pin: Unused SDIO mode: N/A	SDIO pin: HPS I/O Set 0 SDIO mode: 4-bit Data
USB Controllers	USB0 pin: Unused USB0 PHY interface mode: N/A USB1 pin: Unused USB1 PHY interface mode: N/A	USB0 pin: Unused USB0 PHY interface mode: N/A USB1 pin: HPS I/O Set 0 USB1 PHY interface mode: SDR with PHY clock output mode
SPI Controllers	SPIM0 pin: Unused SPIM0 mode: N/A SPIM1 pin: Unused SPIM1 mode: N/A SPIS0 pin: Unused SPIS0 mode: N/A SPIS1 pin: Unused SPIS1 mode: N/A	SPIM0 pin: Unused SPIM0 mode: N/A SPIM1 pin: HPS I/O Set 0 SPIM1 mode: Single Slave Select SPIS0 pin: Unused SPIS0 mode: N/A SPIS1 pin: Unused SPIS1 mode: N/A
UART Controllers	UART0 pin: Unused UART0 mode: N/A UART1 pin: Unused UART1 mode: N/A	UART0 pin: HPS I/O Set 0 UART0 mode: No Flow Control UART1 pin: Unused UART1 mode: N/A
I2C Controllers	I2C0 pin: Unused I2C0 mode: N/A I2C1 pin: Unused I2C1 mode: N/A I2C2 pin: Unused I2C2 mode: N/A I2C3 pin: Unused I2C3 mode: N/A	I2C0 pin: HPS I/O Set 0 I2C0 mode: I2C I2C1 pin: HPS I/O Set 0 I2C1 mode: I2C I2C2 pin: Unused I2C2 mode: N/A I2C3 pin: Unused I2C3 mode: N/A
CAN Controllers	CAN0 pin: Unused CAN0 mode: N/A CAN1 pin: Unused CAN1 mode: N/A	CAN0 pin: Unused CAN0 mode: N/A CAN1 pin: Unused CAN1 mode: N/A
Trace Port Interface Unit	TRACE pin: Unused TRACE mode: N/A	TRACE pin: Unused TRACE mode: N/A

図 3-19. HPS ペリフェラルの設定

Peripherals Mux Table セクションでは設定したピンの配置を確認することができます。

1 つのピンには 1 つの役割しか与えることはできません。そのため、HPS の複数ペリフェラルが同じピンを使用したり、同じピンに HPS ペリフェラルと GPIO の役割を与えたりすることはできません。そのため、この **Peripherals Mux Table** セクションで、各ピンの用途を確認してください。

左側の列はピン名を示しており、そのピンが使用されている場合は太字になります。ペリフェラルのピンとして使用していないピンは HPS の GPIO ピンとして使用可能です。その場合はピンごとの各 GPIO のボタンを押すことで有効になります。

ピンに対して競合が起きている場合は **Message Window** に **Error** が表示され、またそのピンの欄が赤くハイライトされますので、どのピンが競合を起こしているのかをリアルタイムに知ることができます。

では実際に使用していないピンを GPIO ピンに設定してみましょう。

16. **Peripherals Mux Table** セクションで **GPIO09** をクリックすることにより、**GPIO09** を有効にします。

Point:

反応に時間がかかる場合がありますので、何度も押さないように注意してください。

The diagram illustrates the process of enabling GPIO09. It shows two versions of the 'Peripherals Mux Table'. In the first table, various pins are mapped to peripherals like USB, EMAC, and GPIO. In the second table, the 'GPIO09' button in the rightmost column is highlighted with a red rectangle, indicating it has been selected for activation.

図 3-20. HPS GPIO09 の設定

Point:

もしクリックできなかった場合は、HPS component ダイアログボックスの右下にある **[Finish]** を選択、または Parameters タブの “x” をクリックして HPS component ダイアログボックスを一旦閉じた後、再度 hps_0 コンポーネントをダブルクリックしてパラメーター・ウィンドウを開き作業を続けてください。

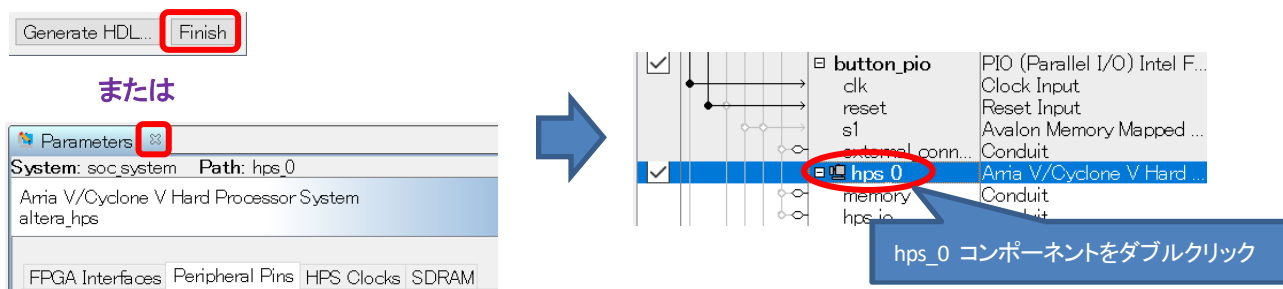


図 3-21. GPIO 設定時にクリックできなかった場合の対応方法

17. 同様に **GPIO35**、**GPIO40**、**GPIO53**、**GPIO54**、**GPIO61** を有効にします。

QSPI_CLK			QSPI_CLK (Set1) (Set0)	GPIO34	LOAN034
QSPI_SS1			QSPI_SS1 (Set1)	GPIO35	LOAN035
SDMMC_CMD		USB0_D0 (Set0)	SDIO_CMD (Set0)	GPIO36	LOAN036
SDMMC_PWREN		USB0_D1 (Set0)	SDIO_PWREN (Set0)	GPIO37	LOAN037
SDMMC_D0		USB0_D2 (Set0)	SDIO_D0 (Set0)	GPIO38	LOAN038
SDMMC_D1		USB0_D3 (Set0)	SDIO_D1 (Set0)	GPIO39	LOAN039
SDMMC_D4		USB0_D4 (Set0)	SDIO_D4 (Set0)	GPIO40	LOAN040
SDMMC_D5		USB0_D5 (Set0)	SDIO_D5 (Set0)	GPIO41	LOAN041
SDMMC_D6		USB0_D6 (Set0)	SDIO_D6 (Set0)	GPIO42	LOAN042
SDMMC_D7		USB0_D7 (Set0)	SDIO_D7 (Set0)	GPIO43	LOAN043
HPS_GPIO44		USB0_CLK (Set0)	SDIO_CLK (Set0)	GPIO44	LOAN044
SDMMC_CCLK_OUT		USB0_STP (Set0)	SDIO_D2 (Set0)	GPIO45	LOAN045
SDMMC_D2		USB0_RST (Set0)	SDIO_D3 (Set0)	GPIO46	LOAN046
SDMMC_D3		USB0_NXT (Set0)	SDIO_D4 (Set0)	GPIO47	LOAN047
TRACE_CLK			TRACE_CLK (Set0)	GPIO48	LOAN048
TRACE_D0	UART0_RX (Set0)	SPIS0_CLK (Set0)	TRACE_D0 (Set0)	GPIO49	LOAN049
TRACE_D1	UART0_TX (Set0)	SPIS0_MISO (Set0)	TRACE_D1 (Set0)	GPIO50	LOAN050
TRACE_D2	ZC1_SDA (Set0)	SPIS0_MISO (Set0)	TRACE_D2 (Set0)	GPIO51	LOAN051
TRACE_D3	ZC1_SCL (Set0)	SPIS0_SSD (Set0)	TRACE_D3 (Set0)	GPIO52	LOAN052
TRACE_D4	CAN1_RX (Set0)	SPIS1_CLK (Set0)	TRACE_D4 (Set0)	GPIO53	LOAN053
TRACE_D5	CAN1_TX (Set0)	SPIS1_MISO (Set0)	TRACE_D5 (Set0)	GPIO54	LOAN054
TRACE_D6	ZC0_SDA (Set0)	SPIS1_SSD (Set0)	TRACE_D6 (Set0)	GPIO55	LOAN055
TRACE_D7	ZC0_SCL (Set0)	SPIS1_MISO (Set0)	TRACE_D7 (Set0)	GPIO56	LOAN056
SPIM0_CLK	UART0_CTS (Set0) (Set1) (Set0)	QCI_SDA (Set1)	SPIM0_CLK (Set0)	GPIO57	LOAN057
SPIM0_MISO	UART0_RTS (Set0) (Set1) (Set0)	QCI_SCL (Set1)	SPIM0_MISO (Set0)	GPIO58	LOAN058
SPIM0_SS0	UART1_CTS (Set0)	CAN1_RX (Set1)	SPIM0_SS0 (Set0)	GPIO59	LOAN059
UART0_RX	UART1_RTS (Set0)	CAN1_TX (Set1)	SPIM0_SS1 (Set0)	GPIO60	LOAN060
UART0_TX	SPIM0_SS1 (Set0)	CAN0_RX (Set0)	UART0_RX (Set1)	GPIO61	LOAN061
		CAN0_TX (Set0)	UART0_TX (Set1)	GPIO62	LOAN062

図 3-22. HPS GPIO の設定

18. 設定後、下図のように2つのエラー以外に、エラーが出ていないことを確認してください（この2つのエラーは後で解決します）。

Type	Path	Message
2 Errors		
✖	soc_system.hps_0	hps_0.h2f_axi_clock must be connected to a clock output
✖	soc_system.hps_0	hps_0.h2f_lw_axi_clock must be connected to a clock output
7 Warnings		
⚠	soc_system.hps_0	"Configuration/HPS-to-FPGA user 0 clock frequency" (desired_cfg.clk.mhz) requested 100.0 MHz, but only achieved 97.368421 MHz
⚠	soc_system.hps_0	1 or more output clock frequencies cannot be achieved precisely, consider revising desired output clock frequencies.
⚠	soc_system.hps_0	ODT is disabled. Enabling ODT (Mode Register 1) may improve signal integrity
⚠	soc_system.button_pio	button_pio.s1 must be connected to an Avalon-MM master
⚠	soc_system.dipsw_pio	dipsw_pio.s1 must be connected to an Avalon-MM master
⚠	soc_system.led_pio	led_pio.s1 must be connected to an Avalon-MM master
⚠	soc_system.onchip_memory2_0	onchip_memory2_0.s1 must be connected to an Avalon-MM master
5 Info Messages		
i	soc_system.button_pio	PiO inputs are not hardwired in test bench. Undefined values will be read from PiO inputs during simulation.
i	soc_system.dipsw_pio	PiO inputs are not hardwired in test bench. Undefined values will be read from PiO inputs during simulation.
i	soc_system.hps_0	HPS Main PLL counter settings: n = 0 m = 73
i	soc_system.hps_0	HPS peripheral PLL counter settings: n = 0 m = 39
i	soc_system.led_pio	PiO inputs are not hardwired in test bench. Undefined values will be read from PiO inputs during simulation.

図 3-23. ピン競合エラーが無いときの表示例

例えば、図 3-24 のようなエラーが出ている場合、**SPIS0** と **UART0** のピンの競合が起きているので、設定に誤りがないか確認し、修正を行ってください。

この例では、本来使用しない **SPIS0** を使用することとなっているため、エラーになっています。設定を **Unused** とすると、エラーが消えます。

Parameters

System: soc_system Path: hps_0

Amia V/Cyclone V Hard Processor System

altera_hps

SPI Controllers

SPIM0 pin:

Unused

SPIM0 mode:

N/A

SPIM1 pin:

HPS I/O Set 0

SPIM1 mode:

Single Slave Select

SPIS0 pin:

HPS I/O Set 0

SPIS0 mode:

SPI

SPIS1 pin:

Unused

SPIS1 mode:

N/A

UART Controllers

UART0 pin:

HPS I/O Set 0

UART0 mode:

No Flow Control

UART1 pin:

Unused

UART1 mode:

N/A

Messages

2 Errors

✖

soc_system.hps_0

Refer to the Peripherals Mux Table for more details. The selected peripherals 'SPIS0' and 'UART0' are conflicting.

✖

soc_system.hps_0

hps_0.h2f_axi_clock must be connected to a clock output

✖

soc_system.hps_0

hps_0.h2f_lw_axi_clock must be connected to a clock output

使用しない **SPIS0** を誤って設定したため...

SPIS0 と **UART0** が競合するというエラーが発生

図 3-24. ピン競合エラーがあるときの表示例

Ver.20.1 / Rev.3 2020 年 12 月

25/110

ALTIMA Company, MACNICA, Inc.

3-4. ステップ 4 : HPS クロックの設定

HPS Clocks タブでは、Clock ソースと周波数が設定されます。これらのパラメーターは、すべて Clock Manager Component で管理されます。

このタブで設定されたパラメーターは、ブートローダー（Preloader ソフトウェア）の生成時に使用されます。Preloader は「[演習 2: ソフトウェア演習\(1\) Preloader の生成](#)」で生成します。

- ___ 1. **HPS Clocks** タブを選択します。
- ___ 2. **Input Clocks** タブを選択します。
- ___ 3. **EOSC1 / EOSC2 clock frequency** が **25MHz** に設定されていることを確認します。また、すべての **FPGA-to-HPS PLL Reference clocks** が**無効**になっていることを確認します。EOSC1 は HPS 側の専用ピンで HPS の MPU のクロックを生成するために必要なクロックソースです。今回使用している Atlas-SoC ボードや DE10-Nano ボードでは、25MHz が入っているためこのように設定しています。

Category	Parameter	Value	Unit
External Clock Sources	EOSC1 clock frequency:	25.0	MHz
	EOSC2 clock frequency:	25.0	MHz
FPGA-to-HPS PLL Reference Clocks	Enable FPGA-to-HPS SDRAM PLL reference clock	☐	
	Enable FPGA-to-HPS peripheral PLL reference clock	☐	
	FPGA-to-HPS SDRAM PLL reference clock frequency:	0.0	MHz
	FPGA-to-HPS peripheral PLL reference clock frequency:	0.0	MHz
Peripheral FPGA Clocks	EMAC0 emac0_md_clk clock frequency:	2.5	MHz
	EMAC0 emac0_gtx_clk clock frequency:	125	MHz
	EMAC1 emac1_md_clk clock frequency:	2.5	MHz
	EMAC1 emac1_gtx_clk clock frequency:	125	MHz
	QSPI qspi_sclk_out clock frequency:	100	MHz
	SPIM0 spim0_sclk_out clock frequency:	100	MHz
	SPIM1 spim1_sclk_out clock frequency:	100	MHz
	I2C0 i2c0_clk clock frequency:	100	MHz
	I2C1 i2c1_clk clock frequency:	100	MHz
	I2C2 i2c2_clk clock frequency:	100	MHz
	I2C3 i2c3_clk clock frequency:	100	MHz

図 3-25. HPS to FPGA Clock の設定

4. **Output Clocks** タブを選択します。
5. 下図のように設定されていることを確認します（デフォルトから変更はありません）。こちらのタブでは HPS の各ペリフェラルの動作周波数を設定することができます。設定した値に応じて PLL の設定値が自動計算されます。

The screenshot shows the 'HPS Clocks' configuration window with the 'Output Clocks' tab selected. The 'Clock Sources' section lists various clock sources. The 'Main PLL Output Clocks - Desired Frequencies' section shows the Default MPU clock frequency set to 925.0 MHz, with a note indicating that for DE10-Nano, it should be 800.0 MHz. Other frequencies like L3 MP, L3 SP, and L4 MP/SP are also shown. The 'Peripheral PLL Output Clocks - Desired Frequencies' section shows various peripheral clock frequencies. The 'HPS-to-FPGA User Clocks' section shows options to enable user clocks and their frequencies.

Section	Parameter	Value	Unit	Notes
Main PLL Output Clocks - Desired Frequencies	Default MPU clock frequency	925.0	MHz	DE10-Nano の場合は、800.0 MHz
	MPU clock frequency	800.0	MHz	
	L3 MP clock frequency	185.0	MHz	DE10-Nano の場合は、200.0 MHz
	L3 SP clock frequency	92.5	MHz	DE10-Nano の場合は、100.0 MHz
	Debug AT clock frequency	25.0	MHz	
	Debug clock frequency	12.5	MHz	
	Debug trace clock frequency	25.0	MHz	
Peripheral PLL Output Clocks - Desired Frequencies	SDMMC clock frequency	200.0	MHz	
	NAND clock frequency	12.5	MHz	
	QSPI clock frequency	400.0	MHz	
	EMAC0 clock frequency	250.0	MHz	
	EMAC1 clock frequency	250.0	MHz	
	USB clock frequency	200.0	MHz	
	SPI clock frequency	200.0	MHz	
	CAN0 clock frequency	100.0	MHz	
	CAN1 clock frequency	100.0	MHz	
HPS-to-FPGA User Clocks	GPIO debounce clock frequency	32000	Hz	
	Enable HPS-to-FPGA user 0 clock	☐		
	Enable HPS-to-FPGA user 1 clock	☐		
	Enable HPS-to-FPGA user 2 clock	☐		
	HPS-to-FPGA user 0 clock frequency	100.0	MHz	
	HPS-to-FPGA user 1 clock frequency	100.0	MHz	
	HPS-to-FPGA user 2 clock frequency	100.0	MHz	

図 3-26. HPS to FPGA Clock の設定

3-5. ステップ 5 : SDRAM の設定

SDRAM タブには、HPS 側の SDRAM コントローラーおよび接続する DDR に関するパラメーターを設定するオプションがあります。**SDRAM** タブの内部には SDRAM 構成のためさらに 4 つのタブ (**PHY Settings**、**Memory Parameters**、**Memory Timing**、**Board Settings**) があります。

1. **Arria V/Cyclone V Hard Processor System** ウィンドウの下部の **[Finish]** をクリックします。

この操作により、HPS コンポーネントが Platform Designer システムに追加されます (次の手順で **Presets** ウィンドウを表示するために必要な操作です)。

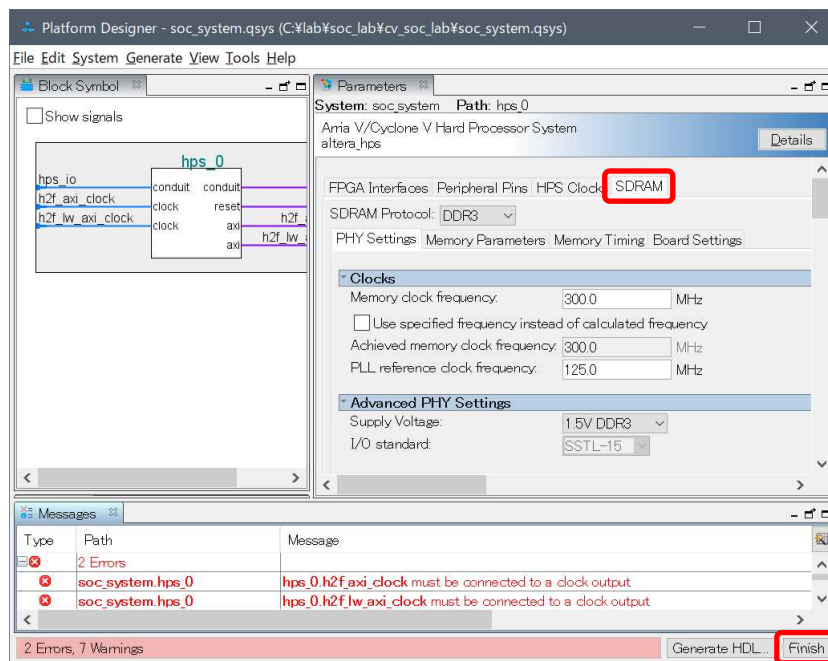


図 3-27. Parameters ウィンドウ表示の準備

2. **System Contents** ウィンドウの HPS コンポーネントをダブルクリックして再度 HPS のオプション設定を **Parameters** ウィンドウ内に表示させます。

これは、次の手順で **Preset** ウィンドウを表示するために必要な操作です。

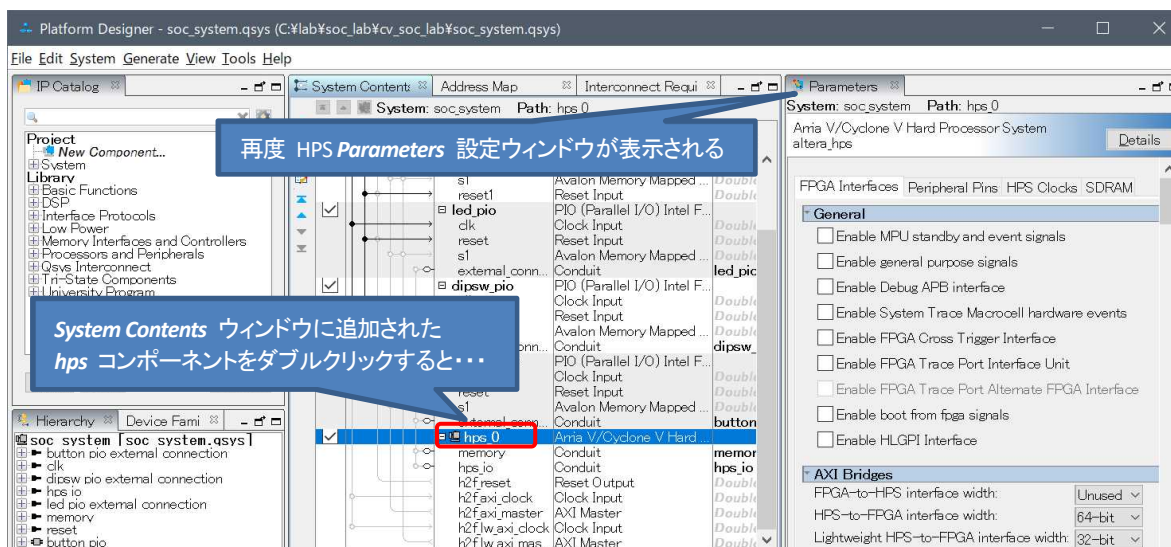


図 3-28. HPS パラメーター設定ウィンドウを再表示

3. **Parameters** ウィンドウの **SDRAM** タブをクリックします。

今回はあらかじめ用意されている Atlas-SoC ボード に載っている SDRAM の Preset を使用します。
Presets ウィンドウが表示されていることを確認します。

① **Note:**

Preset ウィンドウが表示されていない場合は、Platform Designer の **View** メニュー ⇒ **Presets** を選択し表示させてください。

それでも表示されない場合は、Platform Designer の **View** メニュー ⇒ **Reset to System Layout** を選択後、再度 **Preset** を選択してみてください。

4. **Presets** ウィンドウの **Atlas_HPS_SDRAM** プリセットを選択します。

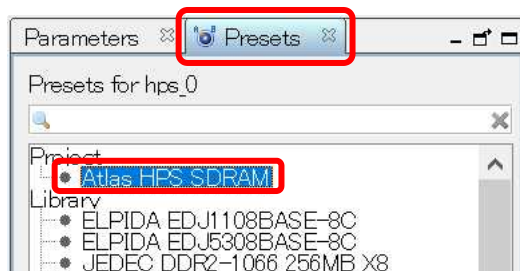


図 3-29. プリセットの選択

5. **Apply** をクリックすると、**Atlas_HPS_SDRAM** が太字でハイライトされるはずです。この状態になっていれば設定が正しく反映されています。

6. **SDRAM** タブが表示されていない場合は、**SDRAM** タブをクリックします。

7. **PHY Settings** タブをクリックし、下図の設定となっていることを確認します。

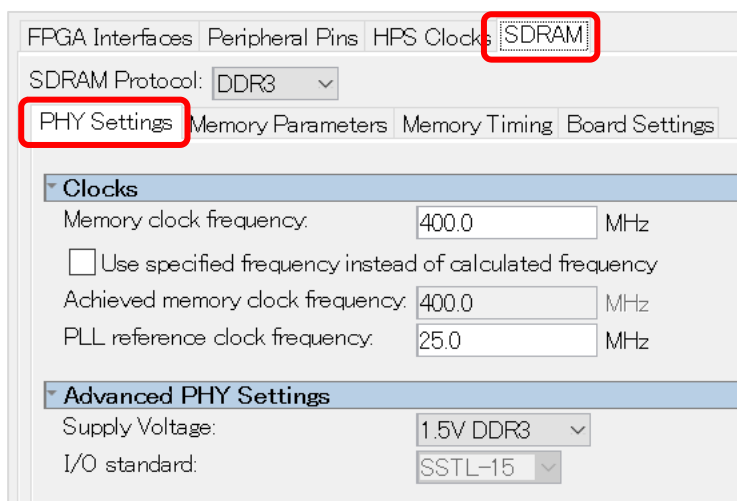
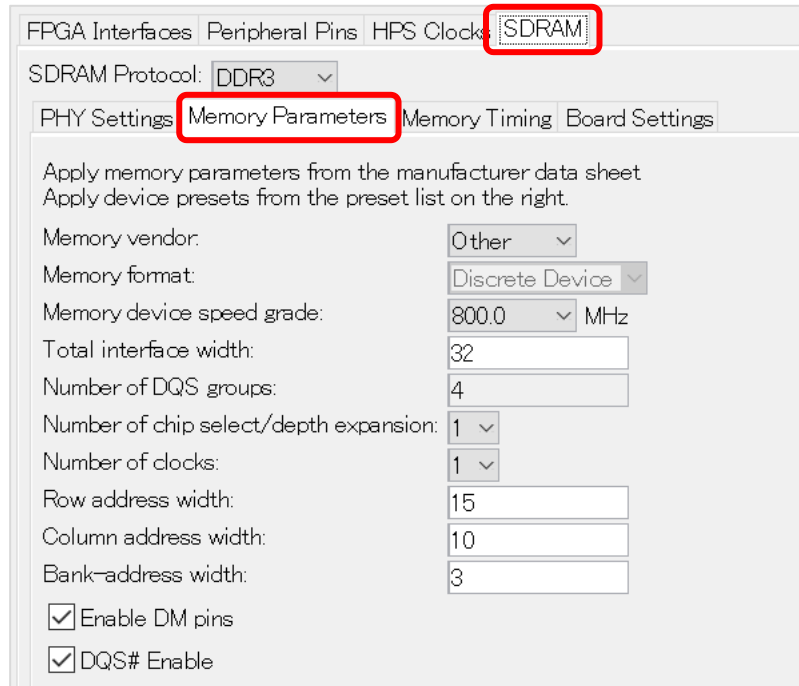


図 3-30. PHY Settings の確認

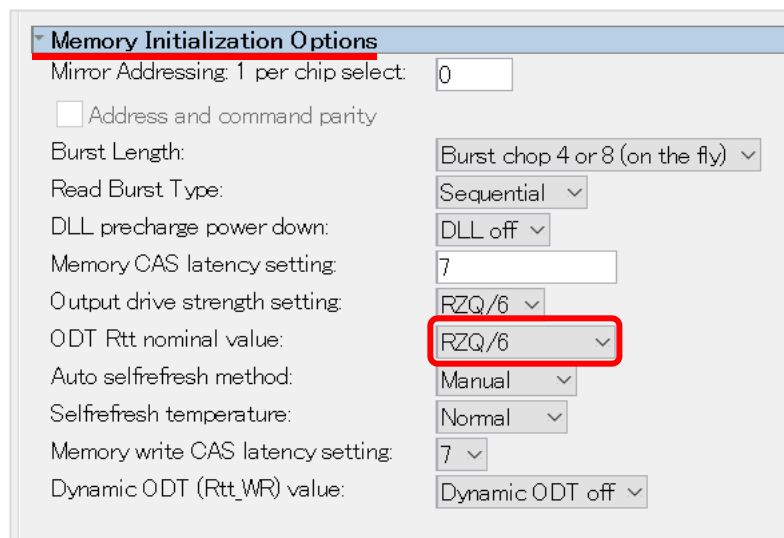
8. **Memory Parameters** タブをクリックし、下図の設定となっていることを確認します。



The screenshot shows the 'SDRAM' configuration window with the 'Memory Parameters' tab selected. The 'SDRAM Protocol' is set to 'DDR3'. The 'Memory vendor' is 'Other', 'Memory format' is 'Discrete Device', and 'Memory device speed grade' is '800.0 MHz'. The 'Total interface width' is 32, 'Number of DQS groups' is 4, 'Number of chip select/depth expansion' is 1, 'Number of clocks' is 1, 'Row address width' is 15, 'Column address width' is 10, and 'Bank-address width' is 3. The 'Enable DM pins' and 'DQS# Enable' checkboxes are checked.

図 3-31. Memory Parameters

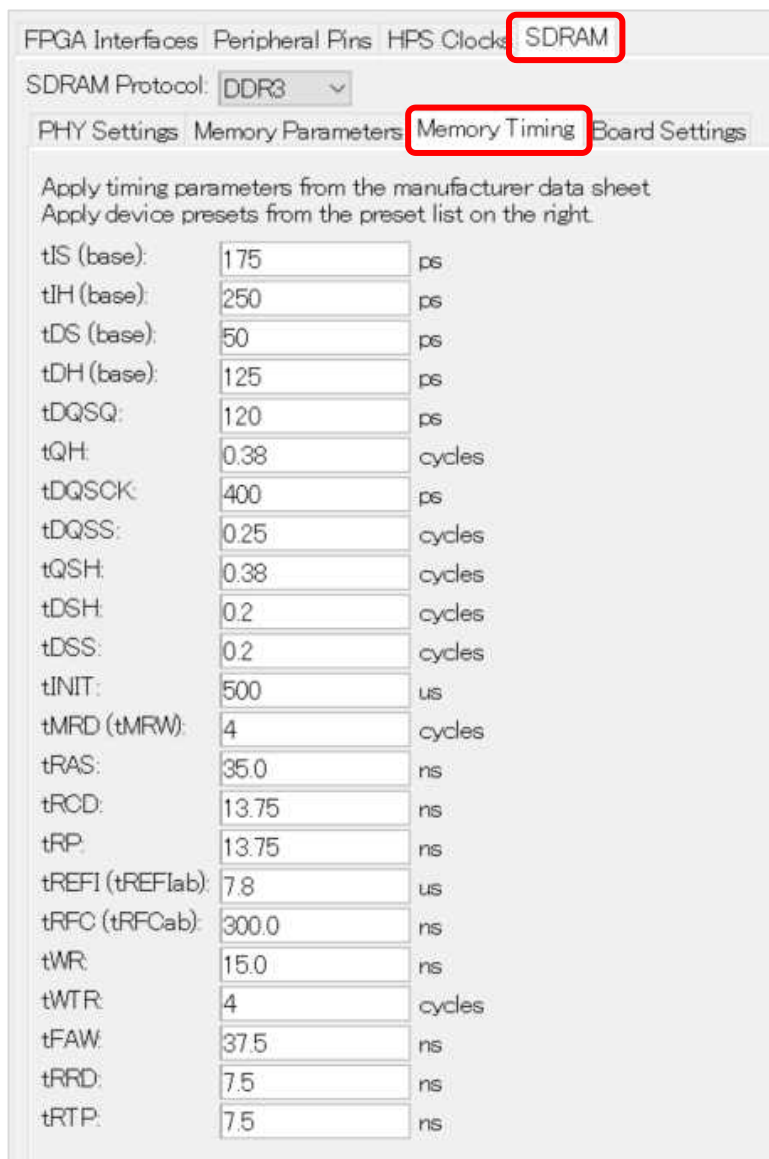
9. **Memory Initialization Options** セクションまでスクロールダウンし、**ODT Rtt nominal value** に **RZQ/6** が設定されていることを確認します。



The screenshot shows the 'Memory Initialization Options' section. The 'Mirror Addressing: 1 per chip select' is set to 0. The 'Address and command parity' checkbox is unchecked. The 'Burst Length' is 'Burst chop 4 or 8 (on the fly)', 'Read Burst Type' is 'Sequential', 'DLL precharge power down' is 'DLL off', 'Memory CAS latency setting' is 7, 'Output drive strength setting' is 'RZQ/6', 'ODT Rtt nominal value' is 'RZQ/6', 'Auto selfrefresh method' is 'Manual', 'Selfrefresh temperature' is 'Normal', 'Memory write CAS latency setting' is 7, and 'Dynamic ODT (Rtt_WR) value' is 'Dynamic ODT off'.

図 3-32. Memory Initialization Options

10. **Memory Timing** タブをクリックし、下図の通りの設定となっていることを確認します。



FPGA Interfaces Peripheral Pins HPS Clocks **SDRAM**

SDRAM Protocol: **DDR3**

PHY Settings Memory Parameters **Memory Timing** Board Settings

Apply timing parameters from the manufacturer data sheet
Apply device presets from the preset list on the right.

tIS (base):	175	ps
tIH (base):	250	ps
tDS (base):	50	ps
tDH (base):	125	ps
tDQSQ:	120	ps
tQH:	0.38	cycles
tDQSCK:	400	ps
tDQSS:	0.25	cycles
tQSH:	0.38	cycles
tDSH:	0.2	cycles
tDSS:	0.2	cycles
tINIT:	500	us
tMRD (tMRW):	4	cycles
tRAS:	35.0	ns
tRCD:	13.75	ns
tRP:	13.75	ns
tREFI (tREFIab):	7.8	us
tRFC (tRFCab):	300.0	ns
tWR:	15.0	ns
tWTR:	4	cycles
tFAW:	37.5	ns
tRRD:	7.5	ns
tRTP:	7.5	ns

図 3-33. Memory Timing

11. **Board Settings** タブをクリックし、**Setup and Hold Derating** セクションおよび **Channel Signal Integrity** セクションで、**Use Altera's default settings** が選択されていることを確認します。

FPGA Interfaces Peripheral Pins HPS Clocks **SDRAM**

SDRAM Protocol: **DDR3**

PHY Settings Memory Parameters Memory Timing **Board Settings**

Use the Board Settings to model the board-level effects in the timing analysis.

The wizard supports single- and multi-rank configurations. Altera has determined the effects on the output signaling of these configurations and has stored the effects on the output slew rate and the channel uncertainty within the UniPHY MegaWizard.

These values are representative of specific Altera boards. You must change the values to account for the board level effects for your board. You can use HyperLynx or similar simulators to obtain values that are representative of your board.

Setup and Hold Derating

The slew rate of the output signals affects the setup and hold times of the memory device.

You can specify the slew rate of the output signals to refer to their effect on the setup and hold times of both the address and command signals and the DQ signals, or specify the setup and hold times directly.

Derating method:

- ☒ Use Altera's default settings
- ☐ Specify slew rates to calculate setup and hold times
- ☐ Specify setup and hold times directly

CK/CK# slew rate (Differential):	2.0	V/ns
Address and command slew rate:	1.0	V/ns
DQS/DQS# slew rate (Differential):	2.0	V/ns
DQ slew rate:	1.0	V/ns
tIS:	0.325	ns
tIH:	0.35	ns
tDS:	0.2	ns
tDH:	0.225	ns

Channel Signal Integrity

Channel Signal Integrity is a measure of the distortion of the eye due to intersymbol interference or crosstalk or other effects. Typically when going from a single-rank configuration to a multi-rank configuration there is an increase in the channel loss as there are multiple stubs causing reflections. Please perform your channel signal integrity simulations and enter the extra channel uncertainty as compared to Altera's reference eye diagram.

Derating Method:

- ☒ Use Altera's default settings
- ☐ Specify channel uncertainty values

Address and command eye reduction (setup): 0.0 ns

図 3-34. Board Settings (1)

12. **Board Skew** セクションまでスクロールダウンし、ボードスキューが下図の通りであることを確認します。

Channel Signal Integrity

Channel Signal Integrity is a measure of the distortion of the eye due to intersymbol interference or crosstalk or other effects. Typically when going from a single-rank configuration to a multi-rank configuration there is an increase in the channel loss as there are multiple stubs causing reflections. Please perform your channel signal integrity simulations and enter the extra channel uncertainty as compared to Altera's reference eye diagram.

Derating Method:

☒ Use Altera's default settings
☐ Specify channel uncertainty values

Address and command eye reduction (setup):	0.0	ns
Address and command eye reduction (hold):	0.0	ns
Write DQ eye reduction:	0.0	ns
Write Delta DQS arrival time:	0.0	ns
Read DQ eye reduction:	0.0	ns
Read Delta DQS arrival time:	0.0	ns

Board Skews

PCB traces can have skews between them that can cause timing margins to be reduced. Furthermore skews between different ranks can further reduce the timing margin in multi-rank topologies.

Restore default values

Maximum CK delay to DIMM/device:	0.6	ns
Maximum DQS delay to DIMM/device:	0.6	ns
Minimum delay difference between CK and DQS:	-0.01	ns
Maximum delay difference between CK and DQS:	0.01	ns
Maximum skew within DQS group:	0.02	ns
Maximum skew between DQS groups:	0.02	ns
Average delay difference between DQ and DQS:	0.0	ns
Maximum skew within address and command bus:	0.02	ns
Average delay difference between address and command and CK:	0.0	ns

図 3-35. Board Settings (2)

13. Platform Designer の **File** メニュー ⇒ **Save** を選択し、ここまでの手順で指定した HPS のパラメーター設定内容を保存します。

3-6. ステップ 6 : HPS のクロックとエクスポート信号の設定

このステップでは、HPS の H2F ブリッジのクロックと LWH2F ブリッジのクロックの設定を行います。

ここで設定するクロックは、各ブリッジの FPGA 側のクロック（下図の h2f_axi_clk と h2f_lw_axi_clk）です。HPS 側のクロックは「[3-4. ステップ 4 : HPS クロックの設定](#)」で設定した l3_main_clk や l4_mp_clk となり、これから設定するクロックとは異なります。クロックの違いは、ブリッジ内で吸収されます。

また、HPS のエクスポート信号の設定も行います。このエクスポート信号は Platform Designer 外部との通信に使用されます。たとえば、Platform Designer と FPGA の他のロジックとの接続やピンへの配置に使用されます。

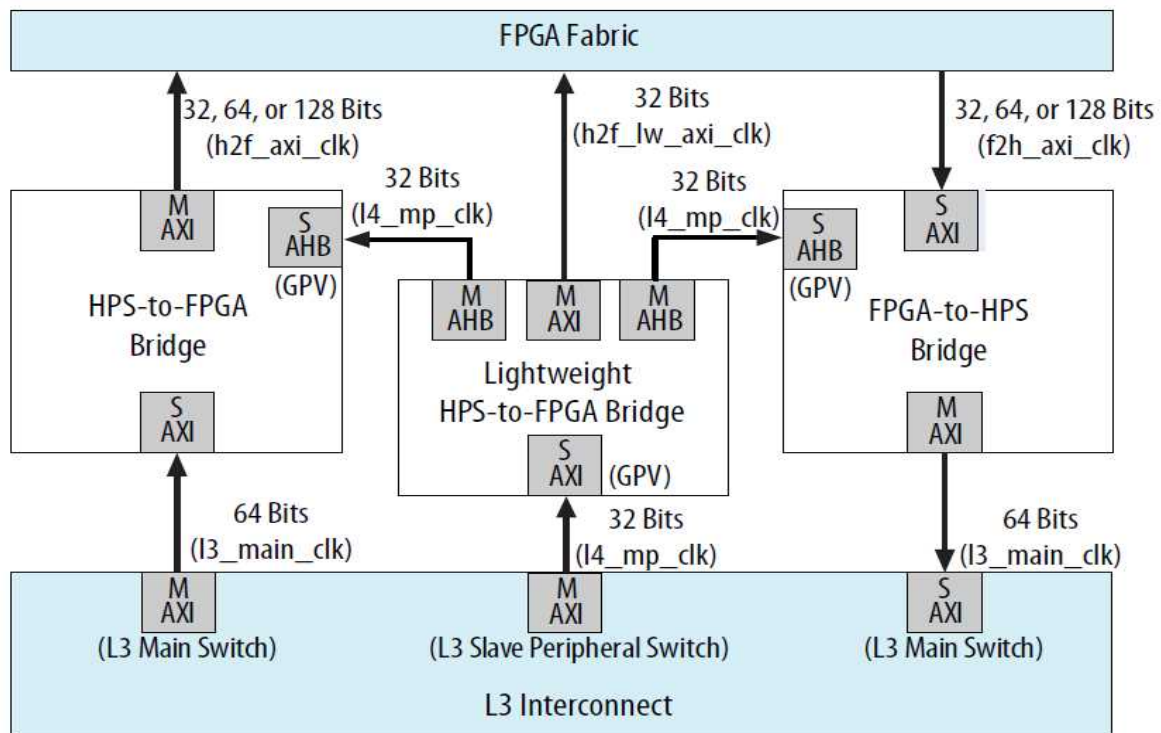


図 3-36. HPS と FPGA 間のクロック

1. **System Contents** タブに移動します。
2. **Export** 列に信号名を記述することで、Platform Designer システムの外部に信号線を出すことができます。先ほど追加した HPS コンポーネントの **hps_io** ポートが **hps_io** という信号名でエクスポートされていることを確認します。
3. 同様に HPS コンポーネントの **memory** ポートが **memory** という信号名でエクスポートされていることを確認します。こちらは先ほど設定した HPS 側の SDRAM の IO です。
4. HPS コンポーネントの **h2f_reset** をエクスポートします。**H2f_reset** の **Export** 列をダブルクリックし、“**h2f_reset**” にリネームした後、Enter キーを押してエクスポートします。
5. HPS 上の Clock Input インターフェイス **h2f_axi_clock** の設定を行います。**H2f_axi_clock** の横の **Clock** 列のプルダウンメニューで **clk_0** を選択し、**h2f_axi_clock** に **clk_0** を接続します。
6. 同様に HPS 上の Clock Input インターフェイス **h2f_lw_axi_clock** の設定を行います。**H2f_lw_axi_clock** の横の **Clock** 列のプルダウンメニューで **clk_0** を選択し、**h2f_lw_axi_clock** に **clk_0** を接続します。

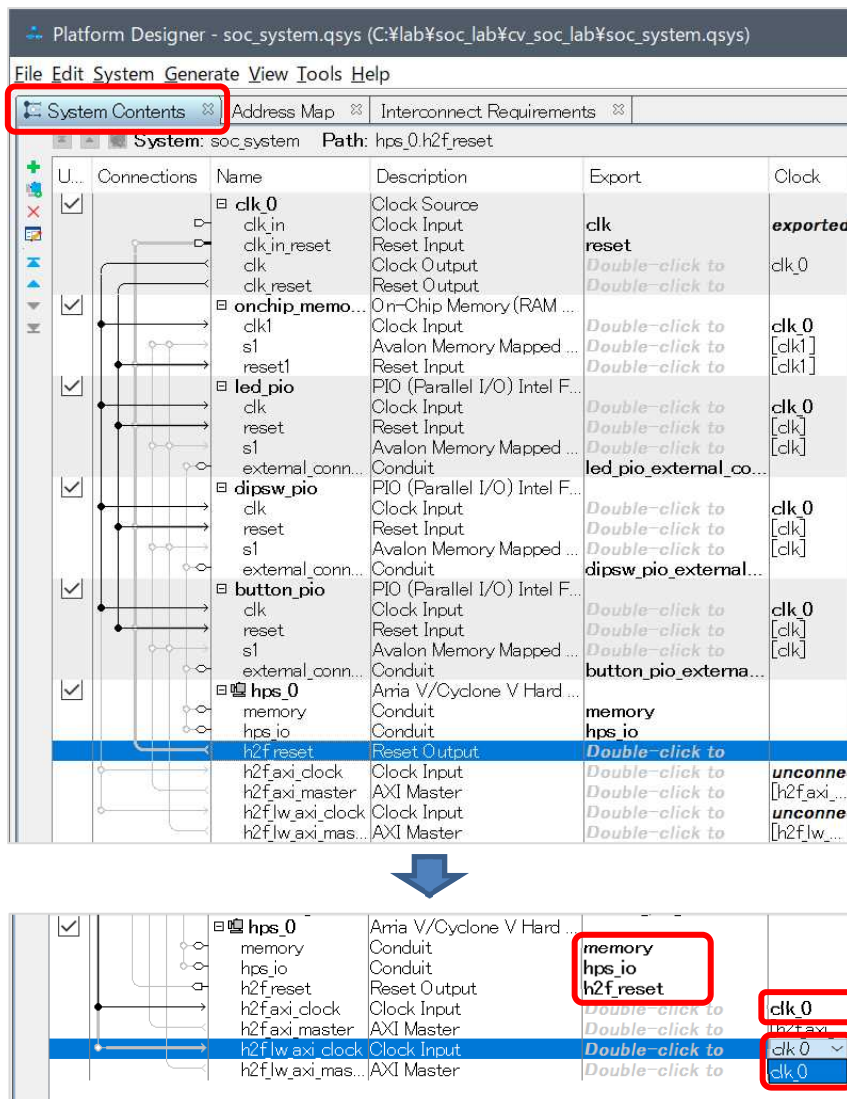


図 3-37. クロックとエクスポート信号の設定

この時点で **clk_0** が接続されたので、図 3-23 において **Message Window** に出ていた2つのエラーメッセージがなくなっているはずで

3-7. ステップ 7 : HPS コンポーネントと他のコンポーネントの接続

このステップでは、Platform Designer システムに追加した HPS コンポーネントと Platform Designer システムに実装済みのコンポーネントを接続します。今回 FPGA 側は `clk_0` (50MHz) で動作させるため、あらかじめ `clk_0` が各コンポーネントに接続されています。

- ___ 1. **Onchip_memory2_0** コンポーネントの Clock Input インターフェイスが、**clk_0** に接続されていることを確認します。
- ___ 2. **Led_pio** コンポーネントの Clock Input インターフェイスが、**clk_0** に接続されていることを確認します。
- ___ 3. **Dipsw_pio** コンポーネントの Clock Input インターフェイスが、**clk_0** に接続されていることを確認します。
- ___ 4. **Button_pio** コンポーネントの Clock Input インターフェイスが、**clk_0** に接続されていることを確認します。
- ___ 5. **Onchip_memory2_0** の **s1** を選択した後、右クリックをすることにより表示される接続サブメニューから **hps_0.h2f_axi_master** を選択します。これにより HPS **h2f_axi_master** に、**onchip_memory2_0** コンポーネントの **s1** インターフェイスが接続されます。この設定で Arm® プロセッサ から FPGA 側の **onchip_memory** へアクセスすることができます。

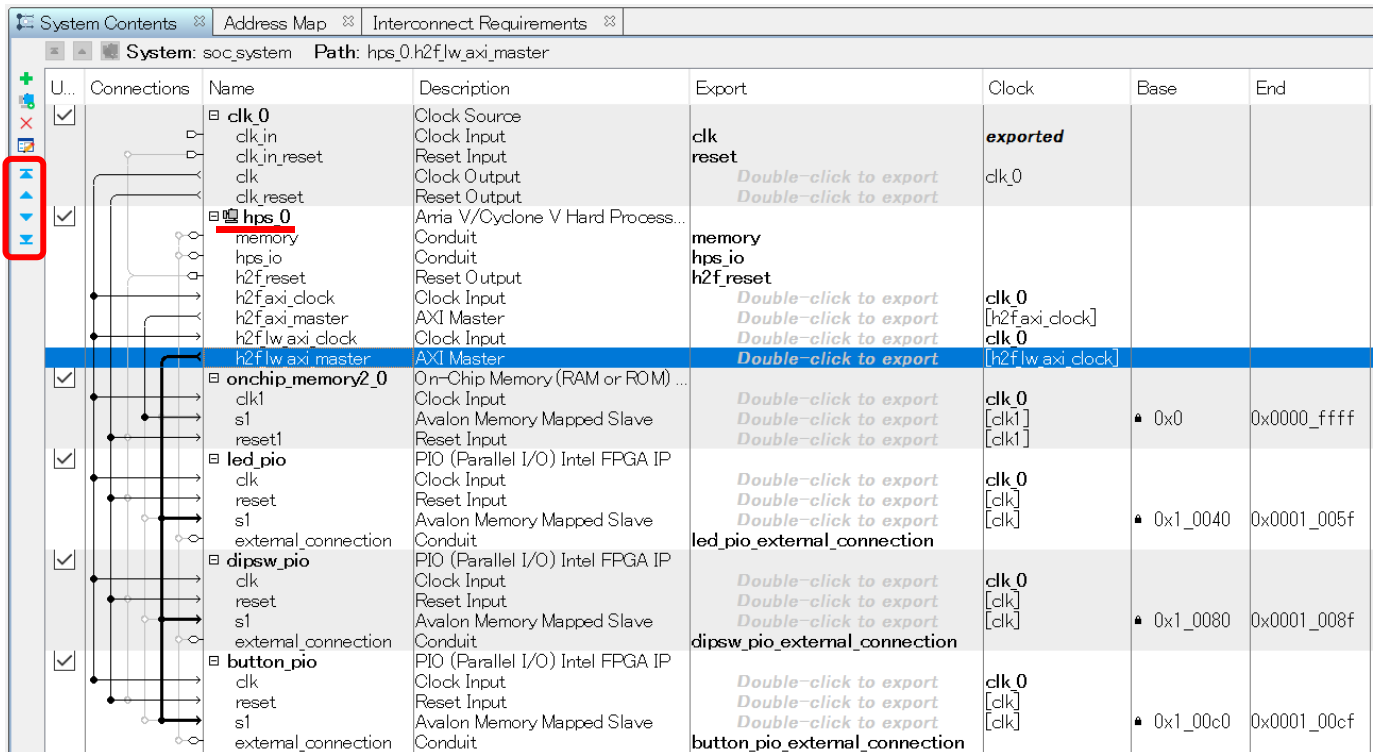


図 3-38. コンポーネント間の接続

- ___ 6. 同様に **button_pio** の **s1** を右クリックし、接続サブメニューから **hps_0.h2f_lw_axi_master** を選択します。これにより HPS **h2f_lw_axi_master** に、**button_pio** コンポーネントの **s1** インターフェイスが接続されます。
接続先が **h2f_lw_axi_master** であることに注意してください。続く各 PIO コンポーネントも同様です。
- ___ 7. 同様に **dipsw_pio** の **s1** を右クリックし、接続サブメニューから **hps_0.h2f_lw_axi_master** を選択します。これにより HPS **h2f_lw_axi_master** に、**dipsw_pio** コンポーネントの **s1** インターフェイスが接続されます。
- ___ 8. 同様に **led_pio** の **s1** を右クリックし、接続サブメニューから **hps_0.h2f_lw_axi_master** を選択します。これにより HPS **h2f_lw_axi_master** に、**led_pio** コンポーネントの **s1** インターフェイスが接続されます。

9. HPS コンポーネントを選択し **System Contents** ウィンドウの左側にある、Platform Designer ツールバーの上下ボタンを  使用して、HPS コンポーネントを **clk_0** の下に移動してください。

設定完了後の Platform Designer システムは以下の通りです。



U...	Connections	Name	Description	Export	Clock	Base	End
☒		clk_0	Clock Source	clk	exported		
☒		clk_in	Clock Input	reset	clk_0		
☒		clk_in_reset	Reset Input	Double-click to export	Double-click to export		
☒		clk	Clock Output	Double-click to export	Double-click to export		
☒		clk_reset	Reset Output	Double-click to export	Double-click to export		
☒		hps_0	Arria V/Cyclone V Hard Process...	memory	clk_0		
☒		memory	Conduit	hps_io	[h2f_axi_clock]		
☒		hps_io	Conduit	h2f_reset	clk_0		
☒		h2f_reset	Reset Output	Double-click to export	Double-click to export		
☒		h2f_axi_clock	Clock Input	Double-click to export	Double-click to export		
☒		h2f_axi_master	AXI Master	Double-click to export	Double-click to export		
☒		h2f_lw_axi_clock	AXI Master	Double-click to export	Double-click to export		
☒		onchip_memory2_0	On-Chip Memory (RAM or ROM) ...	clk_0			
☒		clk1	Clock Input	Double-click to export	Double-click to export		
☒		Avalon Memory Mapped Slave	Reset Input	Double-click to export	Double-click to export		
☒		reset1	Reset Input	Double-click to export	Double-click to export		
☒		led_pio	PIO (Parallel I/O) Intel FPGA IP	clk_0			
☒		clk	Clock Input	Double-click to export	Double-click to export		
☒		reset	Reset Input	Double-click to export	Double-click to export		
☒		s1	Avalon Memory Mapped Slave	Double-click to export	Double-click to export		
☒		external_connection	Conduit	led_pio_external_connection	clk_0		
☒		dipsw_pio	PIO (Parallel I/O) Intel FPGA IP	clk_0			
☒		clk	Clock Input	Double-click to export	Double-click to export		
☒		reset	Reset Input	Double-click to export	Double-click to export		
☒		s1	Avalon Memory Mapped Slave	Double-click to export	Double-click to export		
☒		external_connection	Conduit	dipsw_pio_external_connection	clk_0		
☒		button_pio	PIO (Parallel I/O) Intel FPGA IP	clk_0			
☒		clk	Clock Input	Double-click to export	Double-click to export		
☒		reset	Reset Input	Double-click to export	Double-click to export		
☒		s1	Avalon Memory Mapped Slave	Double-click to export	Double-click to export		
☒		external_connection	Conduit	button_pio_external_connection	clk_0		

図 3-39. 設定完了後の Platform Designer システム

led_pio へのアクセスを考えてみます。

Led_pio の右から 2 つ目の **Base** 列を見ると 0x0001_0040 と設定されています。これが led_pio の Platform Designer でのオフセットアドレスです。先の 8. で設定したように led_pio にアクセスするマスターは HPS h2f_lw_axi_master です。Lightweight HPS-to-FPGA のブリッジのベースアドレスは 0xFF20_0000 でしたので、この led_pio にアクセスする場合は以下の値になります。

ブリッジのベースアドレス (0xFF20_0000) + Platform Designer のオフセットアドレス (0x0001_0040) = **0xFF21 0040**

ほかのコンポーネントも同様に考えることができ、dipsw_pio であれば **0xFF21 0080** です。

次に、onchip_memory へのアクセスを考えてみます。

HPS から FPGA に対してのもうひとつのパスである HPS h2f_axi_master ブリッジのベースアドレスは 0xC000_0000 でした。今回は HPS h2f_axi_master に接続した onchip_memory の Platform Designer のオフセットアドレスが 0x0 なので、この場合はブリッジのベースアドレス (0xC000_0000) がそのまま onchip_memory にアクセスするベースアドレスとなります。

3-8. ステップ 8 : リセットの接続とベースアドレスの割り当て

このステップでは、リセットの一括接続とベースアドレスの自動割り当てを行います。

1. Platform Designer の **System** メニュー ⇒ **Create Global Reset Network** を選択し、デザインのすべてのリセット・インターフェイスを一括で接続します。
2. 重複アドレスが存在しないように、すべてのコンポーネントのためにベースアドレスを自動割り当てします。
System メニュー ⇒ **Assign Base Addresses** を選択します。

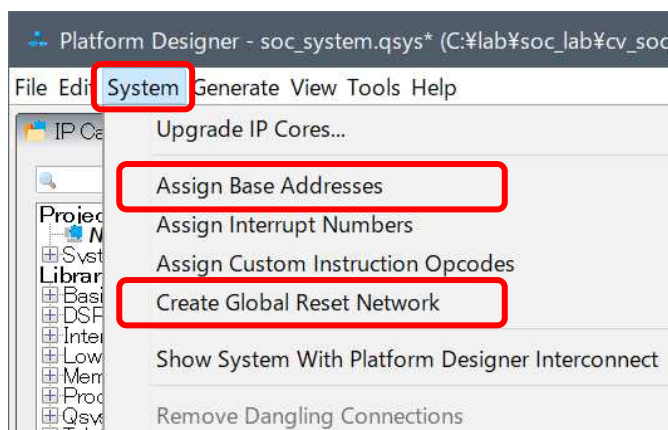


図 3-40. リセットの一括接続とベースアドレスの自動割り当て

Assign Base Address を行っても、何も起こらなかったのではないのでしょうか。

この演習では、事前に各ペリフェラルのベースアドレスを固定してあったため、自動的にアドレスがアサインされませんでした。

図 3-41 に示すように、ベースアドレスの横にある鍵マーク  を使用することにより、アドレス設定を固定できます。クリックするごとに固定されるかどうかトグルします。アドレスを固定したい場合は、アドレス設定後に鍵マークで固定してください。Platform Designer の **Edit** メニュー ⇒ **Lock Base Address** でも固定できます。

hps_0	memory	Arria V/Cyclone V Hard Process...	memory			
	hps_io	Conduit	hps_io			
	h2f_reset	Reset Output	h2f_reset			
	h2f_axi_clock	Clock Input		Double-click to export	clk_0	
	h2f_axi_master	AXI Master		Double-click to export	[h2f_axi_clock]	
	h2f_lw_axi_clock	Clock Input		Double-click to export	clk_0	
	h2f_lw_axi_master	AXI Master		Double-click to export	[h2f_lw_axi_clock]	
onchip_memory2_0	clk1	On-Chip Memory (RAM or ROM) ...		Double-click to export	clk_0	
	s1	Clock Input		Double-click to export	[clk1]	
	reset1	Avalon Memory Mapped Slave		Double-click to export	[clk1]	0x0 0x0000_ffff
led_pio	clk	PIO (Parallel I/O) Intel FPGA IP		Double-click to export	clk_0	
	reset	Clock Input		Double-click to export	[clk]	0x1_0040 0x0001_005f
	s1	Reset Input		Double-click to export	[clk]	
	external_connection	Avalon Memory Mapped Slave	led_pio_external_connection			
		Conduit				

図 3-41. ベースアドレスの固定

3-9. ステップ 9 : Platform Designer システムの確認

1. 設計した Platform Designer システムが以下の「[表 3-1. 設計後の Platform Designer システムの接続状況](#)」の通りであることを確認します。「[図 3-39. 設定完了後の Platform Designer システム](#)」も参考にしてください。

演習用の Quartus® Prime プロジェクトとの整合性を取るため、エクスポート信号が適切にエクスポートされていること、および正しく命名されていることを確認してください。実際の設計においては任意の信号名を利用いただくことが可能です。また、コンポーネントの順序に規定はありません。

表 3-1. 設計後の Platform Designer システムの接続状況

コンポーネント	ポート名	接続
clk_0	clk_in	clk としてエクスポート
	clk_in_reset	reset としてエクスポート
	clk	すべてのコンポーネントに接続
	clk_reset	hps_0 を除く、すべてのコンポーネントに接続
led_pio	external_connection	led_pio_external_connection としてエクスポート
dipsw_pio	external_connection	dipsw_pio_external_connection としてエクスポート
button_pio	external_connection	button_pio_external_connection としてエクスポート
hps_0	h2f_axi_master	onchip_memory2_0.s1 に接続
	h2f_lw_axi_master	led_pio.s1 に接続
		dipsw_pio.s1 に接続
		button_pio.s1 に接続

2. Platform Designer の **View** メニュー ⇒ **Device Family** を選択し、
Device Family が **Cyclone V** になっていること、
Device が Atlas-SoC ボードの場合は **5CSEMA4U23C6**、DE10-Nano ボードの場合は **5CSEBA6U23I7DK** になっていることを確認します。

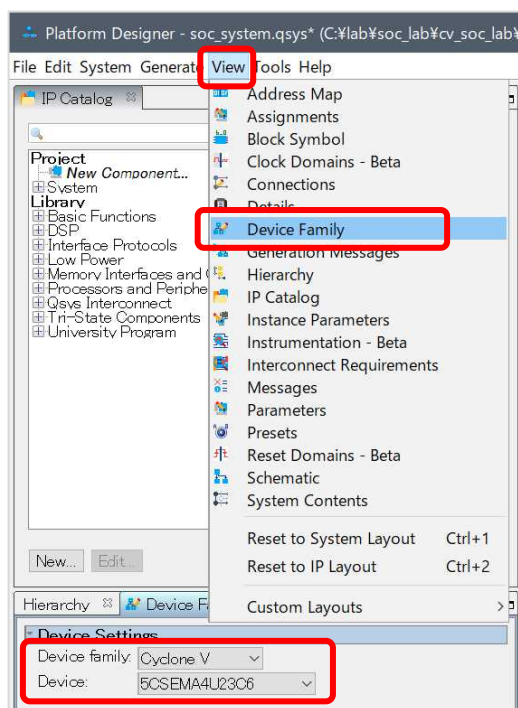


図 3-42. Device Family タブ

3. Platform Designer の **View** メニュー ⇒ **Interconnect Requirements** を選択し、
Limit interconnect pipeline stages to を **1** に設定します。段数を増やすとタイミングに余裕がでますが、同時に
FPGA のロジックも大きくなります。
Clock crossing adapter type が **Handshake** になっていることを確認します。

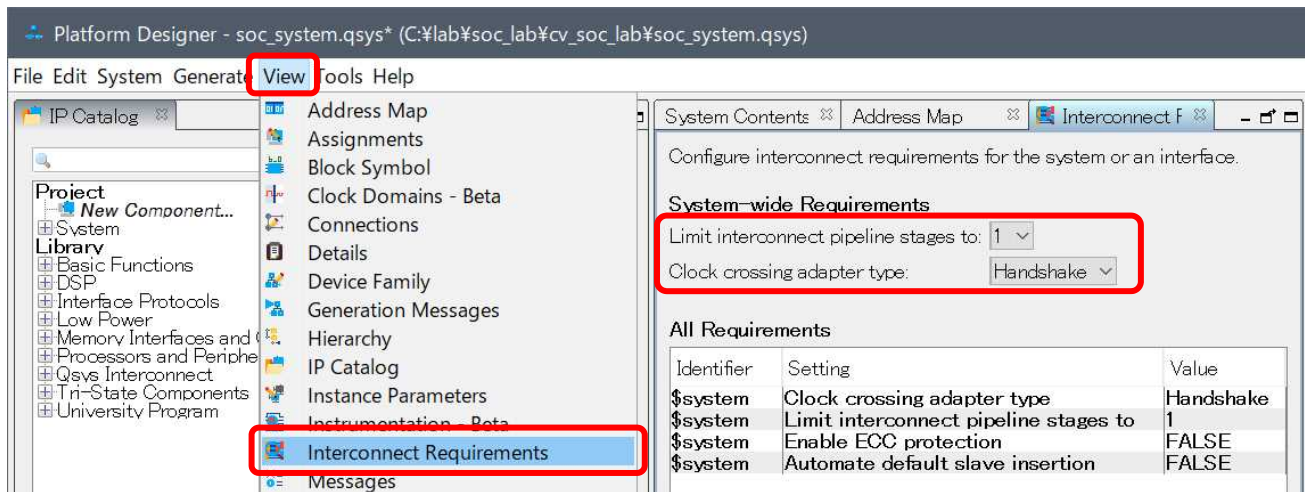
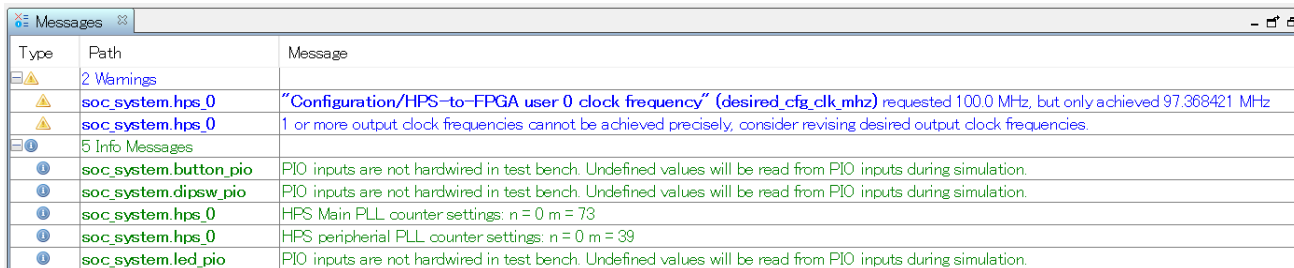


図 3-43. プロジェクト・パラメーターの設定

3-10. ステップ 10 : Platform Designer システムの生成

完成した Platform Designer システムを生成します。

1. **System Contents** タブの **Message** ボックスに、残りのエラーがあるかどうかを確認します。エラーがある場合は、続行する前にそれらを修正する必要があります。
青字で表示される **Warning** に関しては、今回は無視してください。



Type	Path	Message
2 Warnings		
Warning	soc_system.hps_0	"Configuration/HPS-to-FPGA user 0 clock frequency" (desired_cfg_clk_mhz) requested 100.0 MHz, but only achieved 97.368421 MHz
Warning	soc_system.hps_0	1 or more output clock frequencies cannot be achieved precisely, consider revising desired output clock frequencies.
5 Info Messages		
Info	soc_system.button_pio	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Info	soc_system.dipsw_pio	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Info	soc_system.hps_0	HPS Main PLL counter settings: n = 0 m = 73
Info	soc_system.hps_0	HPS peripheral PLL counter settings: n = 0 m = 39
Info	soc_system.led_pio	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.

図 3-44. Message ウィンドウの表示

2. **File** メニュー ⇒ **Save** を選択して、Platform Designer システムを保存します。**Save System Completed** がポップアップされたら **[Close]** します。

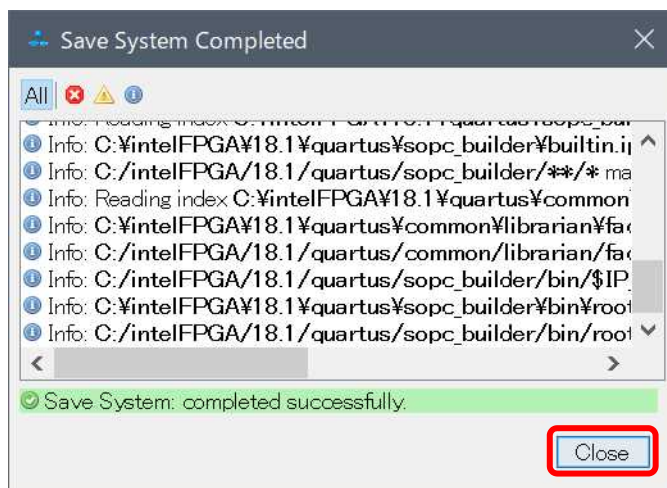


図 3-45. Platform Designer システムの保存

3. **Generate** メニュー ⇒ **Generate HDL** を選択します。

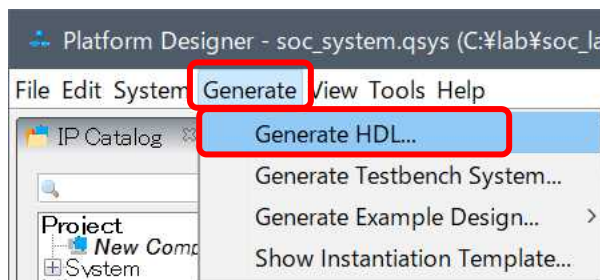


図 3-46. Platform Designer システムの生成

4. **Generation** ウィンドウの設定を確認し、[**Generate**] を実行します。

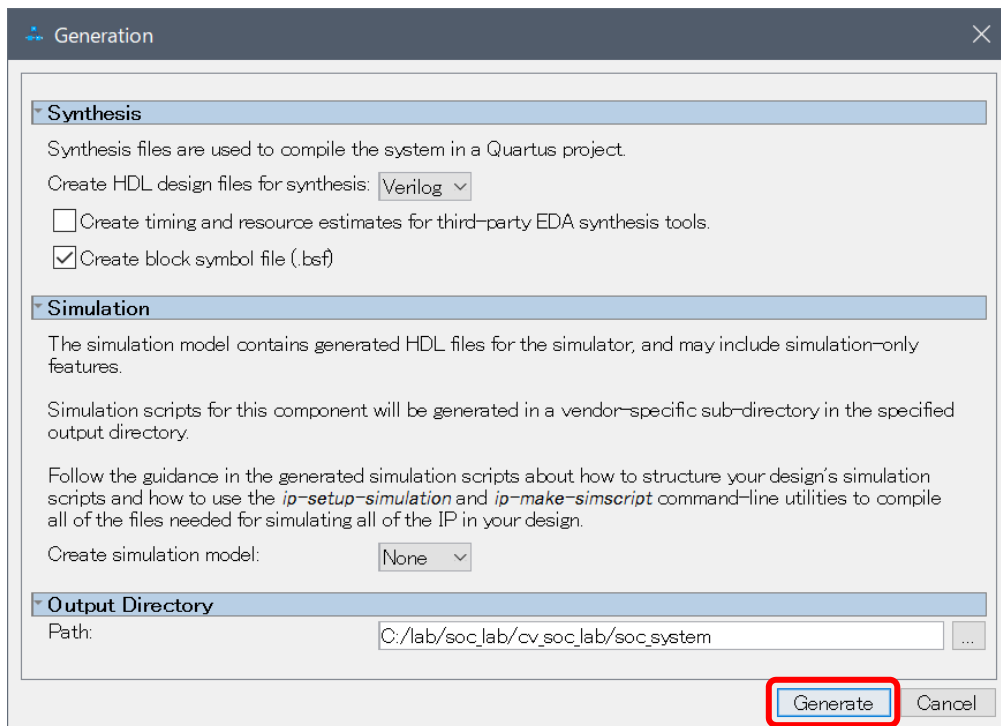


図 3-47. Platform Designer システムの Generate 実行画面

5. Platform Designer の **Generate** メニュー ⇒ **Show Instantiation Template** ではトップデザインにインスタンスする際に使用できるインスタンスの例が表示されます。

今回はインスタンス済みですので、特に作業は発生しませんが実際に使用する場合にはとても有効です。

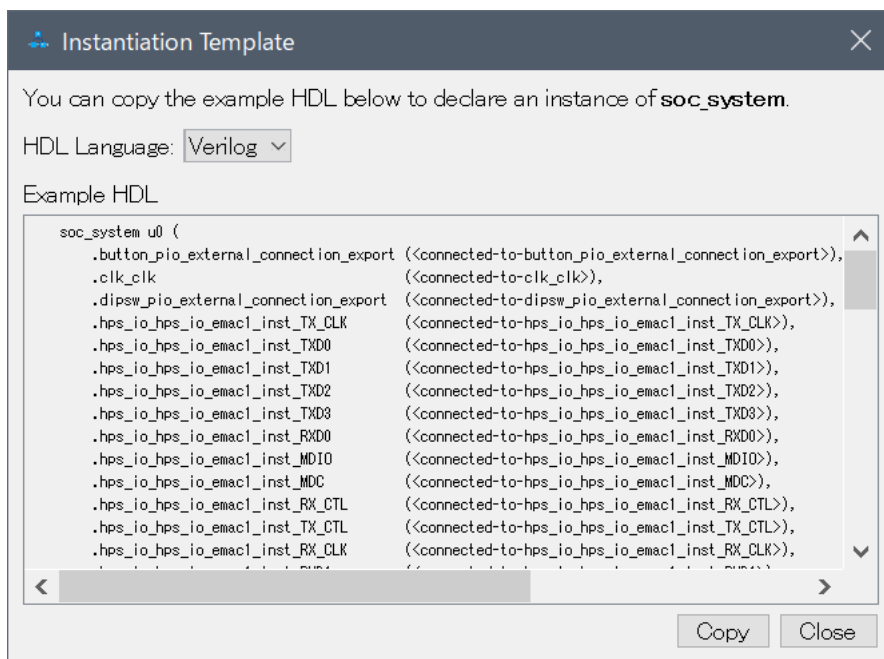


図 3-48. Platform Designer システムのインスタンス例

6. Platform Designer の生成が完了した後に、[Close] ボタンをクリックし、Platform Designer のシステム生成ダイアログボックスを閉じて Quartus® Prime に戻ります。

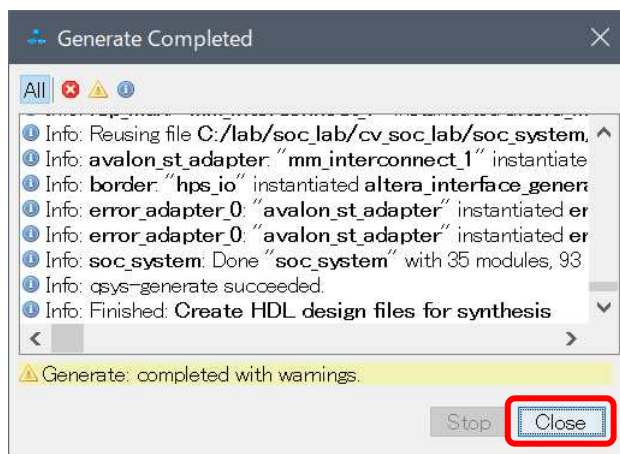


図 3-49. Platform Designer システムの生成完了

7. Quartus® Prime の **Project** メニュー ⇒ **Add/Remove Files in Project** を選択します (**Settings** ダイアログボックスが **Files** カテゴリが選択された状態で表示されます)。

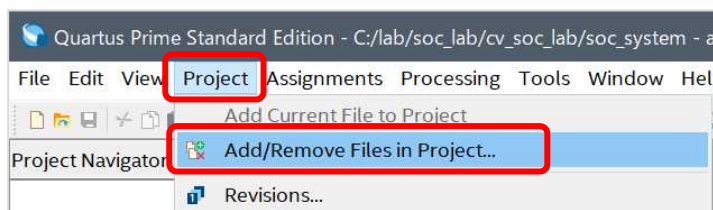


図 3-50. Add/Remove Files in Project を選択

8. **Settings** ダイアログボックス内の **File name** フィールドの横にある **...** ボタンを押し、**Select File** ウィンドウから **soc_system/synthesis** フォルダを参照します。

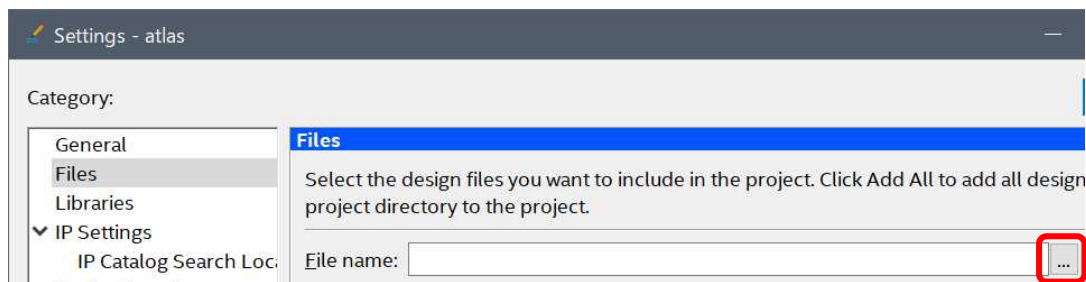


図 3-51. Settings ダイアログボックス

- ___ 9. **Soc_system.qip** ファイルを選択し、**[開く (O)]** をクリックします。この qip ファイルは Platform Designer で Generate したコンポーネントを紐づけているファイルです。Generate したファイルをひとつずつ登録するのではなく、こちらの qip ファイルの登録のみで、Platform Designer システムをプロジェクトに追加することができます。

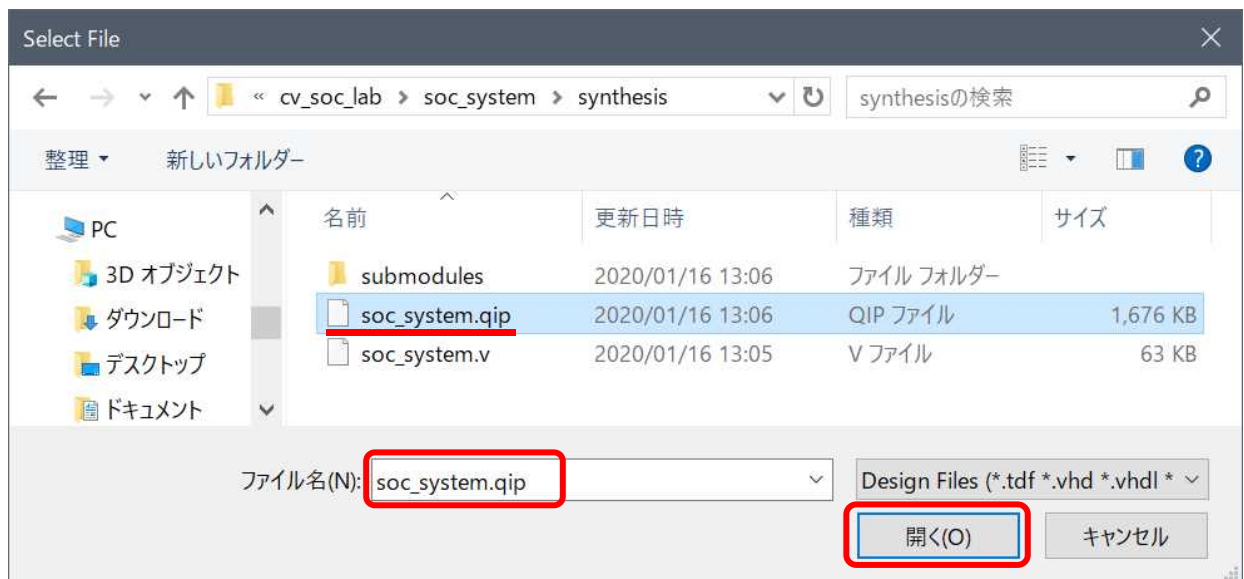


図 3-52. Qip ファイルの指定

- ___ 10. ファイルが追加されたことを確認します。

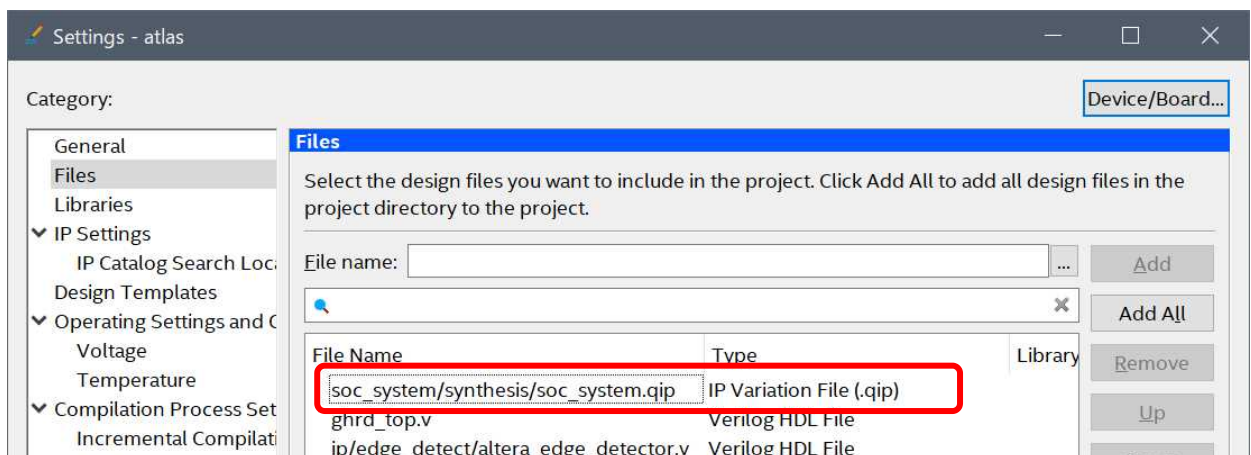


図 3-53. Qip ファイルの登録

- ___ 11. **Settings** ダイアログボックスを **[OK]** ボタンで閉じます。

3-11. ステップ 11 : ピン・アサインメントの設定と Quartus® Prime プロジェクトのコンパイル

HPS 専用 IO に関しては、ピン配置が決まっているため基本的にピン・アサインメントはツールが自動で行います。例外として、SDRAM インターフェイス は、ツールが生成したスクリプトを設計者が実行する必要があります。スクリプトを実行するためには、まずネットリストを生成し、その後にスクリプトを実行します。

そのため、まずはネットリスト作成のための Analysis & Synthesis を実行後、スクリプトを実行し、再度 FPGA のコンパイルを行います。

____ 1. Quartus® Prime の **Processing** メニュー ⇒ **Start** ⇒ **Start Analysis & Synthesis** を選択します。

(または、GUI 上の Start Analysis & Synthesis ボタン  をクリックします)。

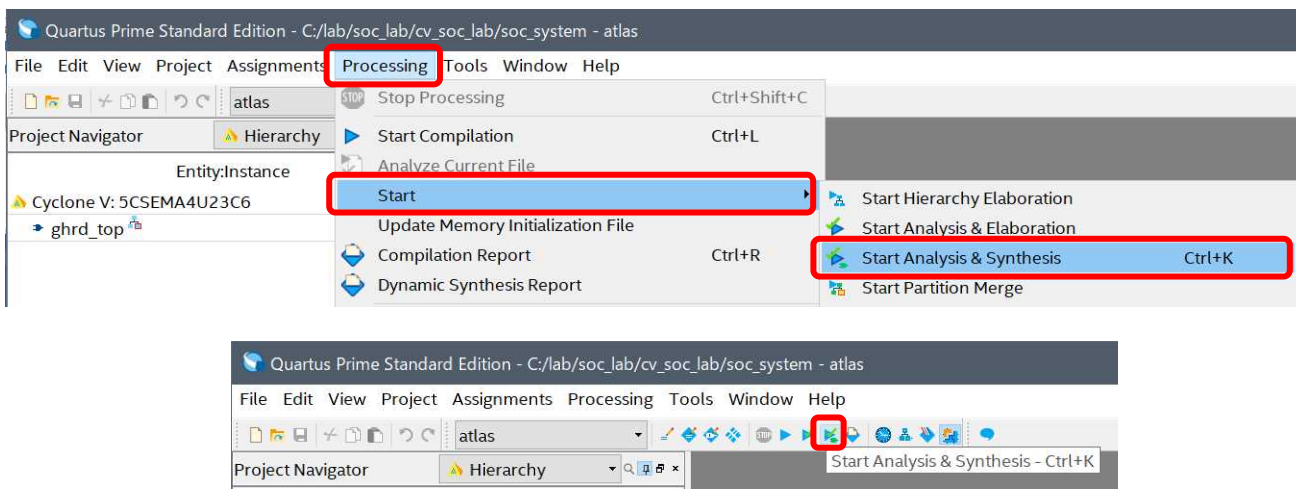


図 3-54. Start Analysis & Synthesis

____ 2. 終了後、エラーがないことを確認します。  があれば、エラーは出ていません。これでネットリストが作成されました。

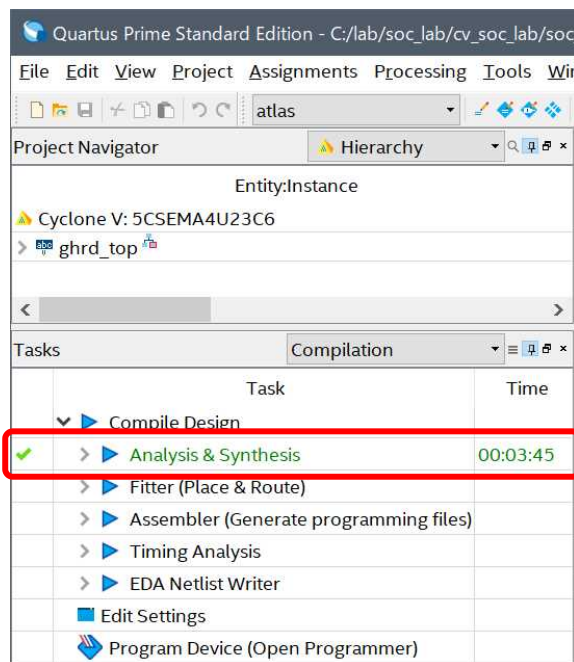


図 3-55. Start Analysis & Synthesis の正常終了確認

3. Quartus® Prime の **Tools** メニュー ⇒ **TCL scripts** を選択します。

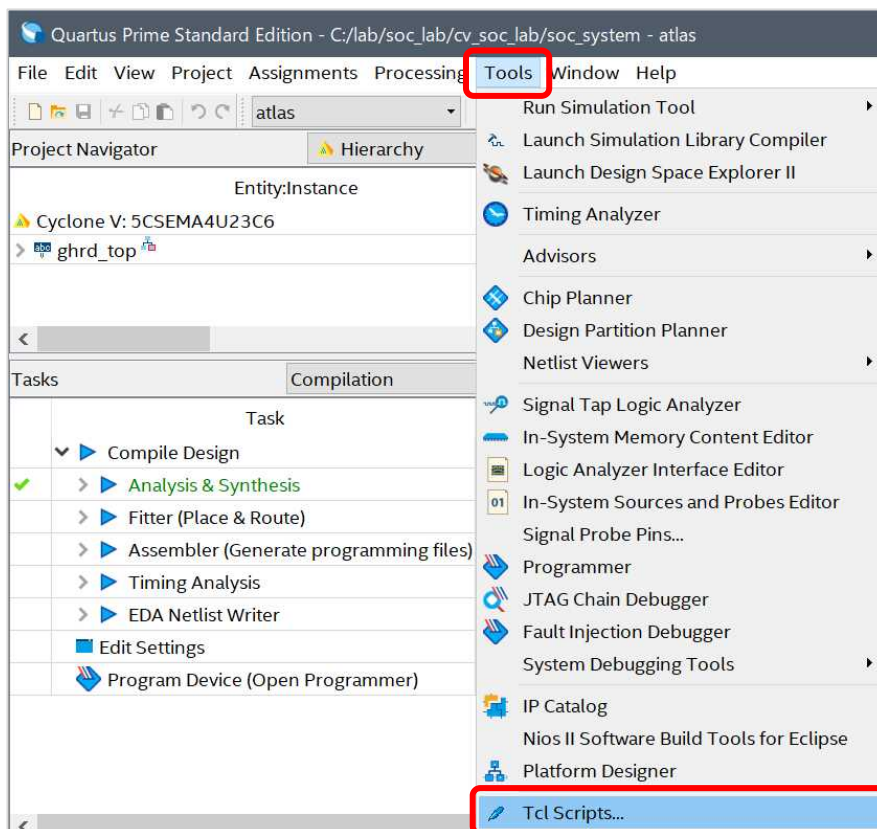


図 3-56. Tcl Scripts ウィンドウの起動

4. **Soc_system** ⇒ **synthesis** ⇒ **submodules** にある **hps_sdram_p0_pin_assignments.tcl** を選択し、[Run] ボタンをクリックします（反映されるまで少し時間がかかります）。

この作業により SDRAM の IO Standard の設定や OCT の設定など HPS の SDRAM Controller タブで設定した内容が反映されます。

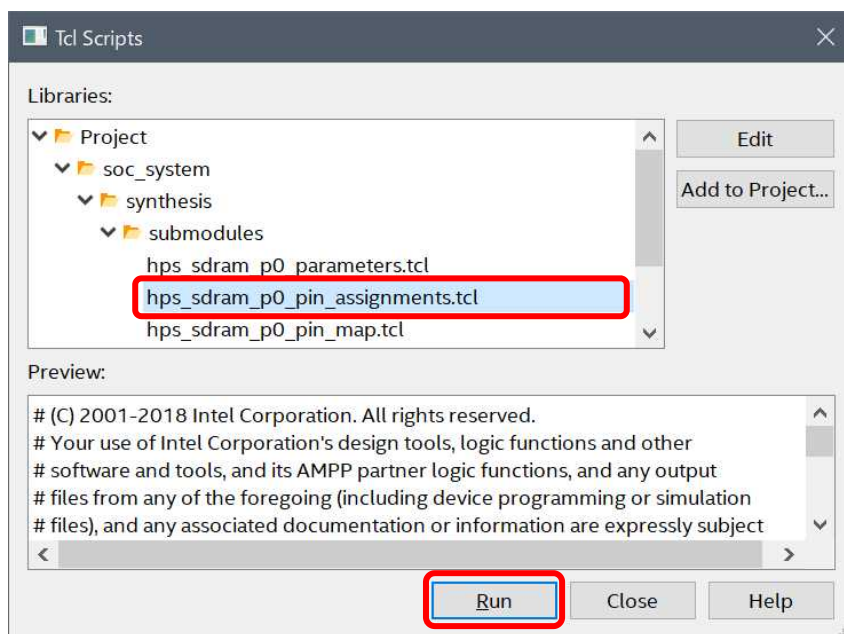


図 3-57. TclScript の実行

___ 5. 完了したら [OK] ボタンをクリックします。

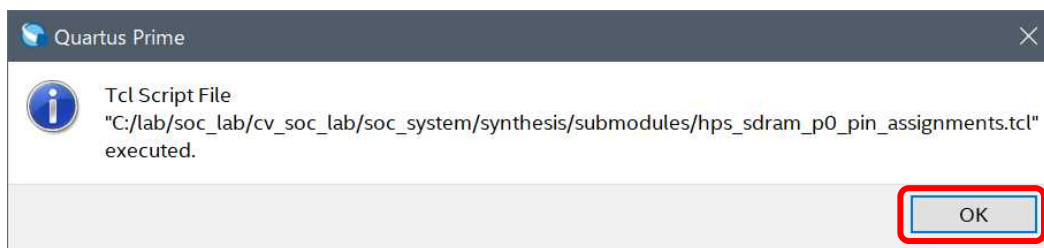


図 3-58. TclScript の完了

___ 6. Tcl Scripts ウィンドウを [Close] します。

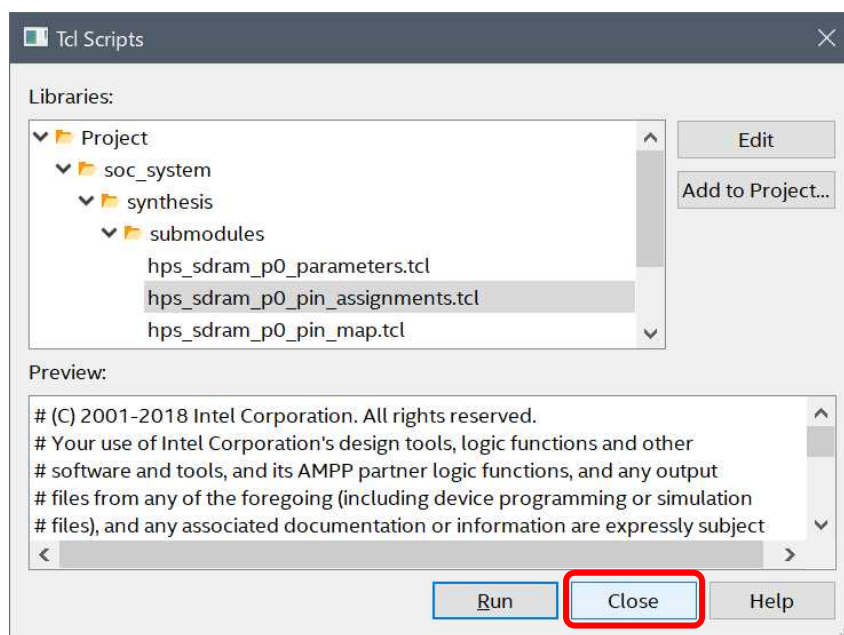


図 3-59. Tcl Scripts ウィンドウの Close

7. Quartus® Prime の **Processing** メニュー ⇒ **Start Compilation** を選択（または、GUI 上の Start Compilation ボタン  をクリック）し、FPGA のコンパイルを行います。このコンパイルで HW の動作イメージとなる .sof ファイル、そして次のソフトウェア開発に引き渡すハンドオフ・ファイルを作成します。

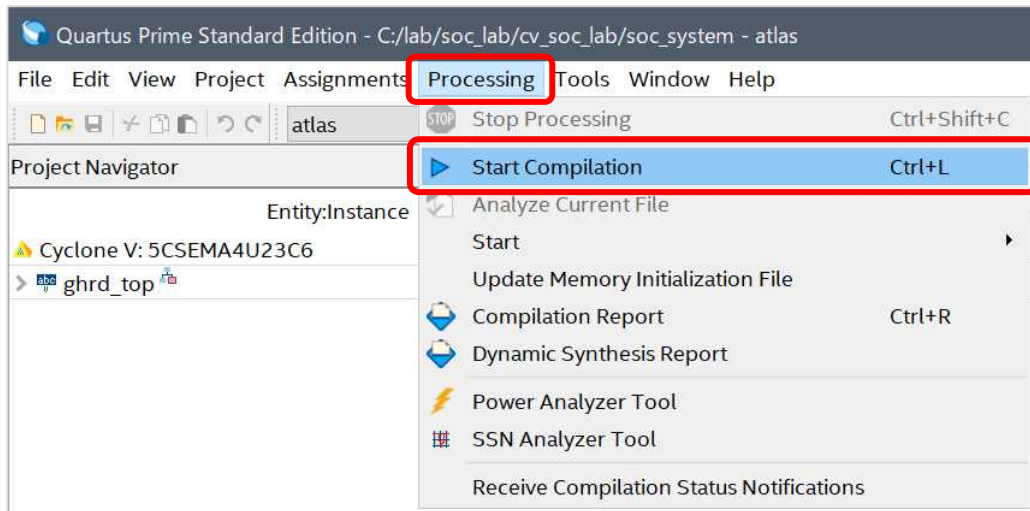


図 3-60. Start Compilation の実行

8. コンパイルの完了を確認します。

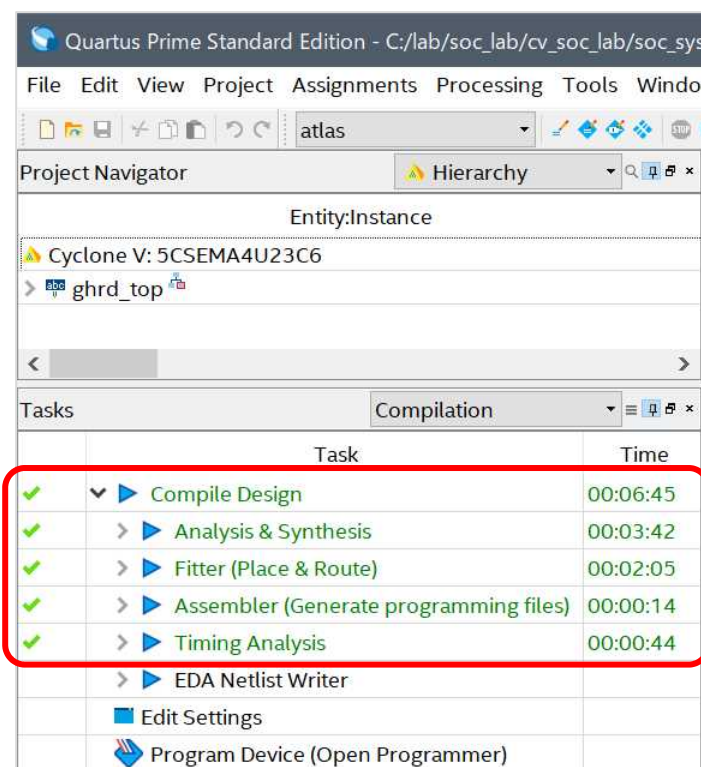


図 3-61. コンパイルの完了

3-12.ステップ 12 : 出力ファイルの確認

Quartus® Prime および Platform Designer で出力したファイルを確認します。

___ 1. Windows® OS のエクスプローラーを使用して、出力ファイルのフォルダー（下記）に移動します。

C:\¥lab¥soc_lab¥cv_soc_lab¥output_files

___ 2. 上記フォルダーに .sof ファイルが出力されていることを確認します。

Atlas-SoC ボードの場合は atlas.sof、

DE10-Nano ボードの場合は DE10-Nano.sof が出力されている事を確認してください。

先ほど説明したように、この ファイルは FPGA の動作イメージファイルです。

このファイルは後の演習で Programmer というツールを使用してボード上の FPGA に書き込みます。

___ 3. Windows® OS のエクスプローラーを使用して、ハードウェア／ソフトウェアのハンドオフ・ディレクトリーに移動します。

C:\¥lab¥soc_lab¥cv_soc_lab¥hps_isw_handoff¥soc_system_hps_0

上記フォルダー以下にツールによって生成されたハードウェア・ソフトウェアのハンドオフ・ファイルがあります。

これらのファイルは Platform Designer の HPS コンポーネント画面で設定した各種データや HPS の SDRAM インターフェイスの情報などの各種ファイルが格納されており、Preloader という HPS 側の初期化に使用されるファイルの生成に利用します。

これらのファイルを用いて後の演習で Preloader の作成を行います。

演習 1 ハードウェア演習のまとめ

このセクションでは、以下の作業を実施し、Arm® プロセッサを含むハードウェアを構成しました。

- Platform Designer システムへの HPS コンポーネントの追加
- HPS コンポーネントの設定
- HPS コンポーネントと他のコンポーネントの接続
- Platform Designer システムの生成
- Quartus® Prime / Platform Designer が出力するファイルの確認

以上で 演習 1 は完了です。

4. 演習 2: ソフトウェア演習(1) Preloader の生成

このセクションでは、「[演習 1: ハードウェア演習](#)」で作成したハンドオフ・ファイルを使用して Preloader を生成します。

⑨ **Point:**

SoC EDS v20.1 (SoC EDS v19.1 以降)では、Preloader (ブートローダー)のビルド作業を Linux OS 環境下で行う必要があります。本演習では Windows® 10 に備わる Windows Subsystem for Linux (WSL1)を使用してブートローダーの生成を行います。

Preloader は U-boot second program loader (以後、u-boot spl) をベースにインテル® SoC FPGA 向けにカスタマイズが加えられたブートローダーです。Preloader の役割は以下の通りです。

- HPS ピン・マルチプレクスの設定
- HPS IOCSR の設定
- HPS PLL とクロックの設定
- HPS ペリフェラルのリセット解除
- SDRAM の初期化 (キャリブレーションなど)
- SDRAM へ次ステージのブート・イメージの展開

上記の通り、Preloader は HPS ブロックの初期化と、U-boot や OS を SDRAM にロードする機能を提供します。Preloader は Quartus® Prime / Platform Designer の設計時に自動生成されるハンドオフ・ファイルを用いることで自動生成されます。このため、ユーザー側で初期化用ソフトウェアの構築をすることなく Quartus® Prime / Platform Designer で設定した内容を HPS ブロックに反映することができます。先ほど確認した sof ファイルは FPGA 側の動作イメージでした。それに対して HPS 側の動作イメージがこの Preloader というファイルです。FPGA 側、HPS 側でそれぞれ異なるファイルを使用して動作イメージを実行するという点にご注意ください。

それでは、Preloader を作成してみましょう。

4-1. ステップ 1 : Embedded Command Shell の起動

1. SoC EDS に含まれている Embedded Command Shell を起動します。

Embedded Command Shell は、Windows® のスタートメニュー、または SoC EDS のインストール・フォルダー以下に格納されている起動用スクリプト Embedded_Command_Shell.bat をダブルクリックして起動します。

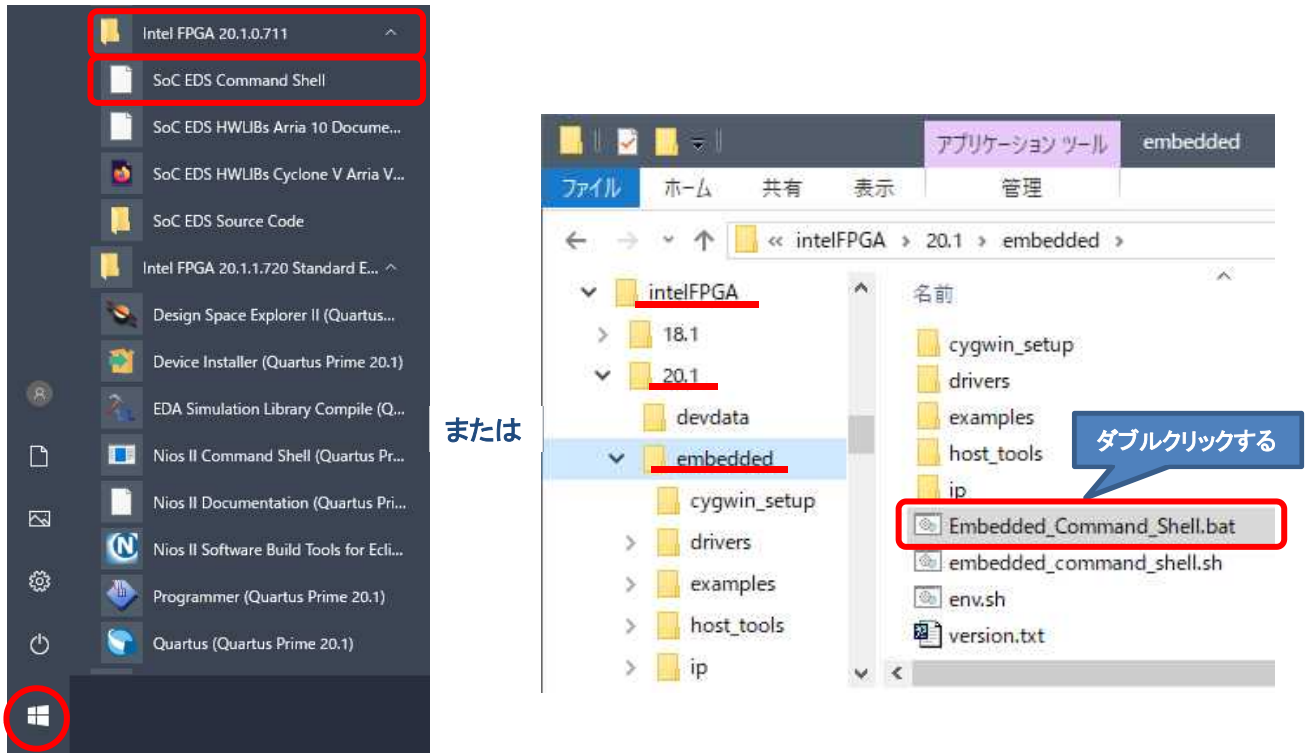


図 4-1. Embedded Command Shell の起動

4-2. ステップ 2 : bsp プロジェクトの生成

Quartus® Prime デザインのハンドオフ情報 (hps_isw_handoff/soc_system_hps_0) から Preloader 生成に必要な bsp プロジェクトを生成します。

- ___ 1. Embedded Command Shell のカレント・ディレクトリーを Quartus® Prime プロジェクト・ディレクトリーへ移動します。

```
$ cd "C:\lab\soc_lab\cv_soc_lab"
```

- ___ 2. 続けて、以下のコマンドを実行し、bsp プロジェクトの出力先となるディレクトリーを作成します。

```
$ mkdir -p software/spl_bsp
```

- ___ 3. 以下のコマンドを実行すると、bsp プロジェクトが生成されます。Bsp プロジェクトの generated ディレクトリーには、ハンドオフ情報が反映された定義を含むソースコードが格納されます。

```
$ bsp-create-settings ¥
--type spl ¥
--bsp-dir software/spl_bsp ¥
--preloader-settings-dir "hps_isw_handoff/soc_system_hps_0" ¥
--settings software/spl_bsp/settings.bsp
```

```

/cygdrive/c/lab/soc_lab/cv_soc_lab
Intel FPGA Embedded Command Shell
Version 20.1 [Build 711]

12088@HD12088C ~
$ cd "C:\lab\soc_lab\cv_soc_lab" 1.
12088@HD12088C /cygdrive/c/lab/soc_lab/cv_soc_lab
$ mkdir -p software/spl_bsp 2.
12088@HD12088C /cygdrive/c/lab/soc_lab/cv_soc_lab
$ bsp-create-settings ¥
> --type spl ¥
> --bsp-dir software/spl_bsp ¥
> --preloader-settings-dir "hps_isw_handoff/soc_system_hps_0" ¥
> --settings software/spl_bsp/settings.bsp 3.

INFO: Creating BSP settings file...
INFO: nios2-bsp-create-settings --type spl --bsp-dir software/spl_bsp --preloader-settings-dir hps_isw_handoff/soc_syste
m_hps_0 --settings software/spl_bsp/settings.bsp
INFO: Searching for BSP components with category: driver_element
INFO: Searching for BSP components with category: software_package_element
INFO: Generated file "C:\lab\soc_lab\cv_soc_lab\software\spl_bsp\settings.bsp"
INFO: BSP settings file was created at location "C:\lab\soc_lab\cv_soc_lab\software\spl_bsp\settings.bsp".
INFO: Generating BSP files...
INFO: Generating BSP files in "C:\lab\soc_lab\cv_soc_lab\software\spl_bsp"
INFO: Generated file "C:\lab\soc_lab\cv_soc_lab\software\spl_bsp\settings.bsp"
INFO: Tcl message: "Reading preloader settings dir: C:/lab/soc_lab/cv_soc_lab/hps_isw_handoff/soc_system_hps_0"
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/sdram_config.h..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/pinmux_config.h..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/pinmux_config_cyclone5.c..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/reset_config.h.template..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/Makefile.template..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/pll_config.h"
INFO: Tcl message: "Reading file: C:/lab/soc_lab/cv_soc_lab/hps_isw_handoff/soc_system_hps_0/hps_isw_handoff/soc_system_hps_0.hiof..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/iocsr_config_cyclone5.h..."
INFO: Tcl message: "Generating file: C:/lab/soc_lab/cv_soc_lab/software/spl_bsp/generated/iocsr_config_cyclone5.c..."
INFO: Finished generating BSP files. Total time taken = 2 seconds
INFO: BSP files generated in "C:\lab\soc_lab\cv_soc_lab\software\spl_bsp"

12088@HD12088C /cygdrive/c/lab/soc_lab/cv_soc_lab
$

```

図 4-2. Bsp プロジェクトの生成

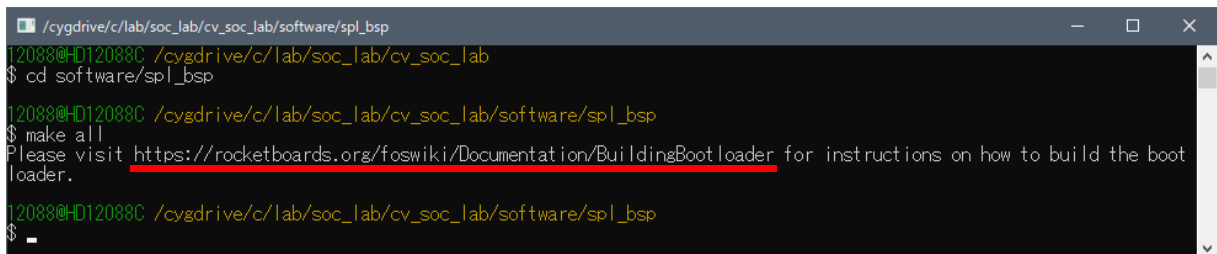
⚠ 注記:

Windows® 10 環境で SoC EDS v20.1std の bsp-create-settings を使用する際には事前にエラー対策が必要となります。エラーが発生する場合は、以下の参考情報サイトに記載の対策を行った上で再度 bsp-create-settings を実行してください。

📖 参考: アルティマ技術サポート「[SoC EDS 環境で bsp-create-settings が実行エラーになるトラブルの回避策](#)」

📌 Note:

bsp プロジェクトには Makefile が含まれますが SoC EDS v20.1 (SoC EDS v19.1 以降)の環境では使用しません。Bsp プロジェクト以下で make を実行すると、ブートローダーのビルド手順が掲載された Rocketboards.org サイト内のドキュメントページ (URL) が表示されます。



```

/cygdrive/c/lab/soc_lab/cv_soc_lab/software/spl_bsp
12088@HD12088C /cygdrive/c/lab/soc_lab/cv_soc_lab
$ cd software/spl_bsp
12088@HD12088C /cygdrive/c/lab/soc_lab/cv_soc_lab/software/spl_bsp
$ make all
Please visit https://rocketboards.org/foswiki/Documentation/BuildingBootloader for instructions on how to build the boot loader.
12088@HD12088C /cygdrive/c/lab/soc_lab/cv_soc_lab/software/spl_bsp
$

```

参照: <https://rocketboards.org/foswiki/Documentation/BuildingBootloader>

bsp-create-settings により、software/spl_bsp ディレクトリー以下に bsp プロジェクトが生成されました。この後、bsp プロジェクトに含まれるソースファイルを使用して Preloader のビルドを行います。

4-3. ステップ 3 : Preloader ビルド環境の起動

演習 2 の最初に記載した通り、Preloader(ブートローダー)のビルドには Linux OS 環境が必要です。この演習では、WSL1 上で動作する Ubuntu 18.04 LTS を使用します。以降のステップは WSL の Ubuntu 18.04 LTS ターミナル上での作業となりますので、事前に WSL / Ubuntu 18.04 LTS 環境のセットアップを実施してください。

⚠ 注意事項:

WSL 環境には、最初にリリースされた WSL 1 の他に、Windows® 10 バージョン 2004 以降で対応している WSL 2 (Windows Subsystem for Linux 2) という異なる 2 つの環境が存在します。

この演習で扱う内容は WSL 1 を使用した例となりますのでご注意ください。

📌 Note:

WSL 1 環境のセットアップ方法は以下の記事を参考にご対応ください。

📖 参考: アルティマ技術サポート「[WSL で Preloader / U-Boot をビルドしてみる【その 1】環境セットアップ編](#)」

※弊社開催の「SoC スタートアップ・トライアル・セミナー」では、予めセットアップ済みとなります。

1. WSL1 上で動作する Ubuntu 18.04 LTS を起動します。

Windows® のスタートメニュー上で Ubuntu と入力し、候補として表示される Ubuntu 18.04 LTS アプリをクリックします。

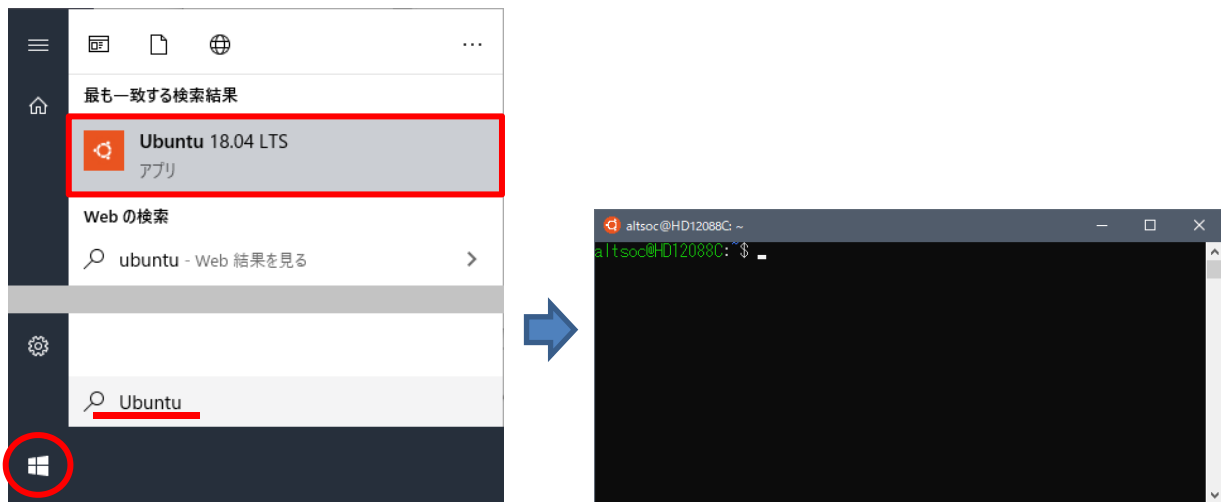


図 4-3. Embedded Command Shell の起動

Note:

使用する Linux OS 環境には、ビルドに必要な各種パッケージを追加でインストールしておく必要があります。Ubuntu 18.04 LTS の場合は予め以下のコマンドで必要なパッケージを追加しておきます。途中、「続行しますか? [Y/n]」と表示されたら Y を入力してください。

```
$ sudo apt update
```

```
$ sudo apt upgrade
```

```
$ sudo apt install build-essential bison flex ncurses-dev
```

※弊社開催の「SoC スタートアップ・トライアル・セミナー」では、予めセットアップ済みとなります。

2. ベアメタル GCC ツールチェーンを WSL1 上の Ubuntu 18.04 LTS にインストールします。

Ubuntu 18.04 LTS ターミナルにて以下のコマンドを実行し、GCC ツールチェーンをダウンロード・展開してください。既にインストール済みの場合は次の手順へ進んでください。

```
$ mkdir ~/toolchains
```

```
$ cd ~/toolchains
```

```
$ wget https://releases.linaro.org/components/toolchain/binaries/latest-7/arm-eabi/gcc-linaro-7.5.0-2019.12-x86_64_arm-eabi.tar.xz
```

```
$ tar xf gcc-linaro-7.5.0-2019.12-x86_64_arm-eabi.tar.xz
```

Note:

GCC ツールチェーンのダウンロードはファイルサイズが大きいので時間が掛かります。

※弊社開催の「SoC スタートアップ・トライアル・セミナー」では、予めセットアップ済みとなります。

3. ベアメタル GCC ツールチェーンを使用するための環境変数を設定します。

Ubuntu 18.04 LTS ターミナル にて以下のコマンドを実行します。

```
$ export PATH=~/.toolchains/gcc-linaro-7.5.0-2019.12-x86_64_arm-eabi/bin:$PATH
$ export ARCH=arm
$ export CROSS_COMPILE=arm-eabi-
```

4-4. ステップ 4 : Preloader のビルド

1. Ubuntu 18.04 LTS ターミナル のカレント・ディレクトリーを bsp プロジェクト・ディレクトリーへ移動します。

```
$ cd /mnt/c/lab/soc_lab/cv_soc_lab/software/spl_bsp
```

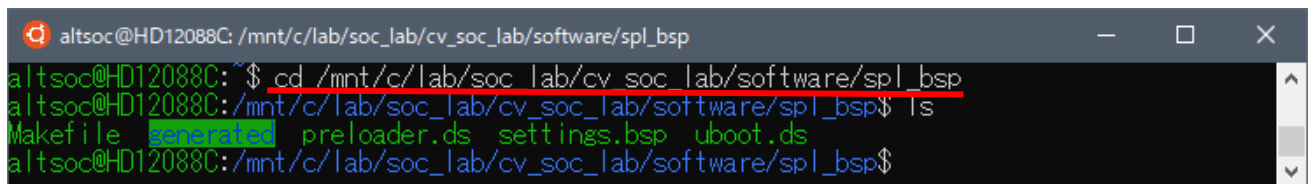


図 4-4. Bsp プロジェクト・ディレクトリーへの移動

2. Bsp プロジェクト・ディレクトリー以下に U-Boot/Preloader のソースツリーをダウンロード・展開します。

Ubuntu 18.04 LTS ターミナル にて以下のコマンドを実行してください。

```
$ wget https://github.com/altera-opensource/u-boot-socfpga/archive/ACDS20.1STD_REL_GSRD_PR.tar.gz
$ tar -xzf ACDS20.1STD_REL_GSRD_PR.tar.gz
$ mv u-boot-socfpga-ACDS20.1STD_REL_GSRD_PR/ uboot-socfpga
```

① Note:

オフライン作業時など、wget が行えない場合は、ダウンロード済みのソースツリーをご使用ください。代わりに以下のコマンドで U-Boot/Preloader のソースツリーをコピー・展開してください。

```
$ tar -xzf /mnt/c/lab/soc_lab/cv_soc_lab/solution/uboot-socfpga/ACDS20.1STD_REL_GSRD_PR.tar.gz
$ mv u-boot-socfpga-ACDS20.1STD_REL_GSRD_PR/ uboot-socfpga
```

※弊社開催の「SoC スタートアップ・トライアル・セミナー」では、こちらの手順で対応してください。

3. Uboot-socfpga ソースツリー直下へ移動し、qts-filter.sh を実行します。

```
$ cd uboot-socfpga
$ ./arch/arm/mach-socfpga/qts-filter.sh
cyclone5 ../../../../board/altera/cyclone5-socdk/qts/
```

この操作により、bsp-create-settings で生成された各種ヘッダーファイルが uboot-socfpga ソースツリー以下へ取り込まれます。

4. U-Boot/Preloader ソースツリーを Cyclone V SoC ターゲット向けにコンフィギュレーションします。

```
$ make socfpga_cyclone5_defconfig
```

```
altsoc@HD12088C: /mnt/c/lab/soc_lab/cv_soc_lab/software/spl_bsp/uboot-socfpga
altsoc@HD12088C:/mnt/c/lab/soc_lab/cv_soc_lab/software/spl_bsp/uboot-socfpga$ make socfpga_cyclone5_defconfig
HOSTCC scripts/basic/fixdep
HOSTCC scripts/kconfig/conf.o
YACC scripts/kconfig/zconf.tab.c
LEX scripts/kconfig/zconf.lex.c
HOSTCC scripts/kconfig/zconf.tab.o
HOSTLD scripts/kconfig/conf
#
# configuration written to .config
#
altsoc@HD12088C:/mnt/c/lab/soc_lab/cv_soc_lab/software/spl_bsp/uboot-socfpga$
```

図 4-5. U-Boot/Preloader ソースツリーのコンフィギュレーション

5. U-Boot/Preloader ソースツリーの一部をデバッグ用に書き換えます。

Ubuntu 18.04 LTS ターミナル にて以下のコマンドを実行し、vi エディターで編集を行います。

エディター起動後、**i** キーをタイプすると入力モードになるので下の図の内容に従ってソースを書き換えます。編集が終わったら **ESC** キーを 1 回押して、続けて **:wq** とキー入力して保存および終了します。

Note:

vi エディターでは、ノーマルモードの状態で **/** (スラッシュ) キーに続けて検索キーワードを入力することで、ファイル内検索が利用できます。入力モード中(ウィンドウ左下に **- 挿入 -** もしくは **- INSERT -** と表示された状態)の場合は **ESC** キーを 1 回押下するとノーマルモードになります。

ソース上の編集箇所が見つからない場合は、下の図の吹き出しの情報を参考に、**/spl_boot_device** や **/socfpga_sdram_apply_static_cfg** をキーワードにファイル内検索を行ってみてください。

```
$ vi arch/arm/mach-socfpga/spl_gen5.c
```

```
#include <watchdog.h>
#include <dm/uclass.h>
#include <asm/arch/fpga_manager.h>

DECLARE_GLOBAL_DATA_PTR;

static const struct socfpga_system_manager *sysmgr_regs =
    (struct socfpga_system_manager *)SOCFPGA_SYSMGR_ADDRESS;

u32 spl_boot_device(void)
{
    const u32 bsel = readl(&sysmgr_regs->bootinfo);

    if(fpgamgr_test_fpga_ready()) {
        /* bridge enable */
        do_bridge_reset(1, 7);
    }

    switch (SYSMGR_GET_BOOTINFO_BSEL(bsel)) {
    case 0x1: /* FPGA (HPS/FPGA Bridge) */
        return BOOT_DEVICE_RAM;
    case 0x2: /* NAND Flash (1.8V) */
    case 0x3: /* NAND Flash (3.0V) */
        return BOOT_DEVICE_NAND;
    }
}
```

※ ヘッダーインクルード追加:

```
#include <asm/arch/fpga_manager.h>
```

※ spl_boot_device 先頭に以下の処理を追加:

```
if(fpgamgr_test_fpga_ready()) {
    /* bridge enable */
    do_bridge_reset(1, 7);
}
```

図 4-6. Arch/arm/mach-socfpga/spl_gen5.c の編集

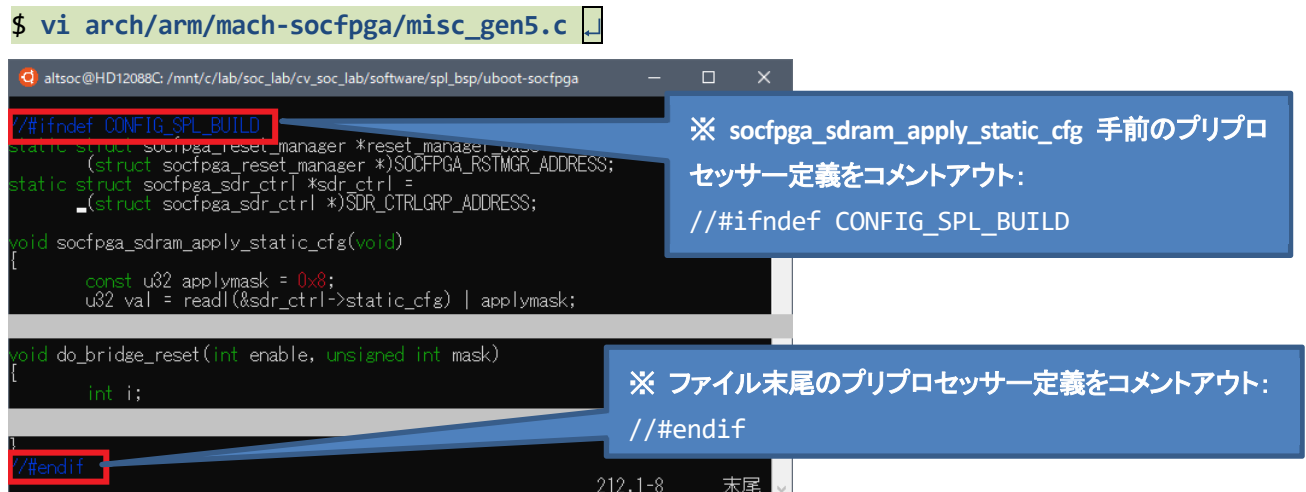


図 4-7. Arch/arm/mach-socfpga/misc_gen5.c の編集

Note:

ここで行ったソースコードの編集内容としては、Preloader 実行中に HPS-FPGA 間のブリッジインターフェースを開通させる処理を追加しています。通常、U-Boot 実行時に `bridge enable` コマンドで行う操作と同等の処理を追加しています。

参考: アルティマ技術サポート「[インテル® SoC FPGA 向け U-Boot の HPS-FPGA 間のブリッジ開放コマンドについて](#)」

6. ソースツリーをビルドし、U-Boot および Preloader を生成します。

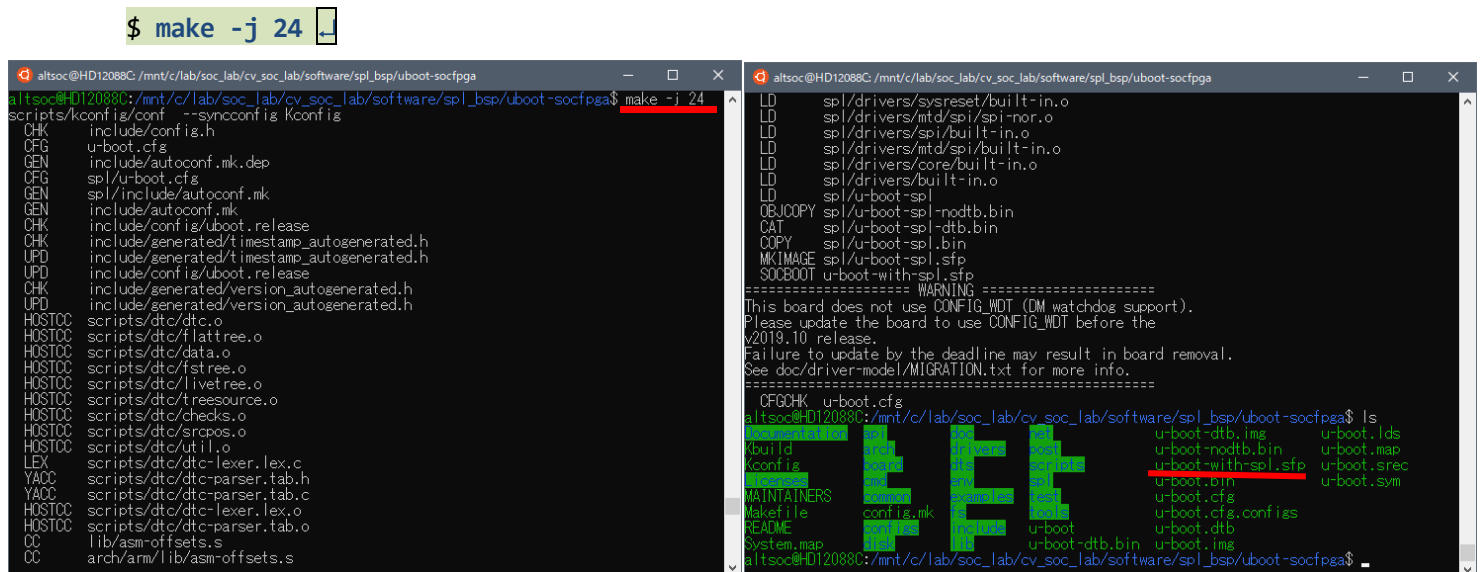


図 4-8. U-boot および Preloader の生成

7. 実行後、エラーがなく終了したことを確認します。

エラーなく終了したことを確認後 `ls` コマンドにて `u-boot-with-spl.sfp` が生成されていることを確認します。
このファイルは、U-Boot と Preloader の両方を含むバイナリーファイルとなっており、SD カードや QSPI フラッシュメモリーへ書き込むファイルとなります。尚、Preloader 部分には BootROM にて参照される `mkpimage` ヘッダー、U-Boot には Preloader にて参照される `mkimage` ヘッダー情報が付加されています。

以上で 演習 2 は完了です。

5. 演習 3: ソフトウェア演習(2) ベアメタル・アプリケーション

このセクションでは、Arm® DS を使用し、SoC EDS に付属の Hello World サンプル・アプリケーションおよび本演習用に用意された LED Blink サンプル・アプリケーションを実行し、ソフトウェアの開発手法およびデバッグ手法について解説します。

以下にサンプル・アプリケーションの概要を記述します。

- Hello World サンプル・アプリケーションの概要

このサンプル・アプリケーションは、Arm® DS が持つセミホスティング機能を使用して、デバッガ・コンソールに “Hello Tim” というメッセージを出力します。

この方法ではデバイスのペリフェラルは使用されず、すべての通信は JTAG を通じて行われます。

実行するアプリケーションは、64KB のオンチップ RAM にダウンロードされ実行が開始されます。このため、ボード上の SDRAM メモリを設定を必要としません。

上記の理由から、インテル® SoC FPGA が実装されたすべてのボードで実行することが可能です。

- LED Blink サンプル・アプリケーションの概要

このサンプル・アプリケーションでは、「[演習 1: ハードウェア演習](#)」にて作成した FPGA デザインを用い、Arm® プロセッサから FPGA ファブリック側に実装された PIO ペリフェラルにアクセスし LED の点灯、消灯を制御します。

このサンプル・アプリケーションはメイン・アプリケーションを実行する前に、**Preloader** と呼ばれる HPS ブロックの初期化ソフトウェアを実行し、SDRAM のキャリブレーション、クロックの設定、HPS-FPGA 間のブリッジの初期化等を行います。これにより、FPGA ファブリック側のペリフェラルにアクセスすることが可能な状態でメイン・アプリケーションを実行します。また、メイン・アプリケーションは SDRAM にロードされ実行を開始します。

⚠ 注記:

- この演習を行う前に、Linux® (または他の OS) が、ボード上で実行されていないことを確認してください。OS は、ベアメタル・アプリケーションのダウンロードおよびデバッグ機能を妨げる可能性があります (microSD カードが挿入されている場合は、外してください)。
- このセクションでの説明、画面スナップショットおよびコマンドは、SoC EDS の Windows® バージョンを使用して作成されたものですが、Linux ホスト PC 上でも同様の方法で実行することができます。
- このセクションで示すパスは、デフォルトのインストール・パスを使用したと仮定します。標準以外の場所が使用されている場合は、それに応じて調整してください。
- ベアメタル・アプリケーションを Arm® DS でデバッグする場合、ライセンスが必要となります。ライセンスは、MAC Address に紐づけられています。
紐づけられているネットワーク・インターフェイスを PC に認識させておいてください。

5-1. FPGA デザインのダウンロード

ソフトウェアの演習を開始する前に、「3. 演習 1: ハードウェア演習」で作成したハードウェア・デザイン（sof ファイル）を FPGA にダウンロードします。「2. ボードの設定」のセクションを参照し、ボードのセットアップが完了していることを再度確認してください。セットアップに問題がなければ、J14 に AC アダプターを接続してください。

1. Quartus® Prime の **Tools** メニュー ⇒ **Programmer**、または **Programmer** アイコン  をクリックし、Programmer を起動します。
2. **Programmer** 内にある **[Hardware Setup]** ボタンをクリックし、Hardware Setup ウィンドウ内の **Currently selected hardware** のプルダウンリストから **DE-SoC** を選択し、ウィンドウを Close します。

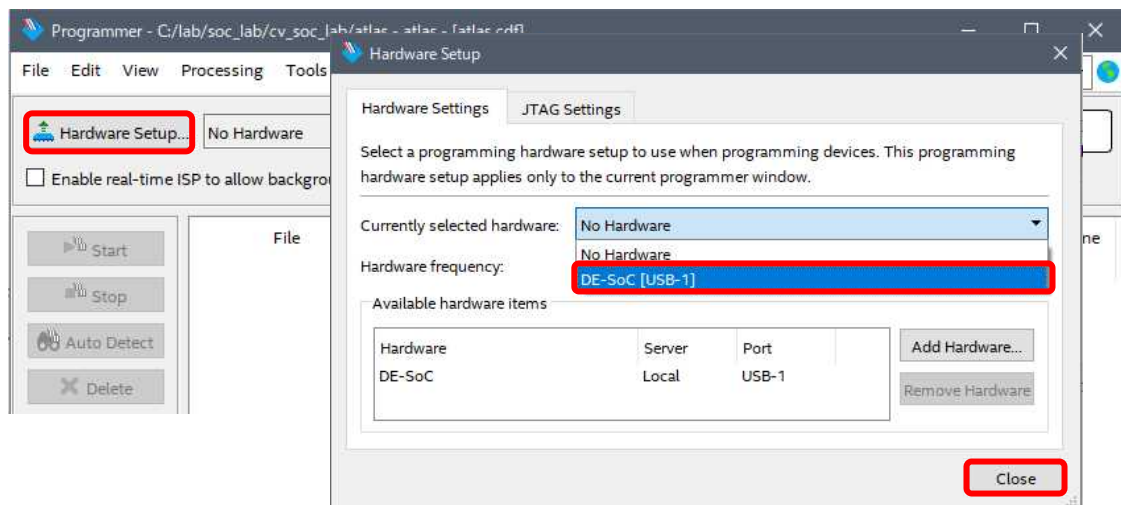


図 5-1. Hardware Setup

3. **[Auto Detect]** ボタンをクリックし、基板上の JTAG チェインに接続されている FPGA を検出します。
4. **Select Device** ウィンドウから Atlas-SoC ボードの場合は **5CSEMA4** を、DE10-Nano ボードの場合は **5CSEBA6** を選択し、**[OK]** をクリックします。

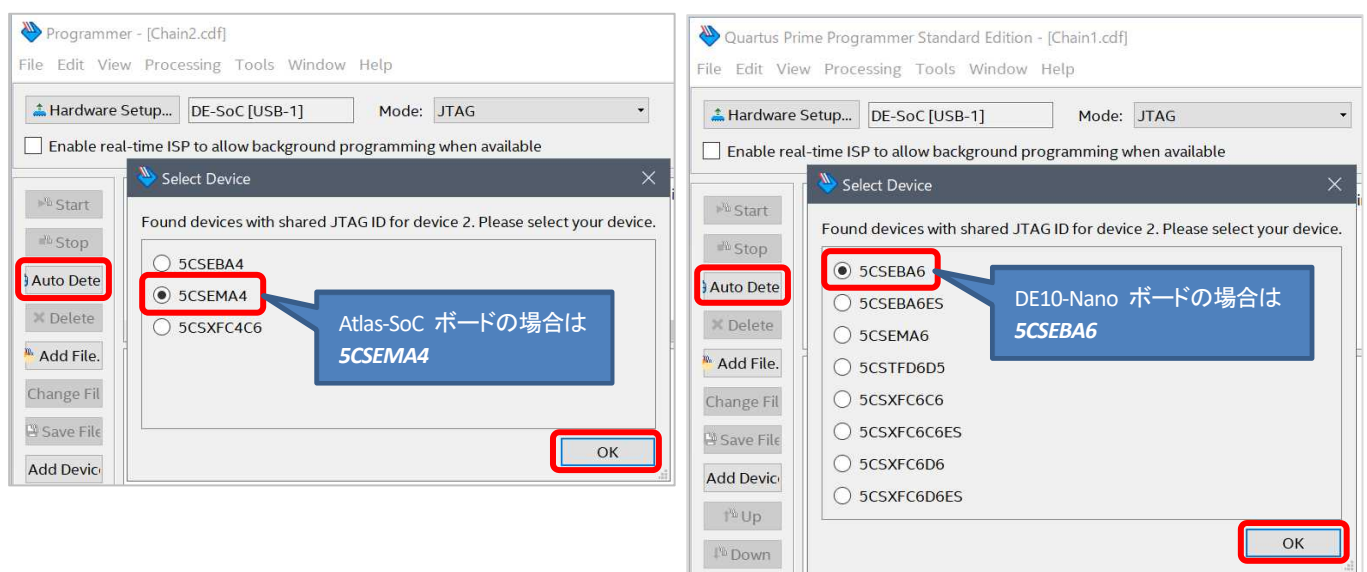


図 5-2. デバイスの選択

5. 以下のダイアログボックスが表示された場合は、[Yes] を選択します。



図 5-3. ダイアログボックス

これにより、JTAG Chain 上に SOCVHPS と 5CSMA4/5CSEBA6 が表示されます。SOCVHPS は HPS 側、5CSMA4/5CSEBA6 は FPGA 側が認識されたことをそれぞれ示しています。

6. ダウンロードするファイルを選択します。

Device 欄の 5CSEMA4/5CSEBA6 上で右クリックし、**Change File** をクリックします。**Select New Programming File** ダイアログボックスで、c:\lab\soc_lab\cv_soc_lab\output_files をブラウズし Atlas-SoC ボードの場合は atlas.sof を、DE10-Nano ボードの場合は DE10-Nano.sof を選択します。

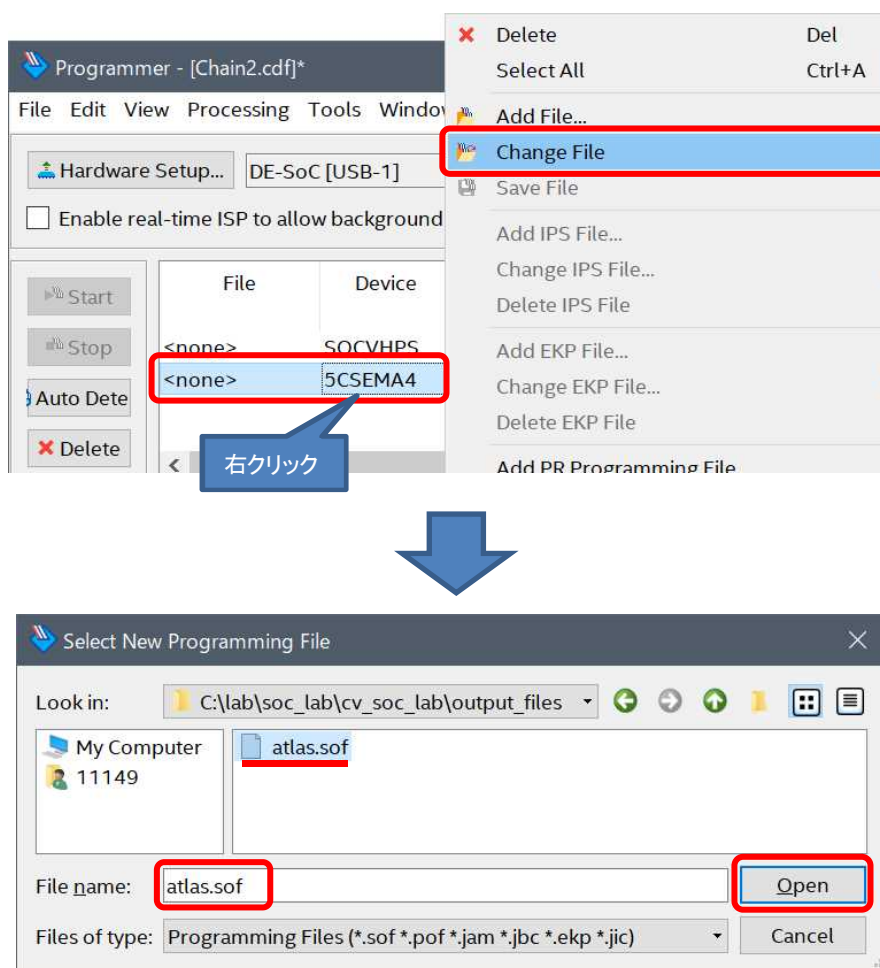


図 5-4. Sof ファイルの選択

7. **Program/Configure** にチェックを入れた後、**[Start]** ボタンをクリックしてコンフィグレーションを行います。
右上の Progress パーが 100% になったら FPGA 側に動作イメージが書き込まれた状態となります。

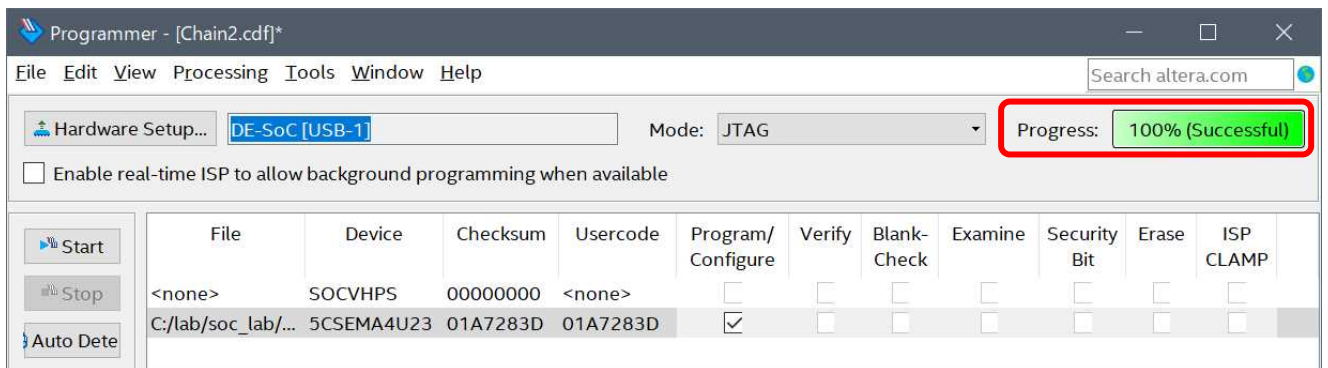
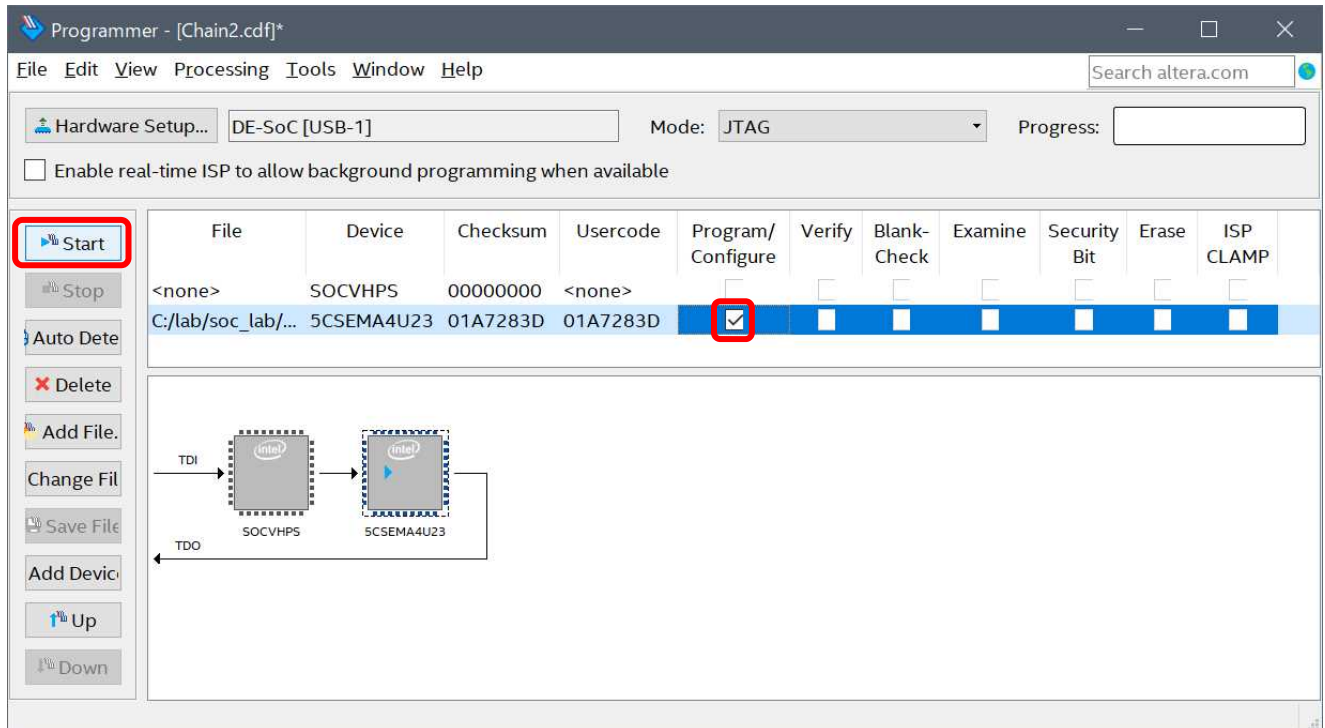


図 5-5. Sof のダウンロード

5-2. Hello World サンプル・アプリケーションの実行

続いて HPS 上でサンプル・アプリケーションを動作させてみましょう。はじめに Arm® DS を立ち上げます。

- ___ 1. SoC EDS に含まれている Embedded Command Shell 上より Arm® DS を起動します。

Embedded Command Shell は、Windows® のスタートメニュー、または SoC EDS のインストール・フォルダー以下に格納されている起動用スクリプト *Embedded_Command_Shell.bat* をダブルクリックして起動します。

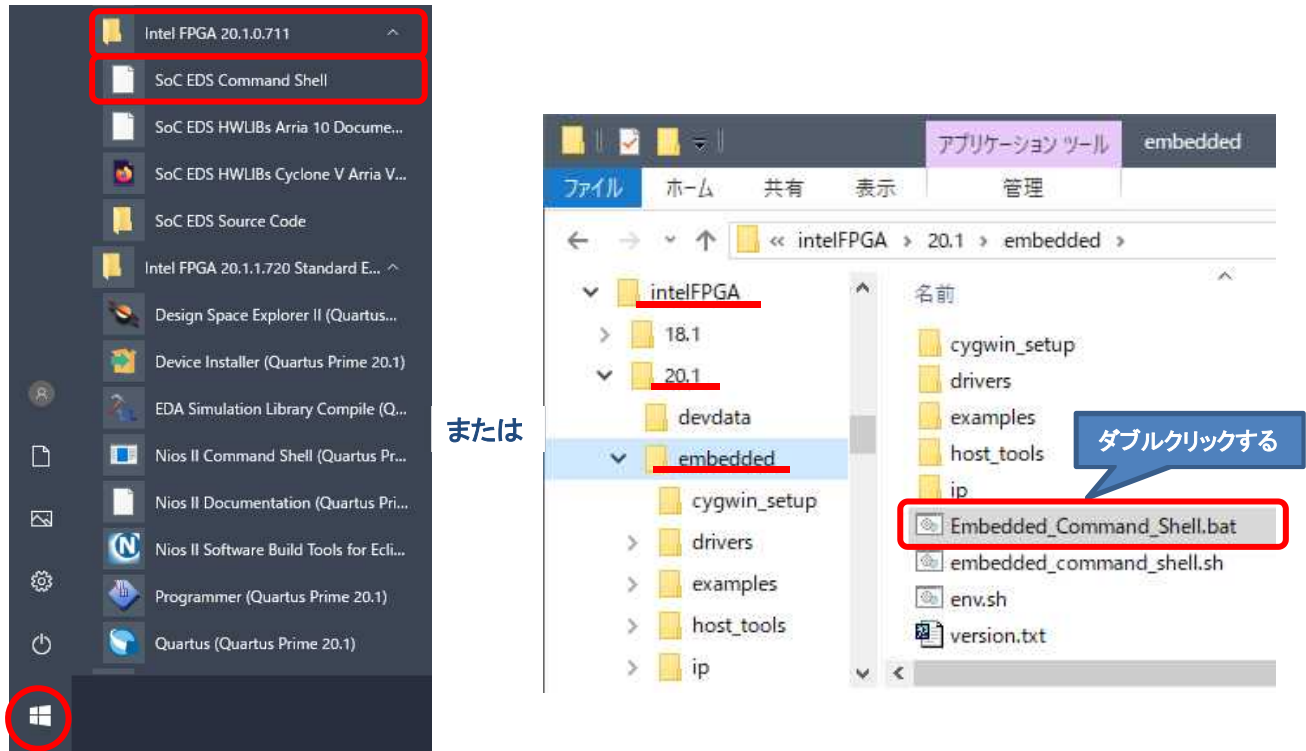


図 5-6. Embedded Command Shell の起動

- ___ 2. Embedded Command Shell 上で以下のコマンドを実行し、Arm® DS を起動します。

```
$ exec /cygdrive/c/ProgramFiles/Arm/Development/Studio/2020.1/bin/cmdsuite.exe
> bash
$ armds_ide &
```

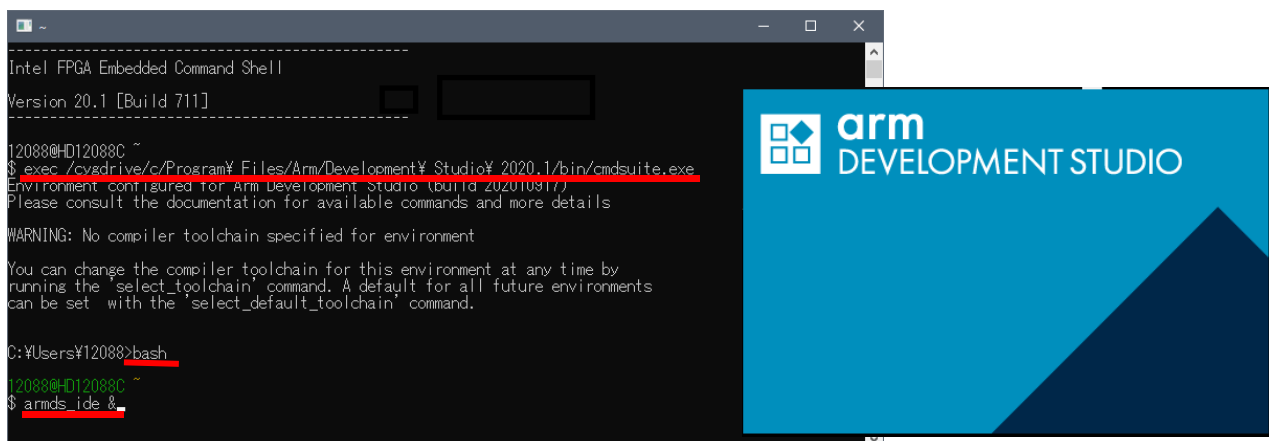


図 5-7. Arm® DS の起動と起動画面

3. Arm® DS での作業に使用するワークスペース・フォルダーを設定します。

この演習では、「3. 演習 1: ハードウェア演習」の作業フォルダーに workspace を作成します。

以下のパスを指定して [Launch] をクリックします (フォルダーが存在しない場合は自動的に作成されます)。

C:\lab\soc_lab\cv_soc_lab\workspace

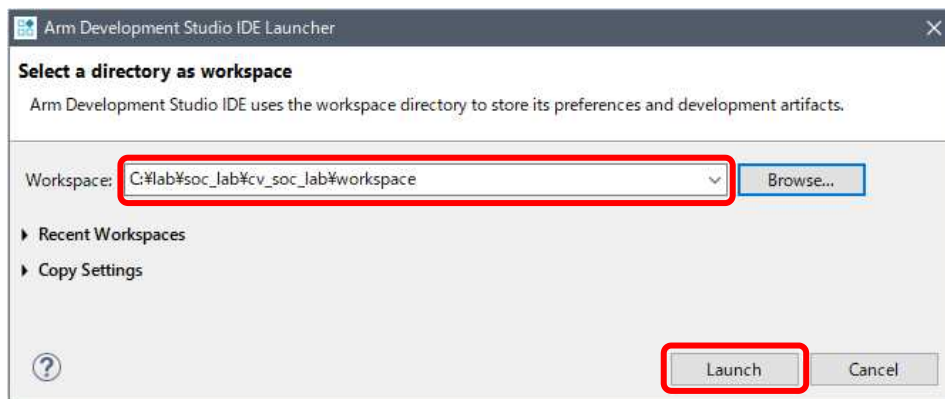


図 5-8. Workspace の作成

4. Preferences Wizard が表示された場合は、内容を確認の上で [Apply & Close]、Arm® DS の Welcome 画面が表示された場合は、[閉じる] (×マーク) をクリックして閉じます。

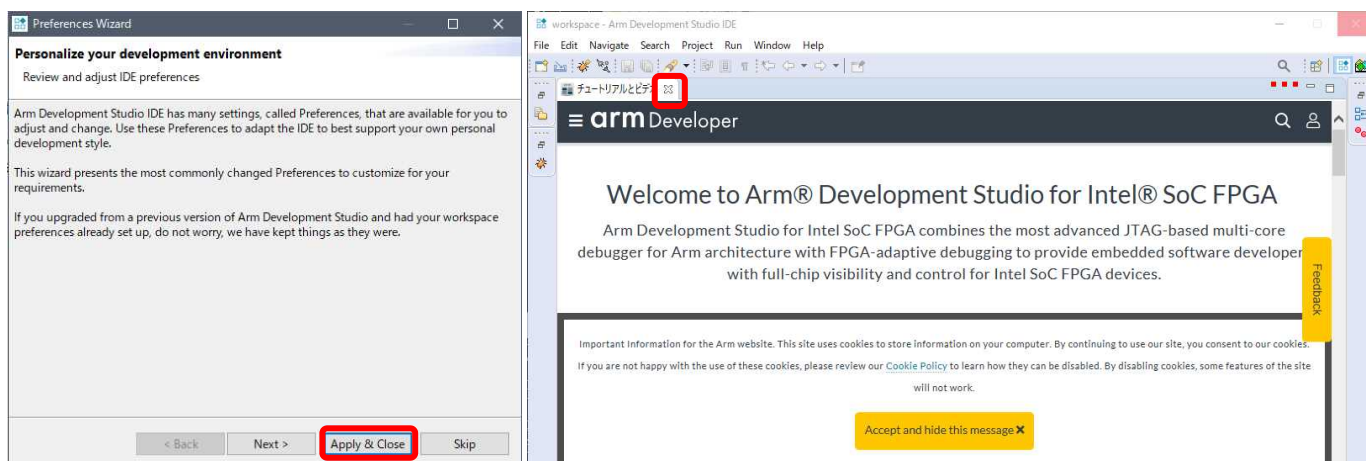


図 5-9. Preference Wizard および Welcome 画面

① Note:

現バージョンの Arm® DS (2020.1) では、Welcome 画面(チュートリアルとビデオ)をオンライン表示する際にウィンドウ操作の反応が重くなる現象が確認されています。オンライン表示が完了するまで無理に操作を行わずにお待ちください。Welcome 画面を閉じた後は、スムーズに操作頂ける状態になります。

Arm® DS をオフラインの状態では起動頂くと、オフライン用の Welcome 画面が使用されるため、この問題を回避することが可能です。

続いて、Hello World サンプル・アプリケーションをインポートします。

Hello World サンプル・アプリケーションは SoC EDS に Software Example として入っています。

___ 5. Arm® DS のメニューから「File」⇒「Import」を選択します。

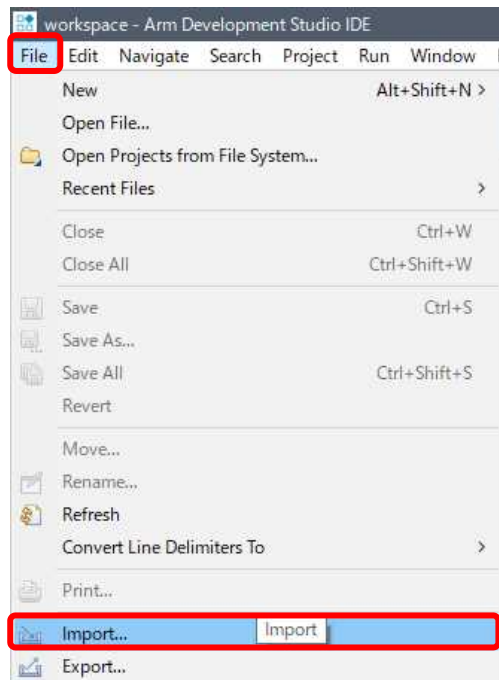


図 5-10. 「インポート」メニュー

___ 6. 「General (一般)」⇒「Existing Projects into Workspae(既存プロジェクトをワークスペースへ)」を選択し [Next] をクリックします。

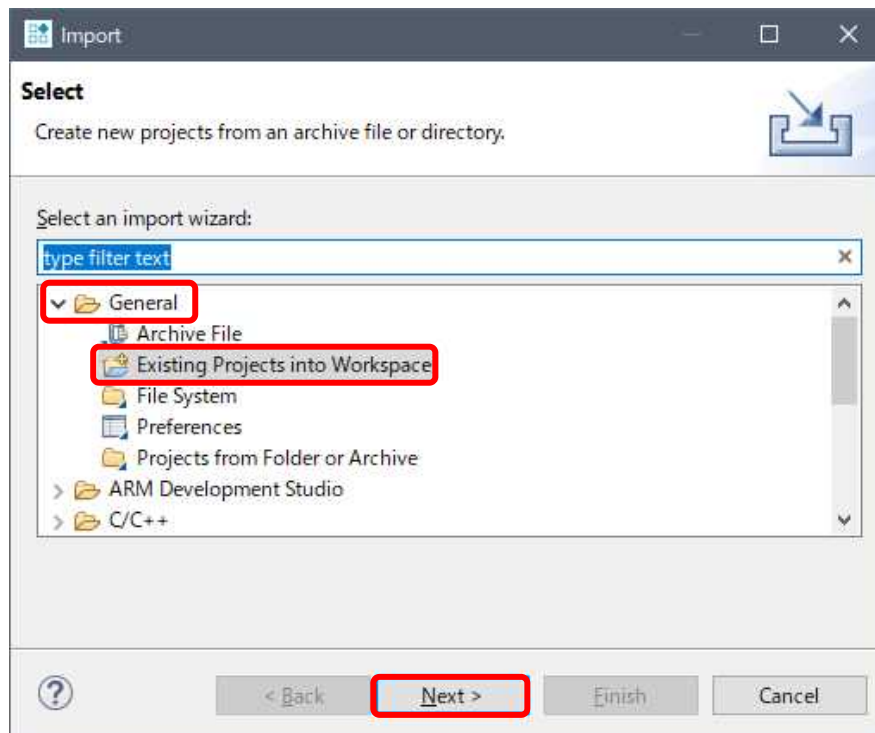


図 5-11. 既存プロジェクトのインポート

7. 「**Select archive file:** (アーカイブ・ファイルの選択)」オプションを選択し、**[Browse]** ボタンよりサンプル・プロジェクトを指定します。

サンプル・プロジェクトは SoC EDS に含まれており、デフォルトでは以下のインストール・フォルダーにあります。

C:\intelFPGA\20.1\embedded\examples\software\Altera-SoCFPGA>HelloWorld-Baremetal-GNU.tar.gz

(<SoC EDS インストール・ディレクトリー>\examples\software\Altera-SoCFPGA>HelloWorld-Baremetal-GNU.tar.gz をインポートしています)。

選択後、**[Finish]** ボタンをクリックします。

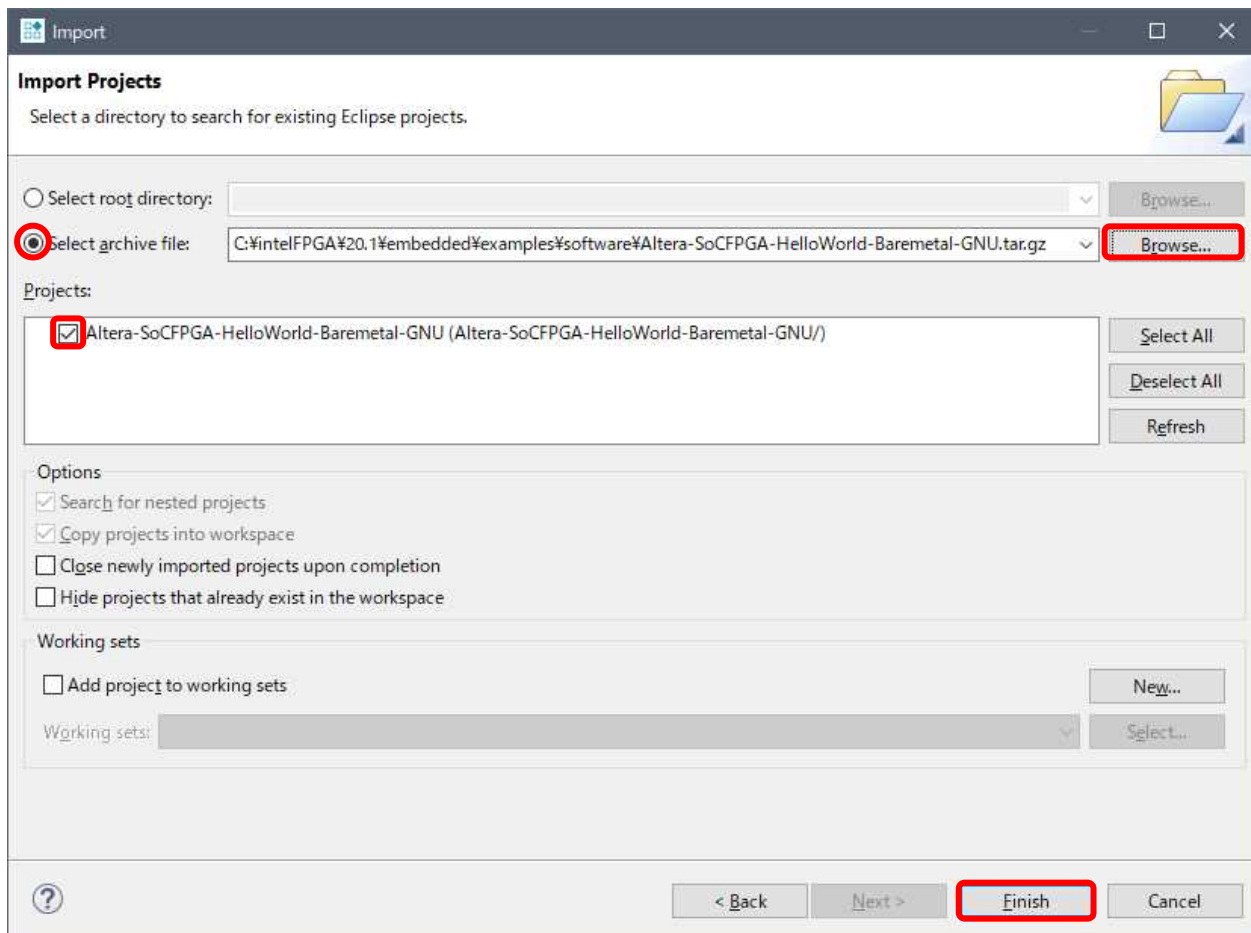


図 5-12. Hello World サンプル・アプリケーションの選択

この作業を実施すると、Arm® DS ウィンドウ左側の プロジェクト・エクスプローラーにプロジェクトに含まれる各種ファイルが表示されます。

次に Hello World サンプル・アプリケーションをコンパイルします。

- ___ 8. プロジェクト・エクスプローラー・タブよりプロジェクトを選択しハイライトします。
- ___ 9. Arm® DS のメニューから「**Project**」⇒「**Build Project**」を選択します。もしくは、プロジェクト・エクスプローラー上でプロジェクトを選択し、**右クリック** ⇒「**Build Project**」を実行します。

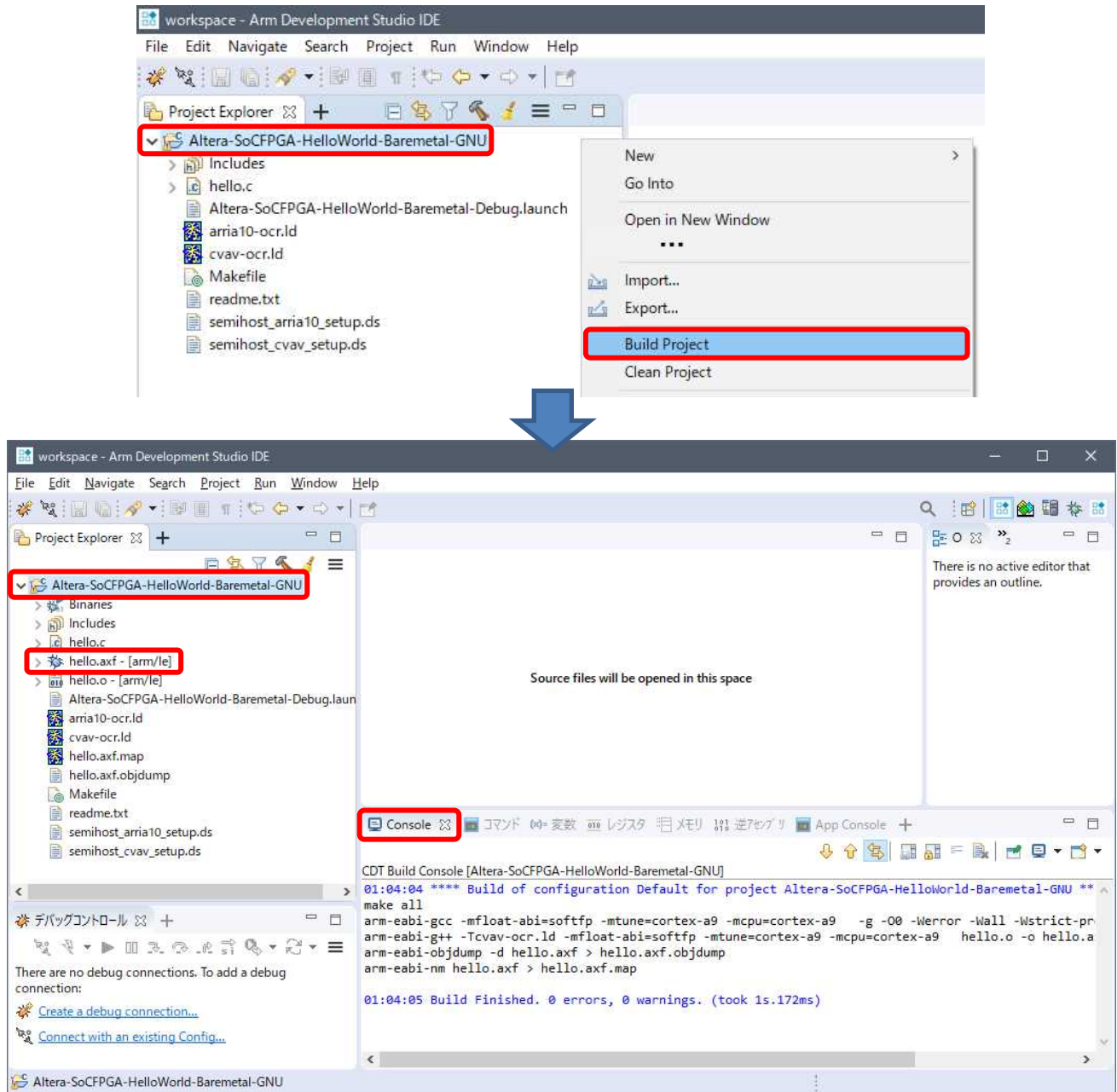


図 5-13. Hello World プロジェクトのビルド

プロジェクトがコンパイルされ、上記の図に示すように、プロジェクト・エクスプローラーに **hello.axf** という Arm® DS 上での実行可能バイナリーが出力されます。

Console ウィンドウ上には、実行可能バイナリーを生成する際に実行されたコマンドが表示されています。

最後に、先ほど生成した Hello World サンプル・アプリケーション (*hello.axf*) を実行します。

- ___ 10. 「Run」メニュー ⇒ 「デバッグコンフィギュレーション(B)」を選択します。

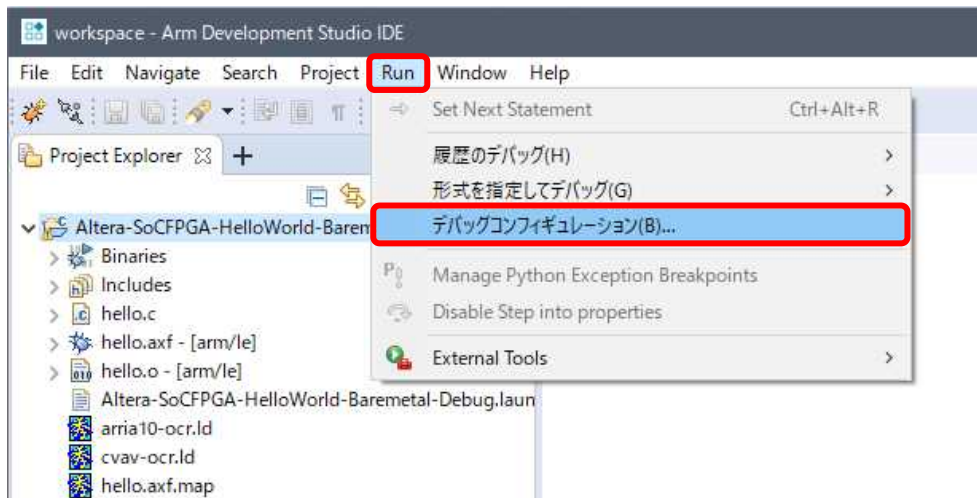


図 5-14. デバッグコンフィギュレーションの選択

- ___ 11. デバッグコンフィギュレーション (Debug Configurations) ウィンドウにある左側のパネルから、
汎用 ARM C/C++ アプリケーション ⇒ **Altera-SoCFPGA-HelloWorld-Baremetal-Debug** を選択します (表示されない場合は、汎用 ARM C/C++ アプリケーションの横にある (>) をクリックしてください)。
 ターゲット接続は、インテル® FPGA ダウンロード・ケーブル (USB-Blaster™) を利用し、
Intel SoC FPGA ⇒ **Cyclone V SoC (Dual Core)** ⇒ **Bare Metal Debug** ⇒ **Debug Cortex-A9_0** となるように設定されています。

- ___ 12. 接続セクションの右側にある [参照] ボタンを押下し、USB-Blaster™ 接続の選択画面を表示させます。

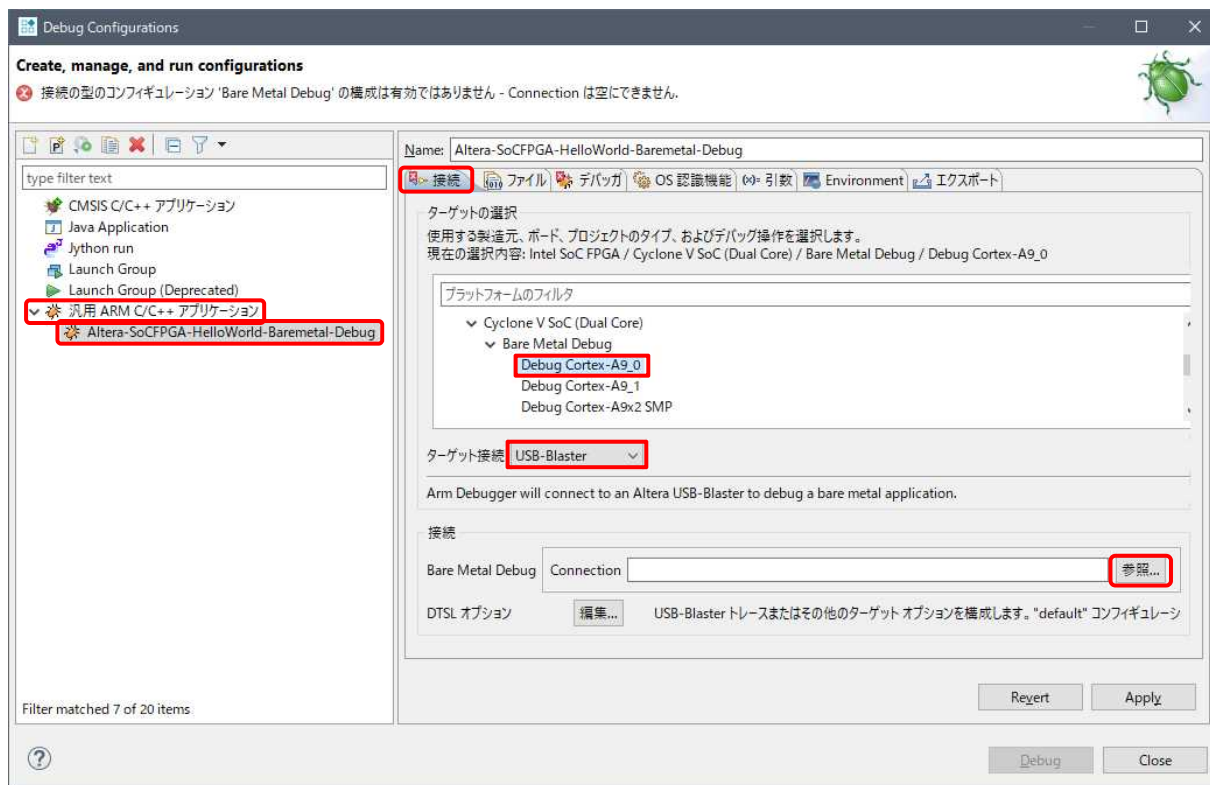


図 5-15. デバッグコンフィギュレーション (接続タブ)

13. 接続ブラウザ・ウィンドウで、目的の USB-Blaster™（この例では DE-SoC on localhost）をハイライトして、[選択] をクリックします。



図 5-16. デバッグケーブルの選択

14. デバッガタブのターゲット初期化デバッグスクリプト(.ds/.py)に指定されている semihost_setup.ds を変更します。[ワークスペース] ボタンよりプロジェクトに含まれる semihost_cvav_setup.ds を選択し、[OK] をクリックします。

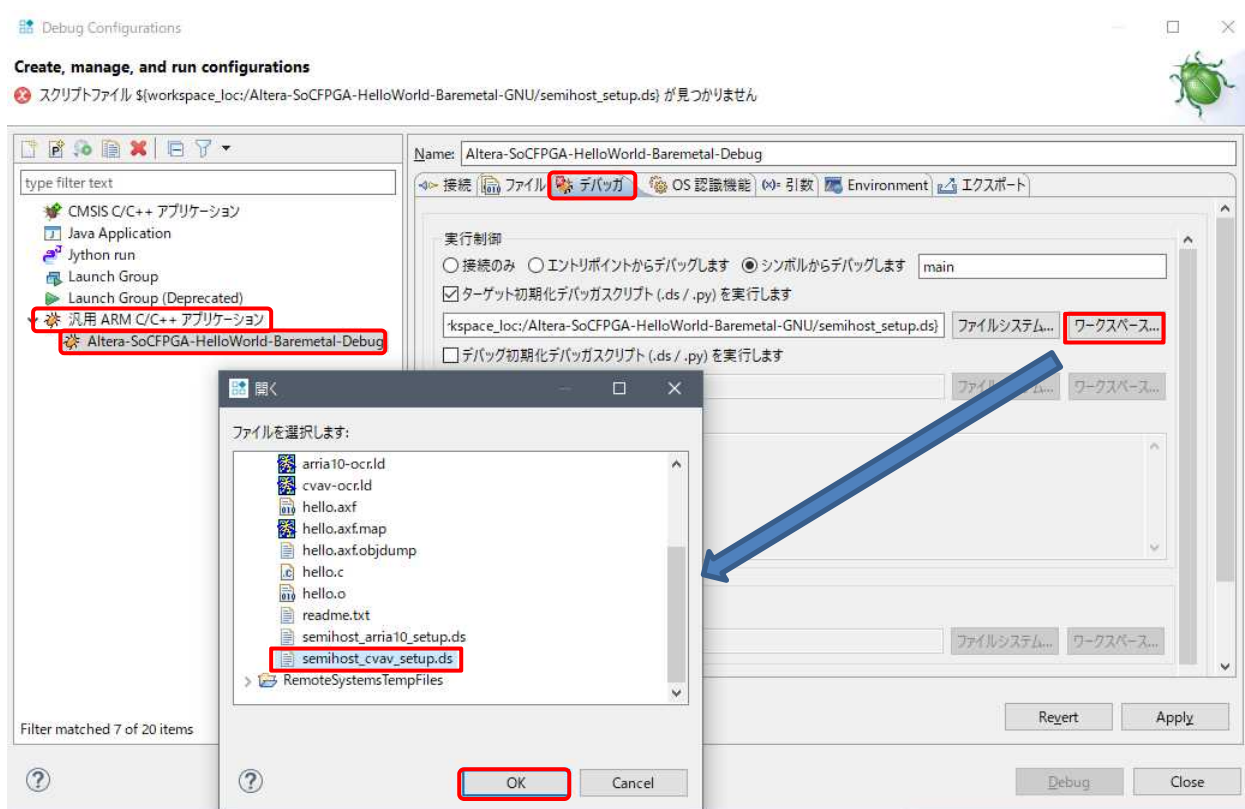


図 5-17. デバッグコンフィギュレーション(デバッガタブ)

15. デバッグコンフィギュレーション・ウィンドウの右下にある [Debug] ボタンをクリックします。



図 5-18. デバッグの実行

16. パースペクティブスイッチの確認が表示された場合は [Yes] をクリックしてそれを受け入れてください。

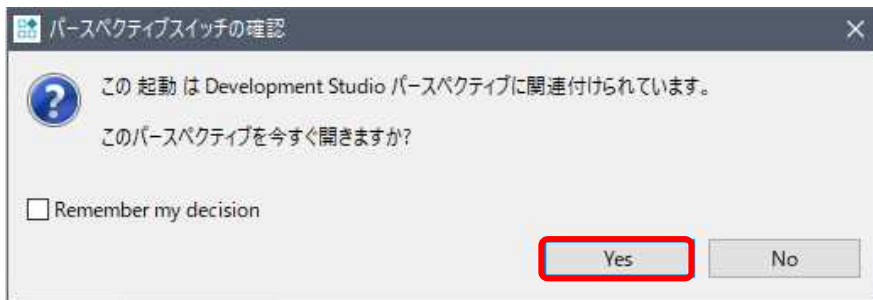


図 5-19. パースペクティブスイッチの確認

Windows Defender ファイアウォールの警告が出た場合は、[アクセスを許可する (A)] をクリックします。



図 5-20. セキュリティの警告

Note:

ダウンロード時にエラーが発生した場合は、以下の確認を行ってください。

- (1) Arm® DS のライセンスが紐づけられているネットワーク・インターフェイス（例えば USB-Ethernet Interface アダプター）が有効になっているか確認してください。
- (2) 評価ボードの電源入切および PC の再起動で復旧しないか確認してください。評価ボードの電源を切った場合は、再度 FPGA のデータをダウンロードすることを忘れないでください。

デバッガは起動スクリプトの指示に従いセミホスティング機能を有効にした後、JTAG を経由してアプリケーションをボードにダウンロードします。プログラムカウンタが main 関数に到達するとブレークされデバッグが開始出来る状態となります。この段階で、Arm® DS のすべてのデバッグ機能を使用することができます（レジスタや変数の表示と編集、逆アセンブリコードの参照、など）。

- ___ 17. 緑色の **Continue** ボタン  をクリック（または F8 キーを押して）アプリケーションを実行します。これにより、**App Console**（アプリケーション・コンソール）に **Hello Tim** メッセージを表示します

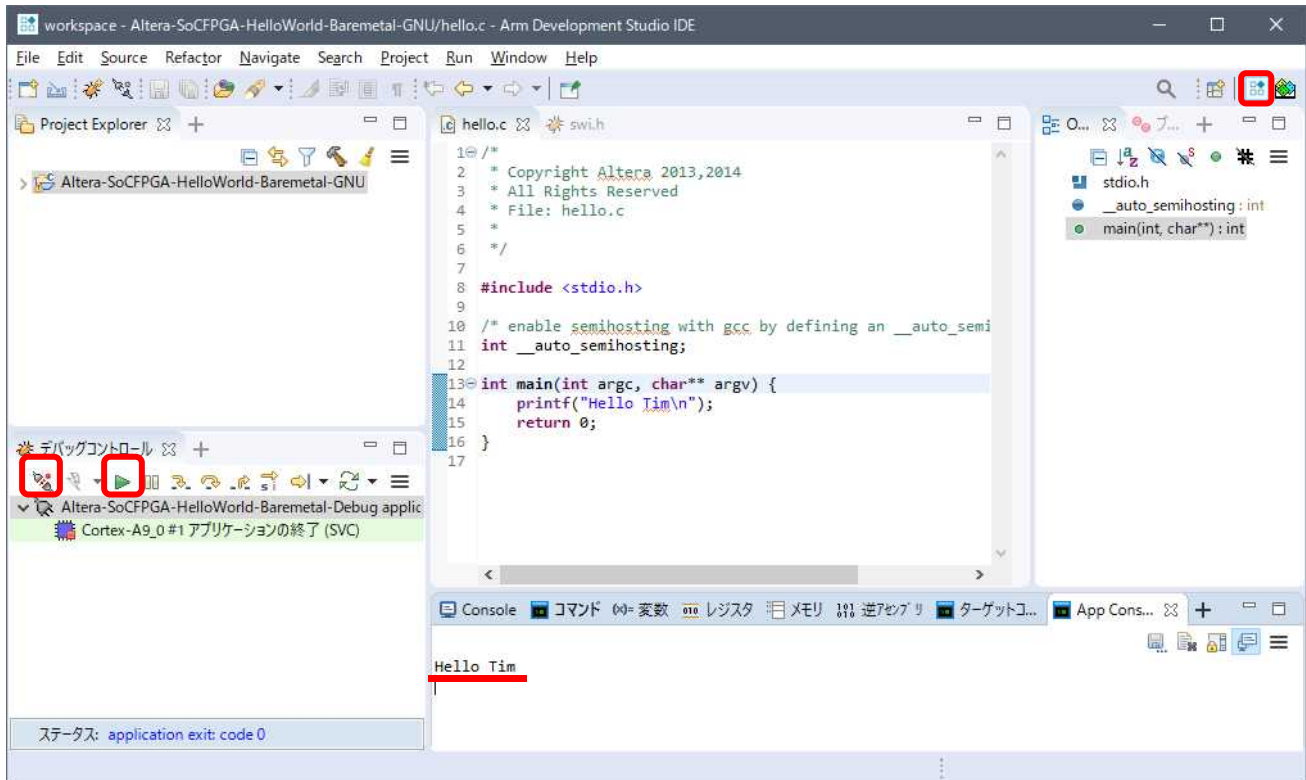


図 5-21. Hello Tim の表示

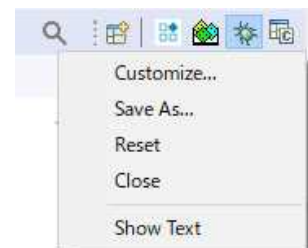
- ___ 18. 接続の切断ボタン  をクリックし CPU との接続を切断します。

- ___ 19. 画面右上のパースペクティブを変更している場合は  をクリックしメイン画面に戻ります。

① **Note:**

Arm® DS ウィンドウ右上のパースペクティブ・メニューを使用することで、複数の異なる画面配置を記憶させておくことができます。各パースペクティブは作業内容に応じて切り替えて使用することが可能です。

 ボタンより、新しいパースペクティブを開きます。パースペクティブは右上にアイコン形式でリストされるので、アイコンをクリックすることでパースペクティブの切り替えが行えます。また、選択中のパースペクティブを右クリックすることで、初期状態へ戻す (Reset) 等の操作も可能です。



5-3. LED Blink サンプル・アプリケーションの実行

Hello World サンプル・アプリケーションと同様に事前に用意された LED Blink サンプル・アプリケーションを Arm® DS にインポートします。

___ 1. Arm® DS のメニューから「File」⇒「Import」を選択します。

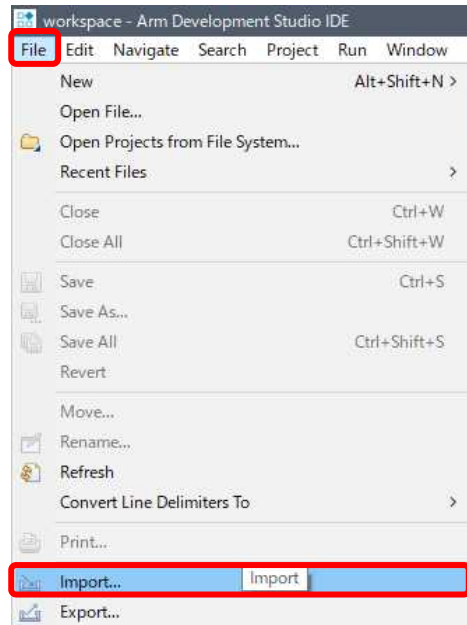


図 5-22. 「インポート」メニュー

___ 2. 「General (一般)」⇒「Existing Projects into Workspae (既存プロジェクトをワークスペースへ)」を選択し [Next] をクリックします。

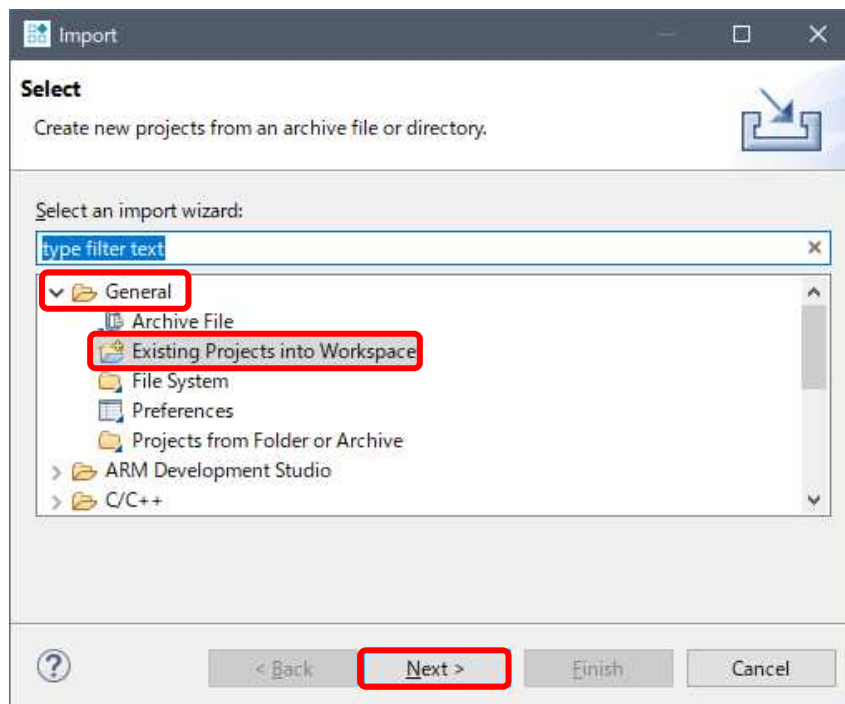


図 5-23. 既存プロジェクトのインポート

3. 「**Select archive file:** (アーカイブ・ファイルの選択)」オプションを選択し、[**Browse**] ボタンよりサンプル・プロジェクトを指定します。

C:\lab\soc_lab\cv_soc_lab\software_example\Atlas-Blinking-LED-Baremetal-GNU.tar.gz

Note:

これはツールのインストール・ディレクトリーではなく 演習データのディレクトリー以下である ことに注意してください。

選択後、[**Finish**] ボタンを押します。

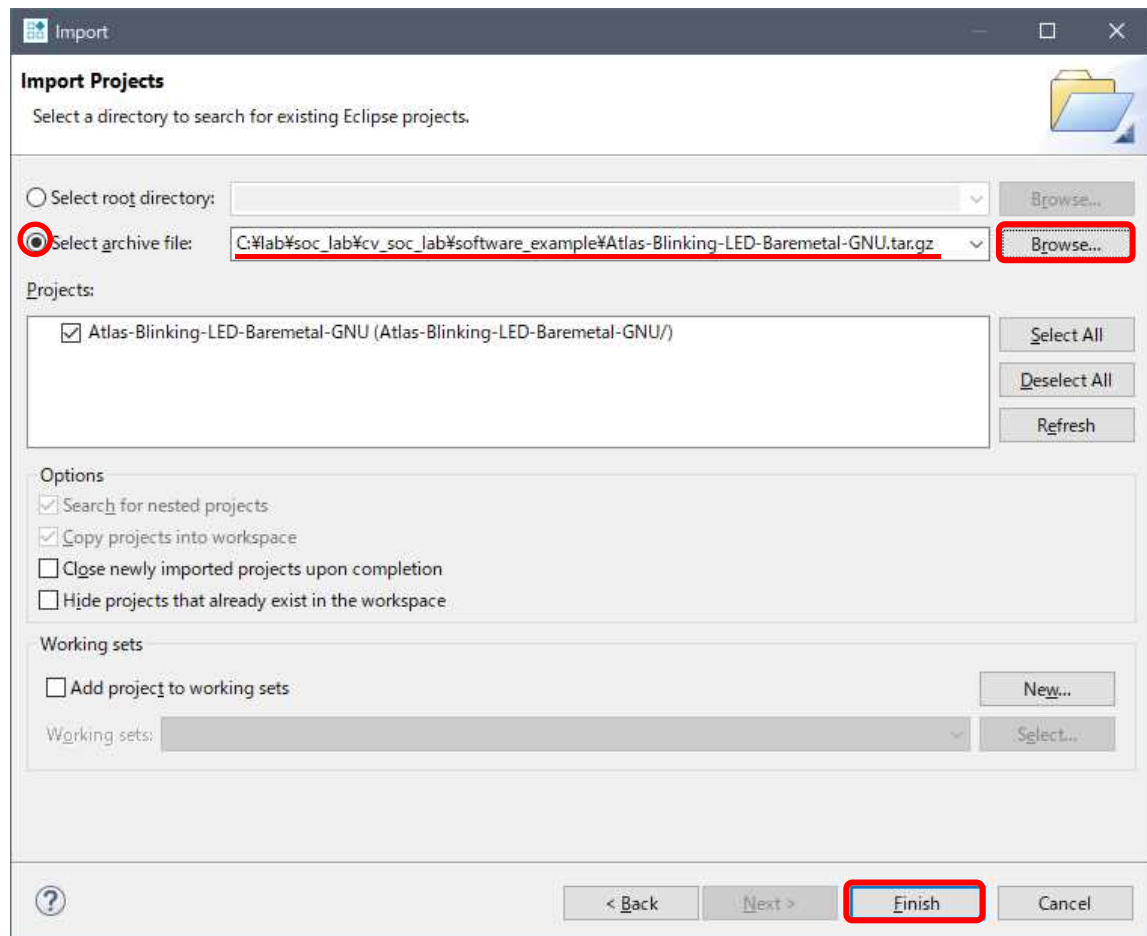


図 5-24. LED Blink サンプル・アプリケーションの選択

この作業を実施すると、Arm® DS ウィンドウ左側の プロジェクト・エクスプローラーにプロジェクトに含まれる各種ファイルが表示されます。

次に、LED Blink サンプル・アプリケーションをコンパイルします。

- ___ 4. プロジェクト・エクスプローラー タブより Atlas-Blinking-LED-Baremetal-GNU プロジェクトを選択しハイライトします。
- ___ 5. Arm® DS のメニューから「**Project**」⇒「**Build Project**」を選択します。もしくは、プロジェクト・エクスプローラー上でプロジェクトを選択し、**右クリック** ⇒「**Build Project**」を実行します。

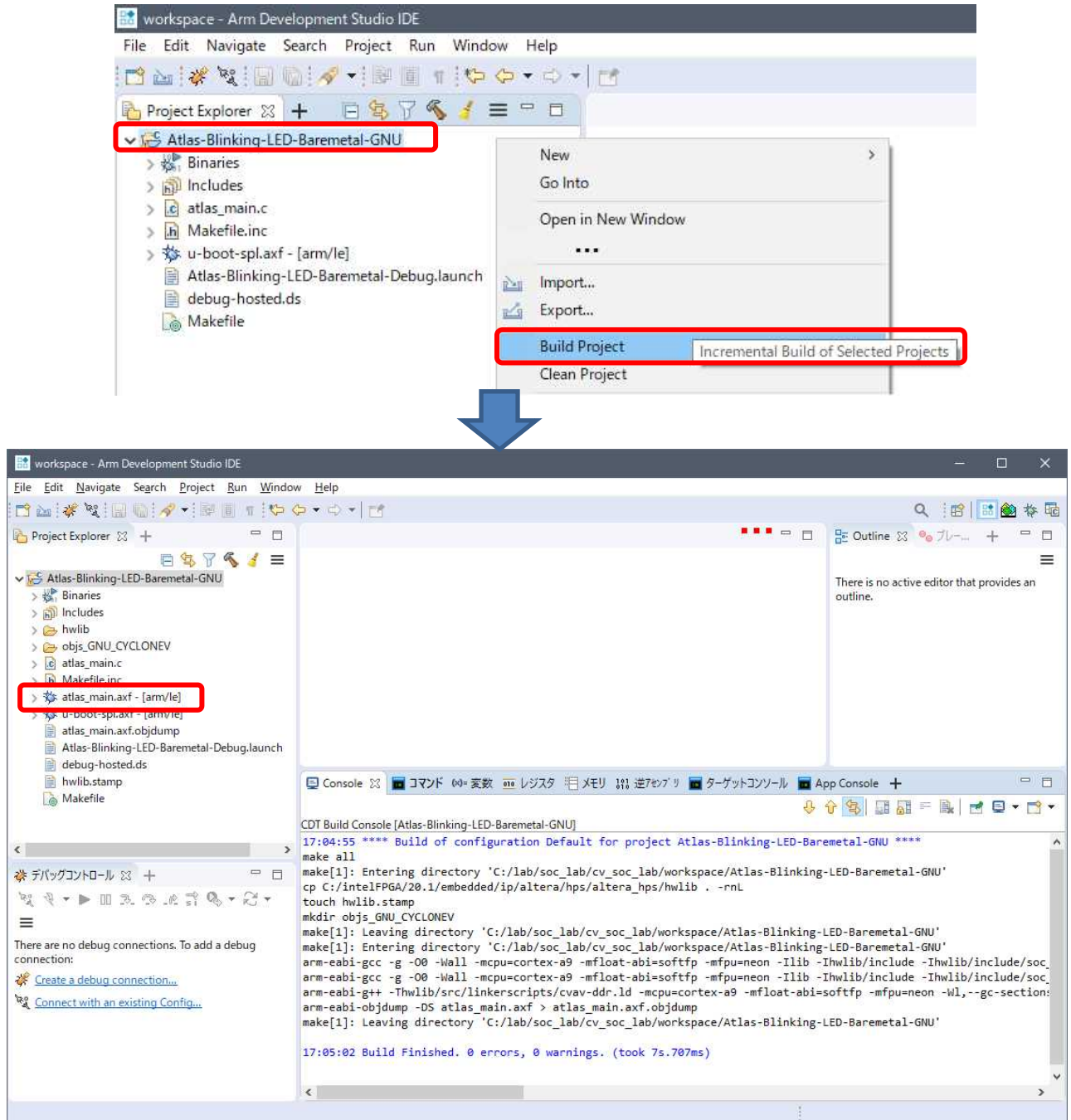


図 5-25. LED Blink サンプル・アプリケーションのビルド

最後に、LED Blink サンプル・アプリケーションを実行します。

___ 6. 「Run」メニュー ⇒ 「デバッグコンフィギュレーション(B)」を選択します。サンプル・プロジェクトには、Atlas-SoC ボード上で実行するための事前設定を付属しています。

___ 7. デバッグコンフィギュレーション (Debug Configurations) ウィンドウにある左側のパネルから、
汎用 ARM C/C++ アプリケーション ⇒ Atlas-Blinking-LED- Baremetal-Debug を選択します (表示されない場合は、汎用 ARM C/C++ アプリケーションの横にある (>) をクリックしてください)。

ターゲット接続は、USB-Blaster™ を利用し、

Intel SoC FPGA ⇒ Cyclone V SoC (Dual Core) ⇒ Bare Metal Debug ⇒ Debug Cortex-A9_0 となるように既に設定されています。

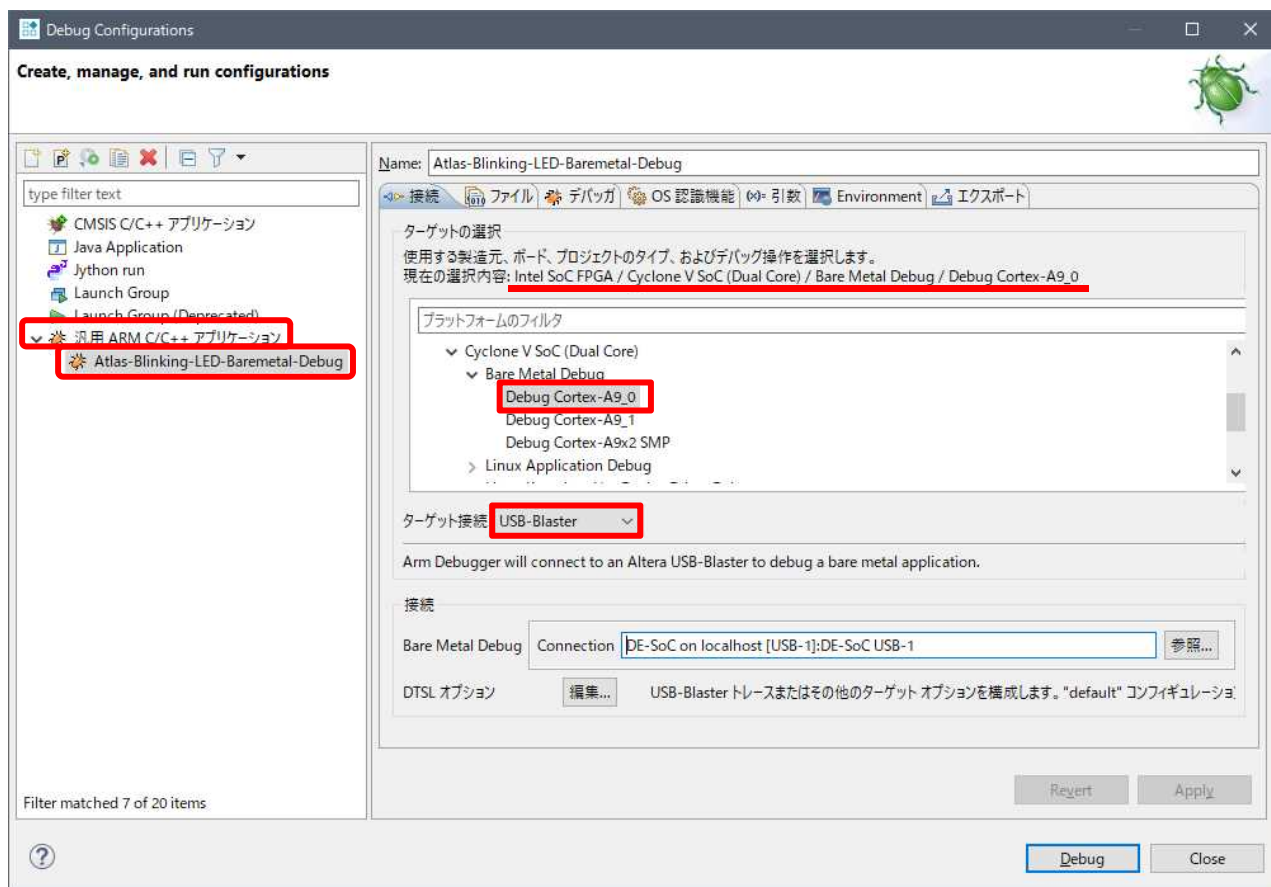


図 5-26. LED Blink サンプル・アプリケーションのデバッグ構成

___ 8. 以下の確認ポップアップが出た場合は、[はい] を選択してください。

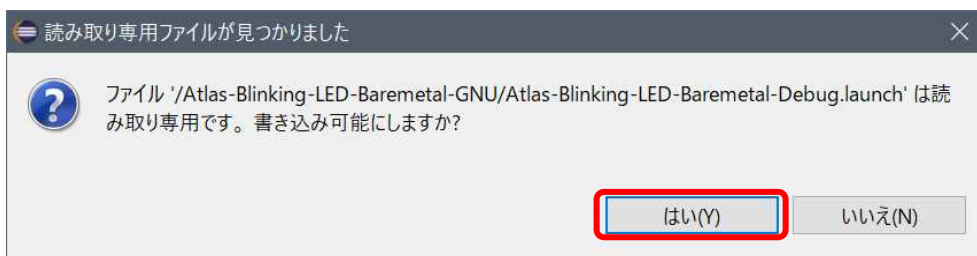


図 5-27. 確認ポップアップ

9. 接続セクションの右側にある [参照] ボタンを押下し、USB-Blaster™ 接続を選択します。
10. 接続ブラウザ・ウィンドウで、目的の USB-Blaster™ (この例では DE-SoC on localhost) をハイライトして、[選択] をクリックします。

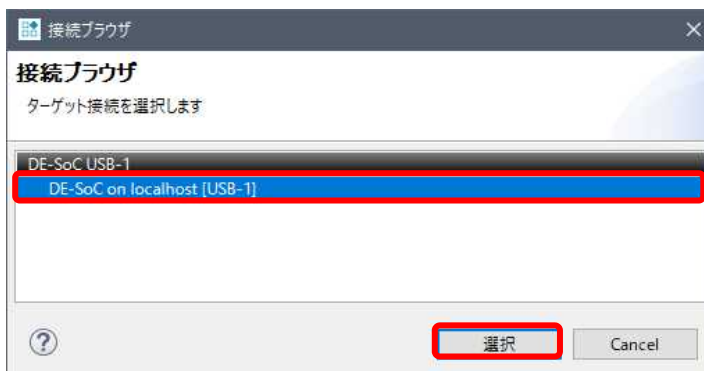


図 5-28. デバッグケーブルの選択

11. デバッグコンフィギュレーション・ウィンドウの右下にある [Debug] ボタンをクリックします。

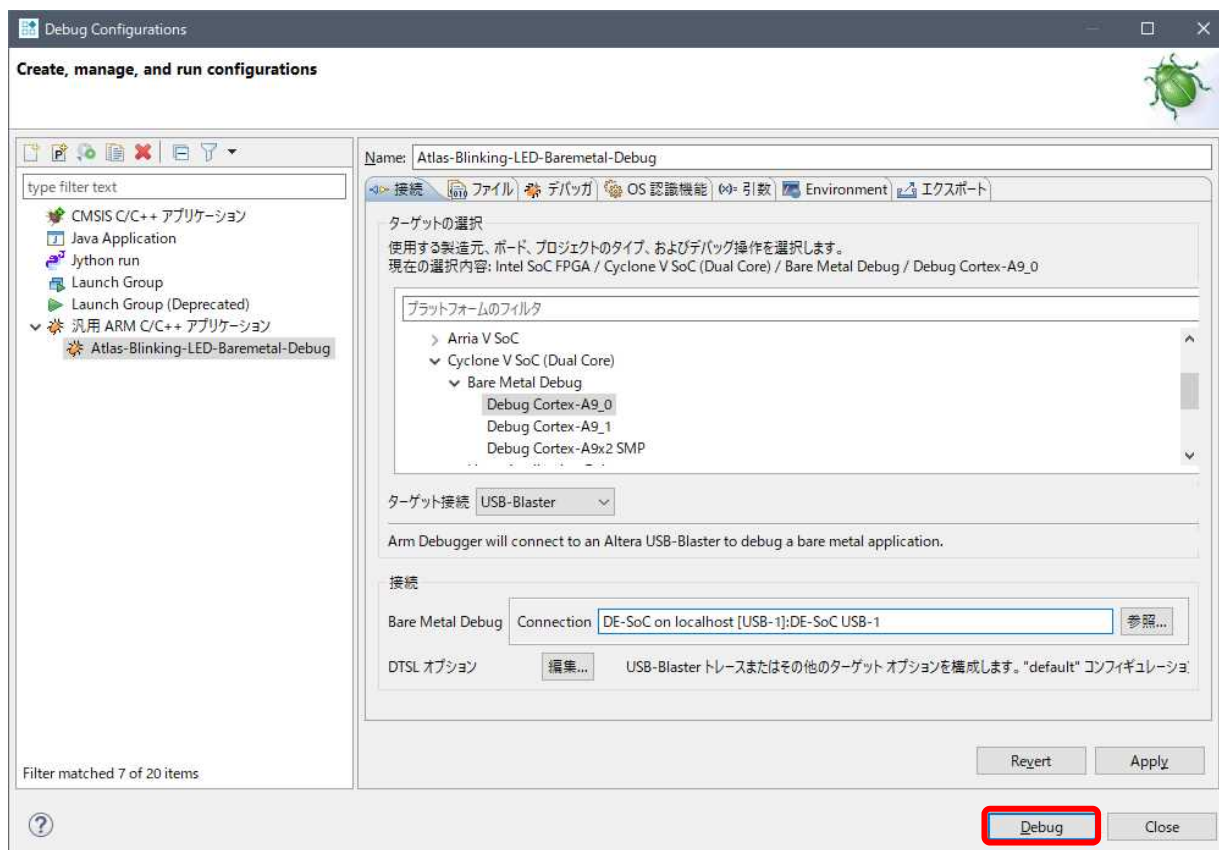


図 5-29. LED Blink サンプル・アプリケーションのデバッグ

12. パースペクティブスイッチの確認が表示された場合は「Yes」をクリックしてそれを受け入れてください。

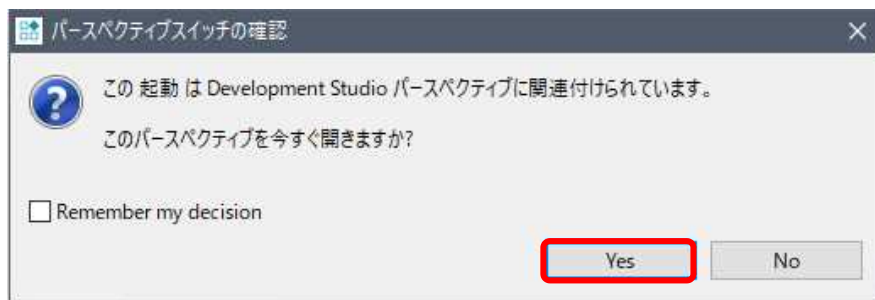


図 5-30. パースペクティブスイッチの確認

Windows Defender ファイアウォールの警告が出た場合は、「アクセスを許可する (A)」をクリックします。

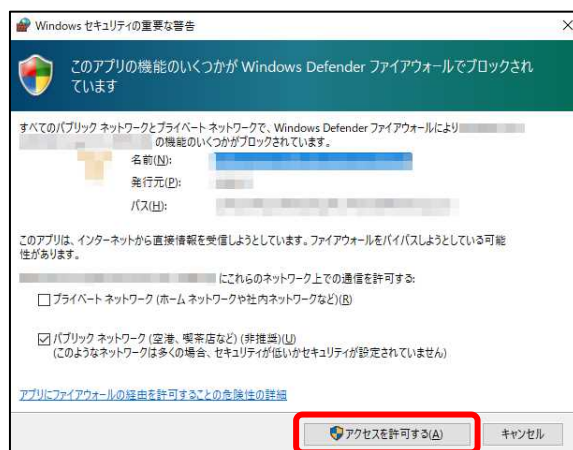


図 5-31. セキュリティの警告

Note:

ダウンロード時にエラーが発生した場合は、以下の確認を行ってください。

- (1) Arm® DS のライセンスが紐づけられているネットワーク・インターフェイス（例えば USB-Ethernet Interface アダプター）が有効になっているか確認してください。
- (2) 評価ボードの電源入切および PC の再起動で復旧しないか確認してください。評価ボードの電源を切った場合は、再度 FPGA のデータをダウンロードすることを忘れないでください。

13. ブレークポイントを設定します。

Atlas_main.c の 26 行目にブレークポイントを設定します。行数表示の左横のスペースをダブルクリックすることで設定可能です。

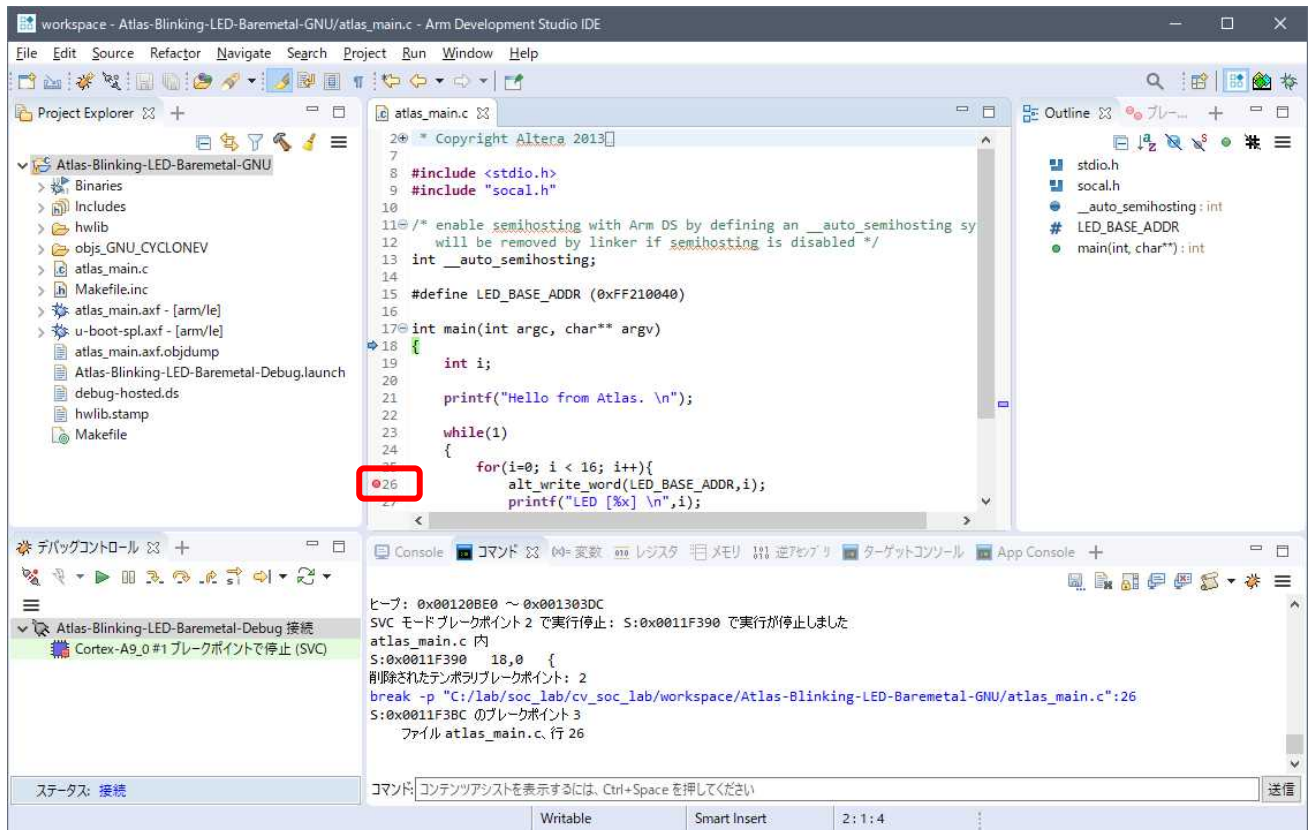


図 5-32. ブレークポイントの設定

14. 緑色の **Continue** ボタン  をクリックして(または F8 キーを押して)アプリケーションを実行します。これにより、**App Console** (アプリケーション・コンソール) に **Hello from Atlas.** メッセージが表示されます。

15. もう 2 回、緑色の **Continue** ボタン  をクリックして(または F8 キーを押して)アプリケーションを実行します。これにより、**アプリケーション・コンソール** に **LED [0]** メッセージが表示され、Atlas-SoC ボード上のユーザー LED (LED [3:0]) の点灯状態が変化することを確認します。

16. さらに  を押すごとに LED の状態が変化することを確認してください。

17. 接続の切断ボタン  をクリックし CPU との接続を切断します。

以上で 演習 3 は完了です。お疲れ様でした。

次のページ以降にオプション演習があります。時間がある方は、こちらも実施してみてください。

5-4. 演習 2 で作成した Preloader による初期化 (オプション演習)

演習 3 では、前もって準備されていた Preloader を使用して HPS を初期化していました。

ここでは、「[演習 2: ソフトウェア演習\(1\) Preloader の生成](#)」で作成した Preloader にて HPS の初期化を実施します。

1. 演習 2 で Preloader イメージが作成されていることを確認します。

Preloader は、C:\lab\soc_lab\cv_soc_lab\software\spl_bsp\uboot-socfpga\spl ディレクトリーの下に、“u-boot-spl”という名で作成されているはずです。このファイルが生成されていることを確認してください。また、Preloader 本体とは別に、同じディレクトリーにデバイスツリー “u-boot-spl.dtb” が生成されていることも確認してください。

もし、生成されていない場合は、再度 演習 2 を実施してください。

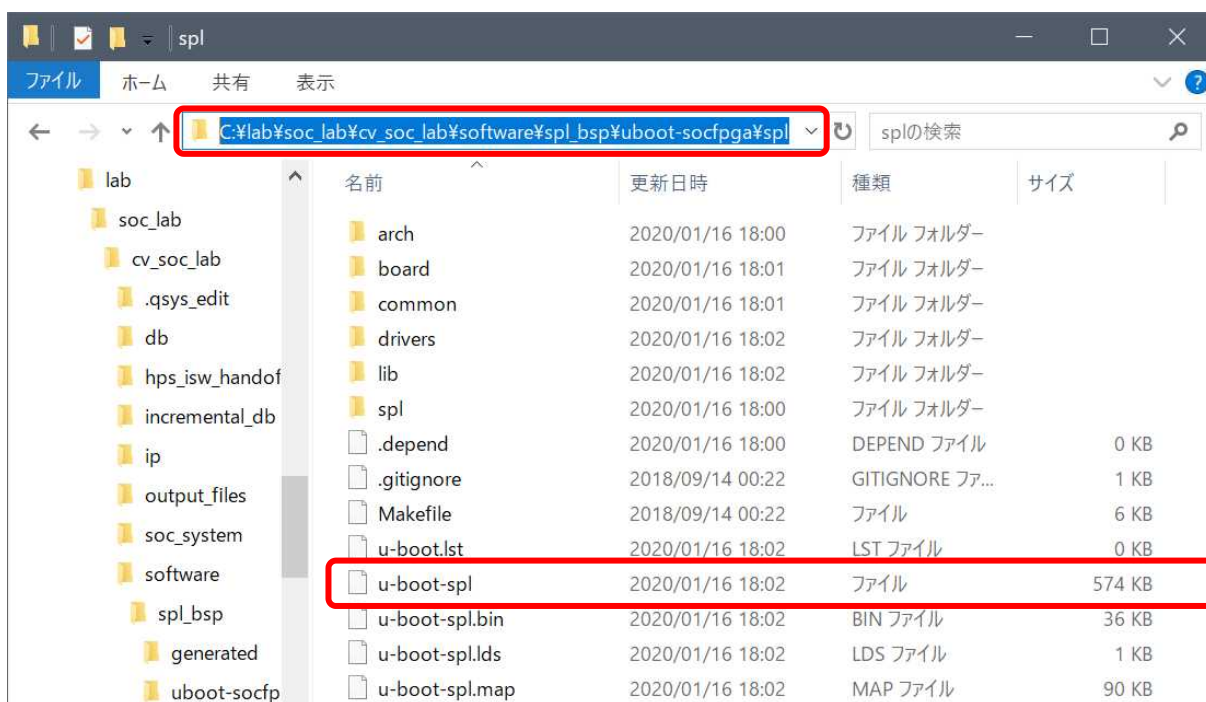


図 5-33. U-boot-spl ファイルの確認

① Note:

ここで確認した “u-boot-spl” ファイルは、Arm® Executable and Linkable Format (ELF) ファイルです。Arm® DS の初期化スクリプトにて読み出され、ユーザー・アプリケーション実行前に実行されています。詳細は、「SoC はじめてガイド – DS-5 によるベアメタル・アプリケーション・デバッグ」の「カスタム・ボードへの対応方法」をご参照ください(参考記事は DS-5™ 向けの内容となりますが、Arm® DS においても考え方は同様となります)。

■ 参考: [SoC はじめてガイド – DS-5 によるベアメタル・アプリケーション・デバッグ](#)

___ 2. 演習 3 で使用した “u-boot-spl.axf” ファイルをリネームします。

C:\lab\soc_lab\cv_soc_lab\workspace\Atlas-Blinking-LED-Baremetal-GNU に “u-boot-spl.axf” ファイルがありますので、このファイルを “_u-boot-spl.axf” 等に変更します。

___ 3. 演習 2 で作成した “u-boot-spl” および “u-boot-spl.dtb” をコピーします。

C:\lab\soc_lab\cv_soc_lab\software\spl_bsp\uboot-socfpga\spl ディレクトリーの下にある、“u-boot-spl” ファイルと “u-boot-spl.dtb” ファイルを、C:\lab\soc_lab\cv_soc_lab\workspace\Atlas-Blinking-LED-Baremetal-GNU ディレクトリーにコピーします。

___ 4. コピーした “u-boot-spl” ファイルを “u-boot-spl.axf” という名前にリネームします。

ここまでの作業で、デバッグ時に使用する Preloader が変更されました。

実際に動作するか確認していきます。

___ 5. LED Blink サンプル・アプリケーションを再度実行します。

75 ページの “___ 6. 「Run」メニュー ⇒ 「デバッグコンフィギュレーション(B)」を選択します。サンプル・プロジェクトには、Atlas-SoC ボード上で実行するための事前設定を付属しています。” から実行してください。

先程と同様に LED Blink サンプル・アプリケーションが実行できれば、演習 2 で作成した Preloader が正常に動作したことになります。

5-5. システム・ヘッダーファイルによるアドレスの解決 (オプション演習)

演習 3 の LED Blink サンプル・アプリケーションでは、LED PIO のアドレスをソースコード上で直接指定していました。

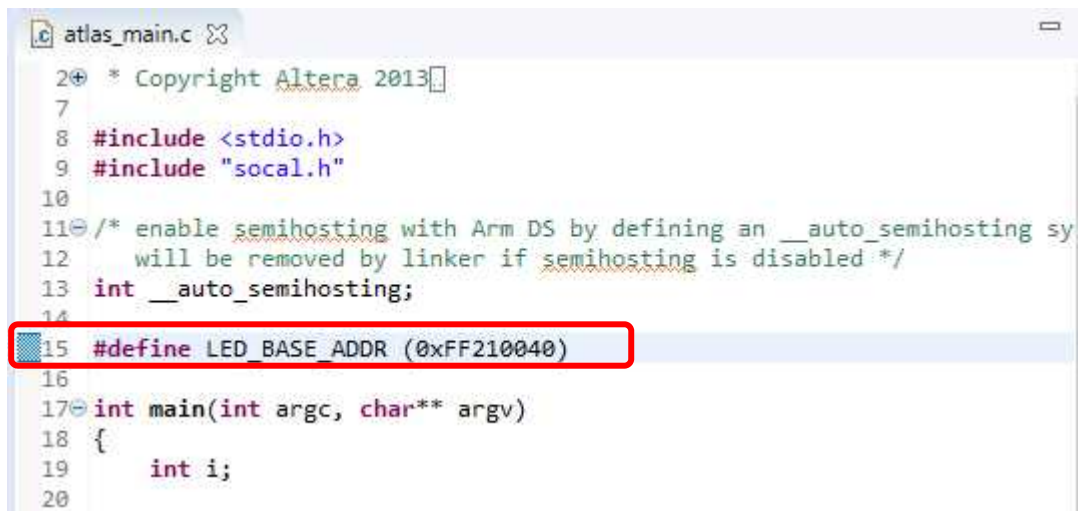


図 5-34. 今までのアドレス指定方法

ここでは、SoC EDS のシステム・ヘッダーファイル生成コマンド (sopc-create-header-files) を使用してシステム・ヘッダーファイルを生成し使用してみましょう。

- ___ 1. Embedded Command Shell が起動していない場合は起動します。
- ___ 2. C:¥lab¥soc_lab¥cv_soc_lab に移動します。

\$ cd "C:¥lab¥soc_lab¥cv_soc_lab"



図 5-35. ディレクトリーの移動

3. Embedded Command Shell 上で、システム・ヘッダーファイル生成コマンド (sopc-create-header-files) を実行します。

```
$ socp-create-header-files soc_system.sopcinfo
```

図 5-36. システム・ヘッダーファイル生成コマンドの実行 (エラー)

⚠ 注記:

ご使用の OS が Windows® 10 の場合、上の図のようなエラーが発生する場合があります。エラーとなる場合は、以下の参考情報サイトに記載の対策を行った上で再度 socp-create-header-files を実行してください。

📖 参考: アルティマ技術サポート「[SoC EDS 環境で socp-create-header-files が実行エラーになるトラブルの回避策](#)」

図 5-37. システム・ヘッダーファイル生成コマンドの実行 (成功)

5 つのファイルが生成されたことを確認します。

- Soc_system.h : Platform Designer 内のすべてのマスターに対するモジュール情報を定義
- hps_0.h : HPS の各ブリッジ (H2F, LWH2F) に接続されているモジュール情報を定義
- hps_0_bridges.h : HPS の各ブリッジ (F2H, H2F, LWH2F) に接続されているモジュール情報を定義
- hps_0_arm_a9_0.h: hps_0_arm_a9_0 向けのモジュール情報を定義。各ブリッジのオフセットも付加されている
- hps_0_arm_a9_1.h: hps_0_arm_a9_1 向けのモジュール情報を定義。各ブリッジのオフセットも付加されている

ここでは hps_0_arm_a9_0.h を使用します。

___ 4. システム・ヘッダーファイルを LED Blink サンプル・アプリケーション・プロジェクトにコピーします。

ファイル名: hps_0_arm_a9_0.h

コピー元: C:\lab\soc_lab\cv_soc_lab

コピー先: C:\lab\soc_lab\cv_soc_lab\workspace\Atlas-Blinking-LED-Baremetal-GNU

___ 5. LED Blink サンプル・アプリケーションのソースコード atlas_main.c を変更します。

変更時に「書き込み可能にしますか？」というポップアップが表示される場合は「はい」を選択します。

記述追加:

```
#include "hps_0_arm_a9_0.h"
```

記述変更:

<変更前> #define LED_BASE_ADDR (0xFF210040)

<変更後> #define LED_BASE_ADDR LED_PIO_BASE

以下の図では、比較しやすいように以前の LED_BASE_ADDR 記述をコメントアウトしてあります。

参考までに、“hps_0_arm_a9_0.h” の該当箇所も図示します。

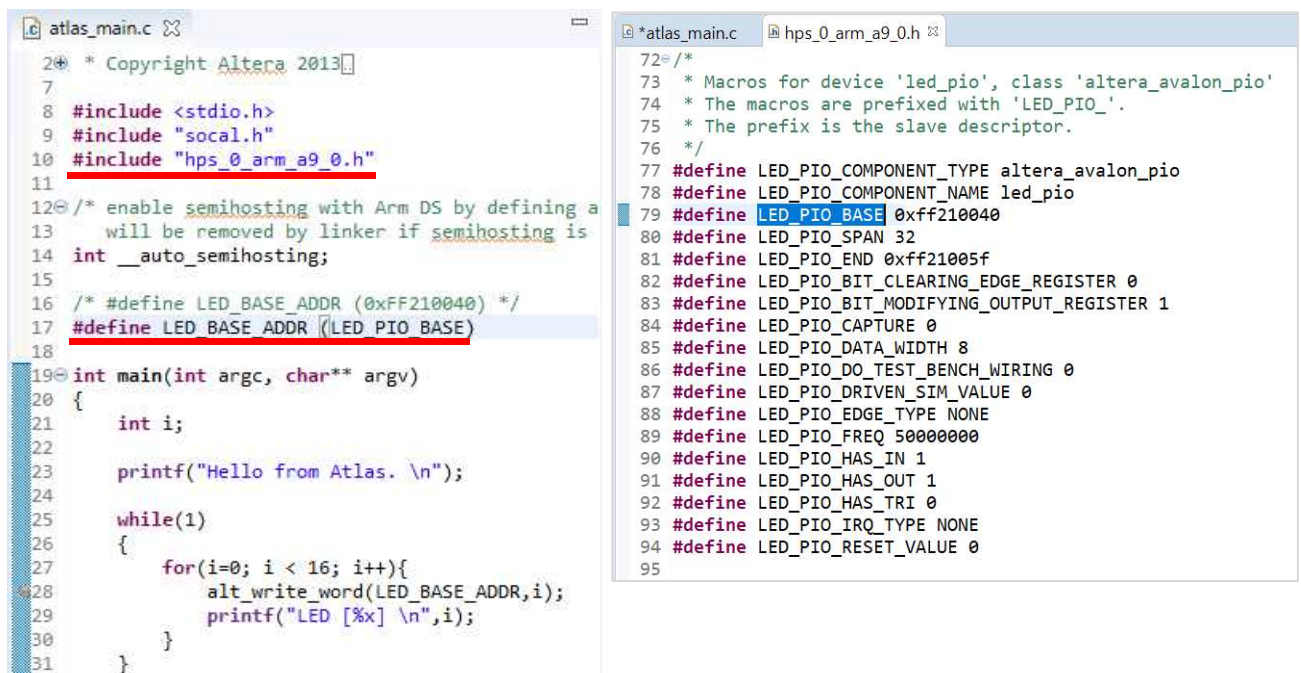


図 5-38. ソースコードの変更箇所とシステム・ヘッダーファイルの該当箇所

___ 6. 変更した atlas_main.c をセーブして、LED Blink サンプル・アプリケーションをビルドします。

___ 7. ビルド後、LED Blink サンプル・アプリケーションを実行し、演習 3 と同様の結果となることを確認します。

以上で 演習 3 (オプション) は完了です。

6. 演習 4: Linux アプリケーション演習 (オプション演習)

この演習では Arm® DS 上から Linux のアプリケーションのひとつとして用意されている Hello World を実行、デバッグします。

① Note:

この演習では、弊社がお貸し出しする「SoC FPGA Seminar in a Box」をご利用のお客様は、同梱されている microSD カードを使用します。

この microSD カードには Linux OS を起動するためのデザインが入っています。

SoC FPGA Seminar in a Box 以外でこの演習を実行されるお客様は、以下の「6-1. microSD カードの準備」の手順により、ご自身で microSD カードをご用意ください。

6-1. microSD カードの準備

SoC FPGA Seminar in a Box に同梱されている microSD カードを使用する場合は、このセクションはとばして次の「6-2. Linux 起動とログイン」に進んでください。ご自身で microSD カードを書き込む場合は以下の手順で行ってください。

- ___ 1. 下記のサイトから使用するボード向けの SD カード・イメージファイルをダウンロードします。
ダウンロードしたファイルは任意のフォルダーに解凍しておきます。解凍したフォルダー内に .img イメージファイルがあることを確認します。
 - [Atlas-SoC ボード向け SD カード・イメージファイル](#)
 - [DE10-Nano ボード向け SD カード・イメージファイル](#)
- ___ 2. Windows® をご使用の場合、SD カード・イメージファイルの書き込みには汎用のソフトウェアを利用します。
ここでは Win32DiskImager を紹介します。以下よりダウンロード可能です。
 - [Win32DiskImager](#)
- ___ 3. microSD カード (8GB 以上を推奨) を PC の SD カードスロットに挿入します (または USB カードリーダー／ライターを使用します)。microSD カードに割り当てられたドライブ (この例では、ドライブ E) を確認します。

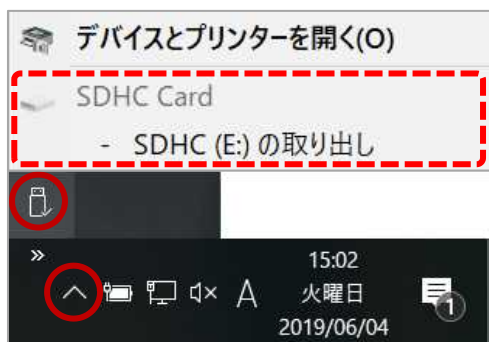


図 6-1. microSD カードに割り当てられたドライブの確認

4. あらかじめ PC にインストールしておいた Win32DiskImager を起動します。

- ① Device として PC に挿入した microSD カードのドライブが選択されていることを確認します。
- ② 先ほど解凍した SD カード・イメージファイルを選択して開きます。
- ③ [Write] ボタンをクリックして イメージファイルを書き込みます。
- ④ 書き込みが完了したら [OK] ボタンをクリックします。
- ⑤ [Exit] ボタンをクリックして Win32DiskImager を終了します。

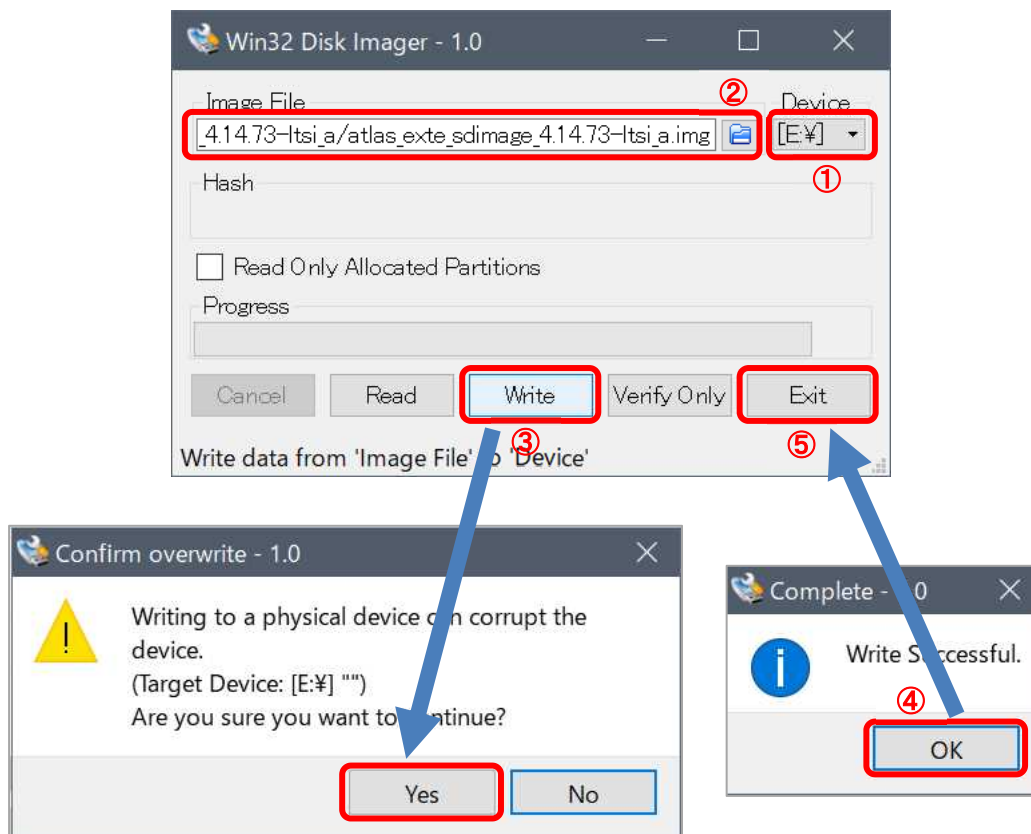


図 6-2. Win32DiskImager

5. PC から microSD カードを安全に取り外します。

⚠ 注記:

ホスト PC の OS が Windows® 10 の場合、SD カードの書き込みの際に、カード内に FAT 以外のパーティション（ボリューム）が存在している場合は、以下の現象が発生することがあります。

- カード挿入時に警告ウィンドウが表示される
- SD カードイメージの書き込みに失敗する

これらの現象への対処方法については、以下の参考情報サイトの記事をご参照ください。

📄 参考:

アルティマ技術サポート「[Windows® 10 で SD カードイメージの書き込みに失敗する場合の対処法](#)」

6-2. Linux 起動とログイン

この演習では、以下のインターフェイスを使用します。

DE10-Nano ボードも基本的には同じです。

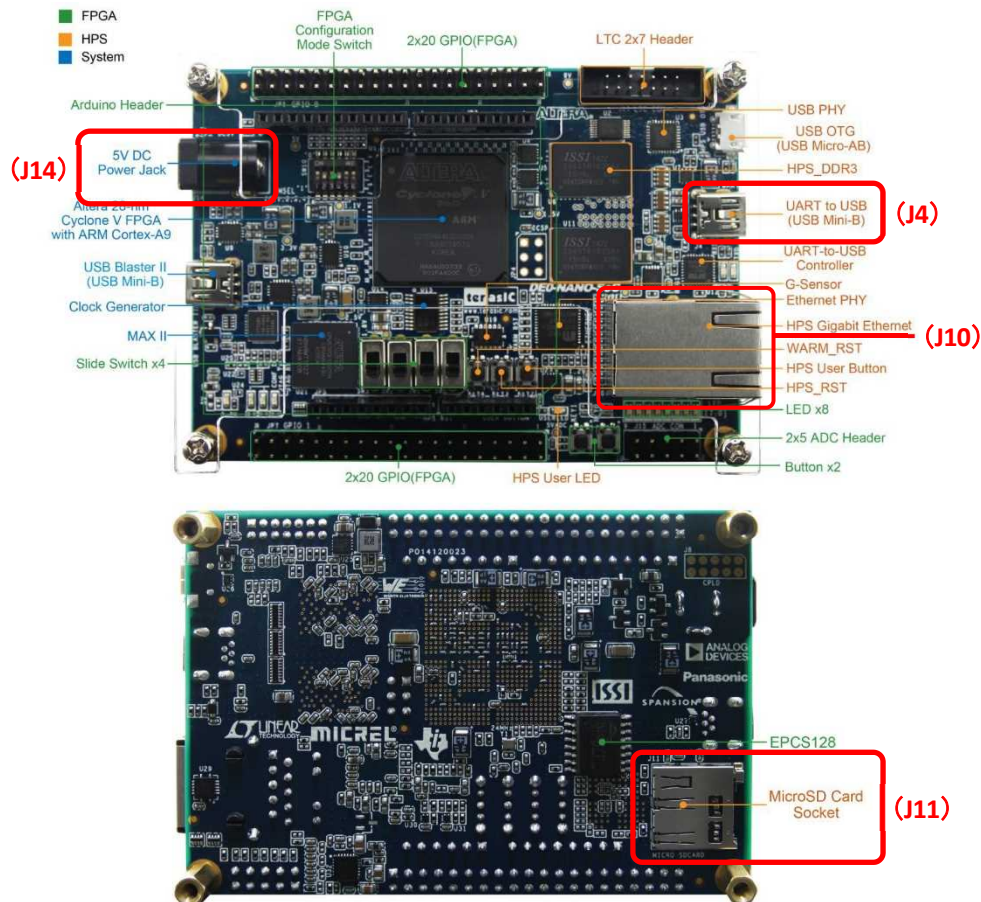


図 6-3. 本演習で使用するインターフェイス

- ___ 1. ボードの 5V DC ジャック (J14) に電源アダプターが接続されている場合は、一旦ケーブルを抜きます。
- ___ 2. ボードの UART USB コネクター (J4) へ USB Mini-B ケーブルを接続します。ケーブルの反対側のコネクターを PC の USB コネクターへ接続します。
- ___ 3. ボードの HPS Ethernet コネクター (J10) へ、イーサネット・ケーブルを接続します。ケーブルの反対側のコネクターを PC のイーサネット・コネクターへ接続します。
- ___ 4. ボード裏側の microSD カードスロット (J11) に microSD カードを挿入します。
- ___ 5. 電源アダプターケーブルをボードの 5V DC ジャック (J14) に接続し、ボードに電源を投入します。

- ___ 6. Windows® の「デバイス マネージャー」を開きます。デバイスマネージャーの「ポート (COM と LPT)」を展開してボードの UART が何番の COM ポートに接続されているかを確認します (この例では COM4)。確認できたら デバイスマネージャー を閉じます。



図 6-4. COM ポートの確認

- ___ 7. あらかじめインストールしておいたターミナルソフトを起動して、シリアルポートの設定を行います。先ほど確認した COM ポートを選択して下図のように設定します (この例では COM4)。

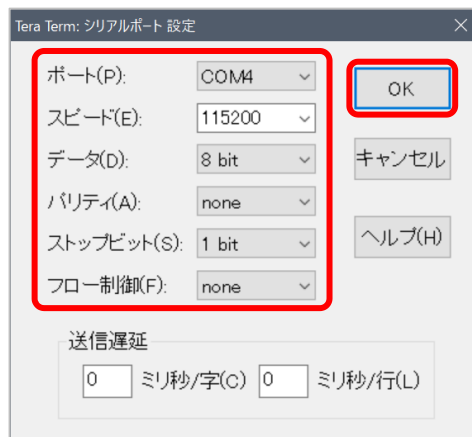


図 6-5. シリアルポートの設定

- ___ 8. ボードの WARM リセットボタン (KEY3) を押します。ターミナルに起動メッセージが表示されます。

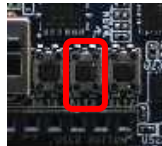


図 6-6. WARM リセットボタン (KEY3)

- ___ 9. Linux カーネルが起動したら、**root** でログインします。

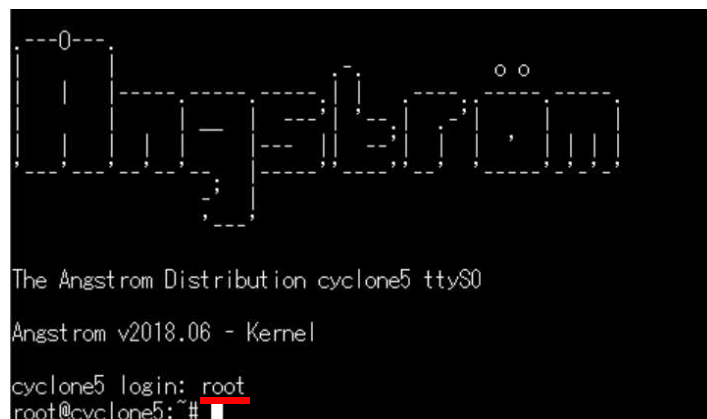


図 6-7. Root でログイン

6-3. Linux での IP アドレスとパスワードの設定

- ___ 1. ターミナルから `ifconfig` コマンドで、ボードの IP アドレスを設定します（この例では 192.168.1.30 を設定しています）。

```
# ifconfig eth0 192.168.1.30
```

- ___ 2. その後 `ifconfig` コマンドで設定内容を確認します。

```
# ifconfig eth0
```

- ___ 3. `passwd` コマンドで任意のパスワードを設定します。このパスワードは後でリモート・システムによるデバッグで使用します。

```
# passwd
```

- ___ 4. 再度パスワードを入力します。

```
The Angstrom Distribution cyclone5 ttyS0
Angstrom v2018.06 - Kernel

cyclone5 login: root
root@cyclone5:~# ifconfig eth0 192.168.1.30
root@cyclone5:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 86:7c:0f:0d:a2:91
          inet addr:192.168.1.30  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::847c:fff:fe0d:a291/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:250 errors:0 dropped:2 overruns:0 frame:0
          TX packets:116 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:16080 (15.7 KiB)  TX bytes:34449 (33.6 KiB)
          Interrupt:27 Base address:0xa000

root@cyclone5:~# passwd
New password:
Retype new password:
passwd: password updated successfully
root@cyclone5:~#
```

図 6-8. IP アドレスとパスワードの設定

6-4. ホスト PC 側のネットワーク設定

Arm® DS での リモート・システム・エクスプローラー (RSE) を使用した Linux アプリの実行・デバッグを行う上で、ホスト PC 側のネットワーク設定を行います。

- ___ 1. まずホスト PC 側の IP アドレスを設定します。「コントロール パネル」から「ネットワークと共有センター」をクリックし、左側の「アダプターの設定の変更」をクリックします。

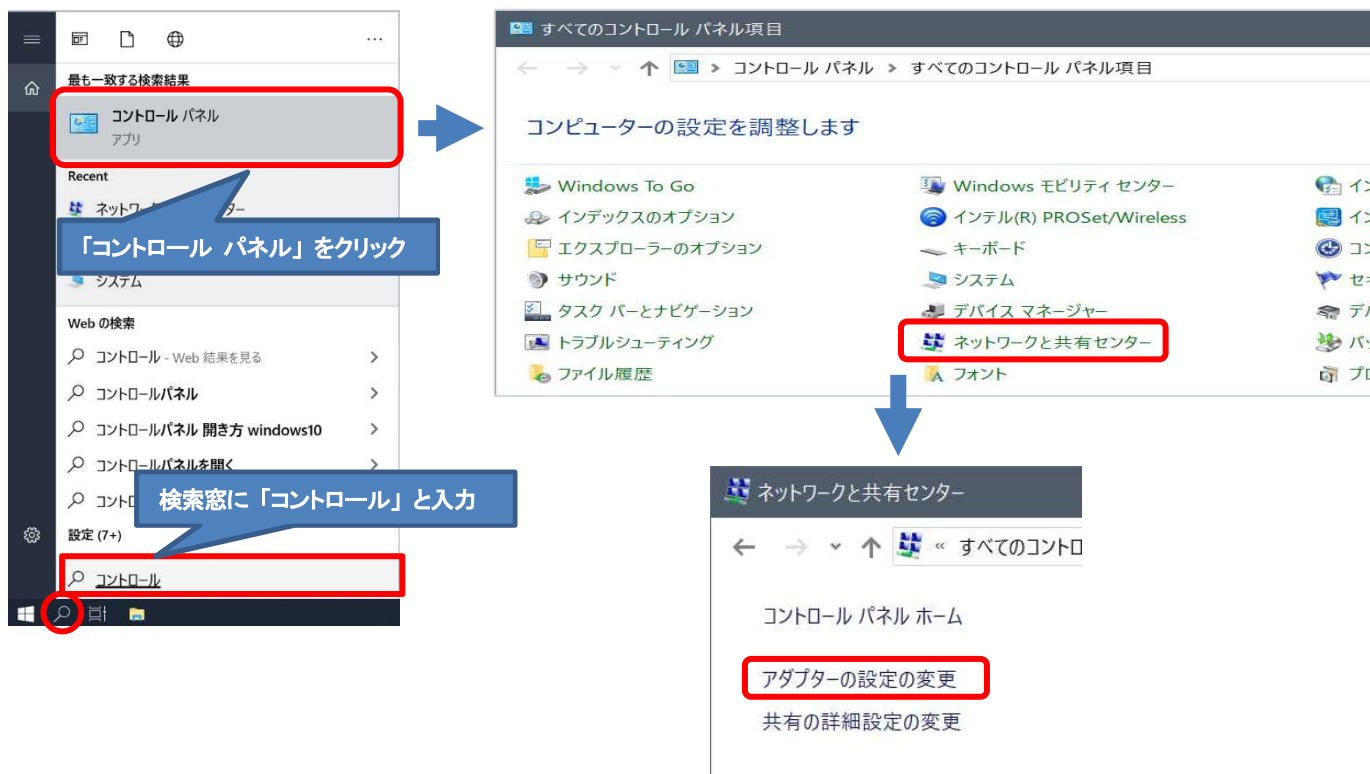


図 6-9. アダプターの設定の変更

- ___ 2. 「イーサネット」をダブルクリックします。



図 6-10. 「イーサネット」をダブルクリック

3. 「プロパティ」ボタンをクリックします。
4. 「インターネット プロトコル バージョン 4 (TCP/IPv4)」をダブルクリックします。

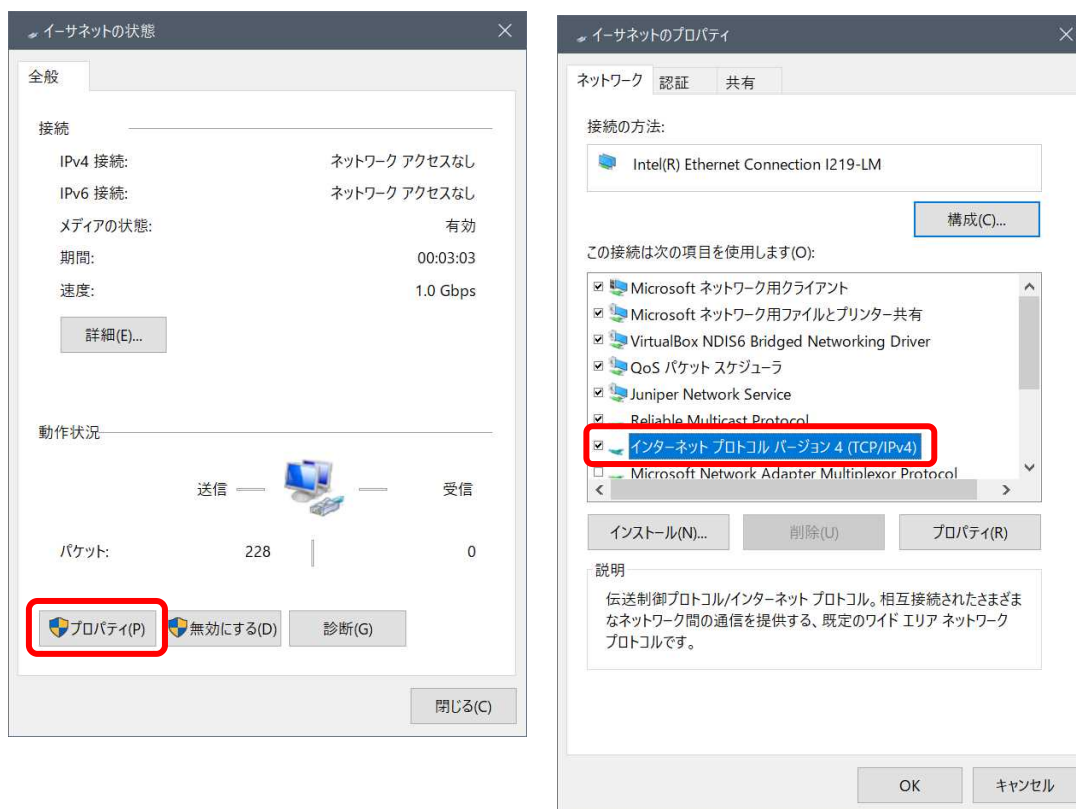


図 6-11. ローカル エリア接続のプロパティ

5. 「次の IP アドレスを使う (S):」にチェックを入れて「IP アドレス」と「サブネット マスク」を設定します（この例では、IP アドレスを 192.168.1.31、サブネット マスクを 255.255.255.0 に設定しています）。

設定後、[OK] をクリックします。



図 6-12. 「IP アドレス」と「サブネット マスク」の設定

- ___ 6. ネットワークの接続を確認します。ボードの Linux からホスト PC に対して ping を実行して接続を確認してみます（この例では、PC の IP アドレスを 192.168.1.31 に設定しています）。

```
# ping 192.168.1.31
```

- ___ 7. **Ctrl** + **C** をキー入力して ping を停止します。

```
root@cyclone5:~# ping 192.168.1.31
PING 192.168.1.31 (192.168.1.31): 56 data bytes
64 bytes from 192.168.1.31: seq=0 ttl=128 time=0.655 ms
64 bytes from 192.168.1.31: seq=1 ttl=128 time=1.026 ms
64 bytes from 192.168.1.31: seq=2 ttl=128 time=0.953 ms
64 bytes from 192.168.1.31: seq=3 ttl=128 time=0.920 ms
^C
--- 192.168.1.31 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.655/0.888/1.026 ms
root@cyclone5:~#
```

図 6-13. PC に対して ping を実行して接続を確認

- ___ 8. もし ping 応答が無い場合は Windows Defender ファイアウォール設定を確認します。
「パブリックネットワークの設定」を確認し、Windows Defender ファイアウォールが“有効”に設定されている場合は“無効”に設定して、再度 ping を実行して接続を確認してください。

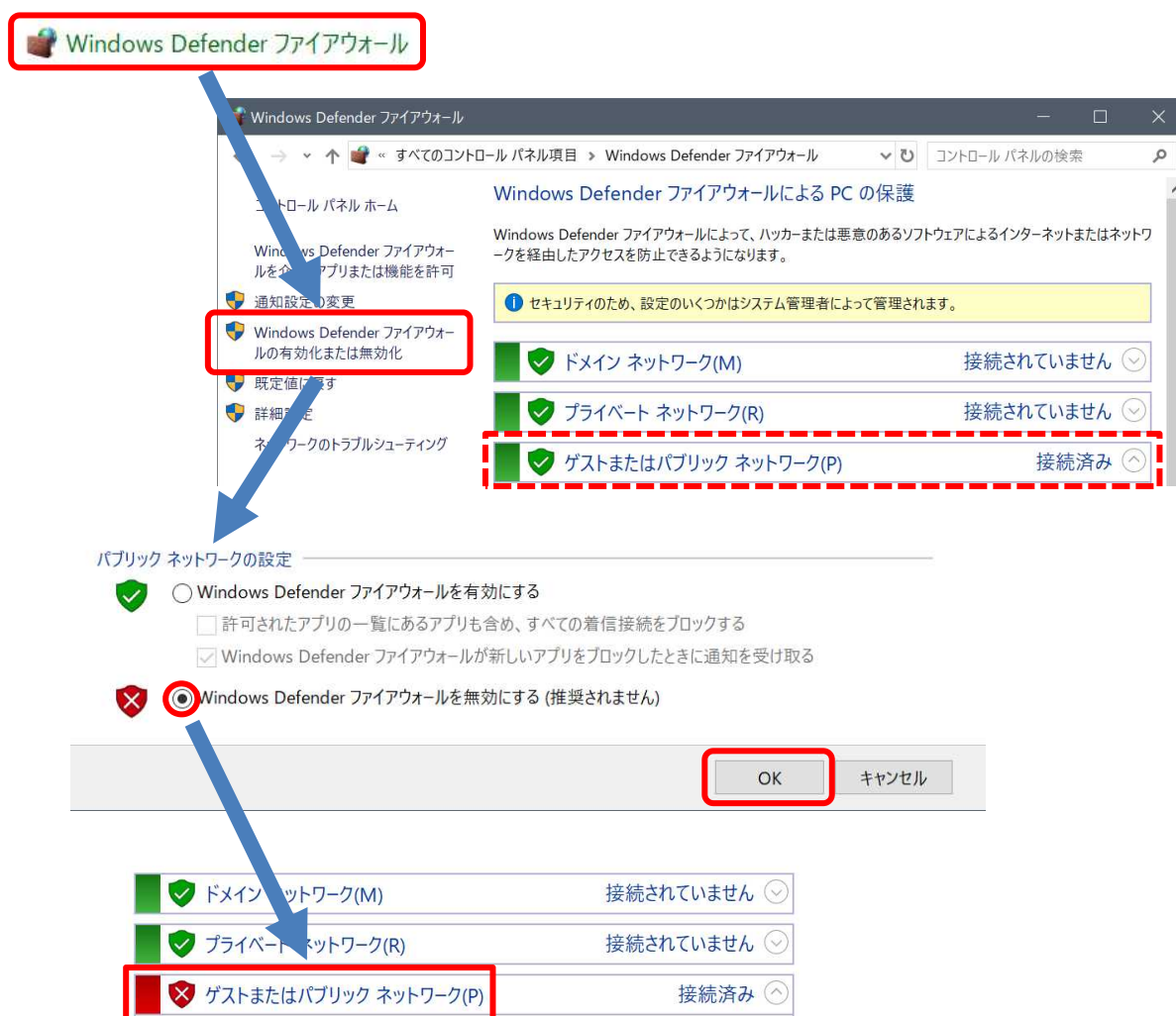


図 6-14. Windows ファイアウォール設定

6-5. Arm® DS の起動と Linux サンプル・アプリケーションのインポートおよびビルド

1. Windows® のスタートメニューまたは、SoC EDS のインストール・フォルダー（intelFPGA¥<バージョン>¥embedded）に格納されている起動用スクリプト **Embedded_Command_Shell.bat** をダブルクリックして、Embedded Command Shell を起動します。

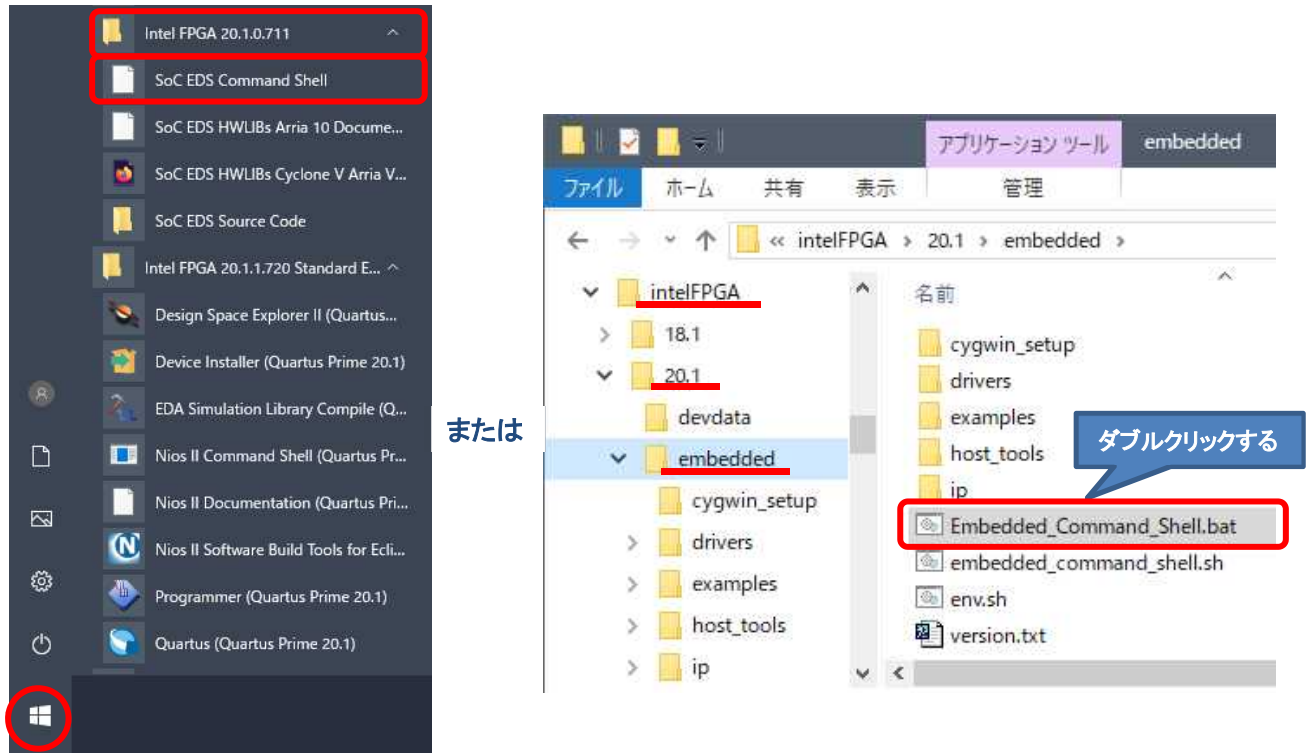


図 6-15. Embedded Command Shell を起動

2. Embedded Command Shell 上で以下のコマンドを実行し、Arm® DS を起動します。

```
$ exec /cygdrive/c/Program Files/Arm/Development Studio/2020.1/bin/cmdsuite.exe
> bash
$ armds_ide &
```

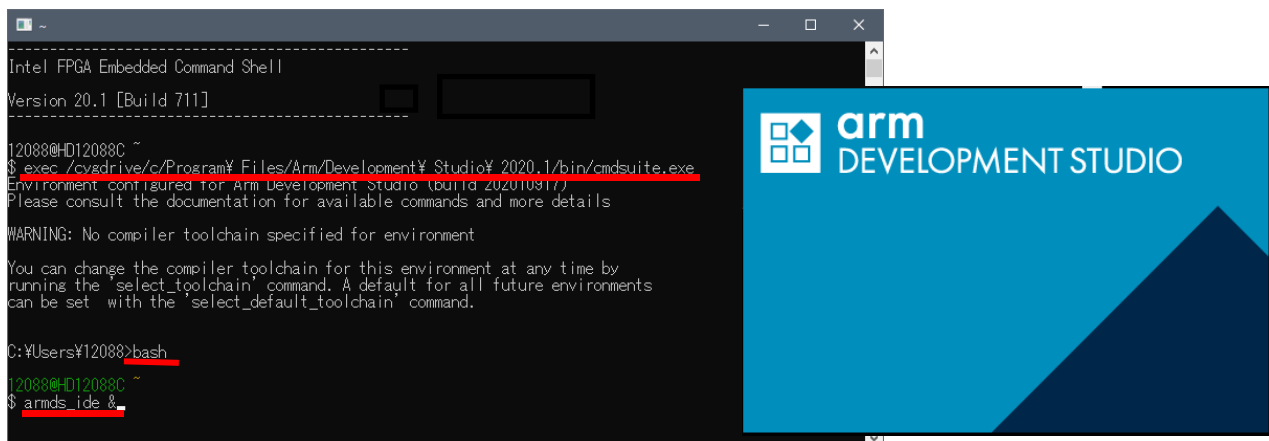


図 6-16. Embedded Command Shell から Arm® DS を起動

3. Arm® DS での作業に使用するワークスペース・フォルダーを設定します。

この演習では、「3. 演習 1: ハードウェア演習」の作業フォルダーに workspace を作成します。

以下のパスを指定して [Launch] をクリックします (フォルダーが存在しない場合は自動的に作成されます)。

C:\lab\soc_lab\cv_soc_lab\workspace

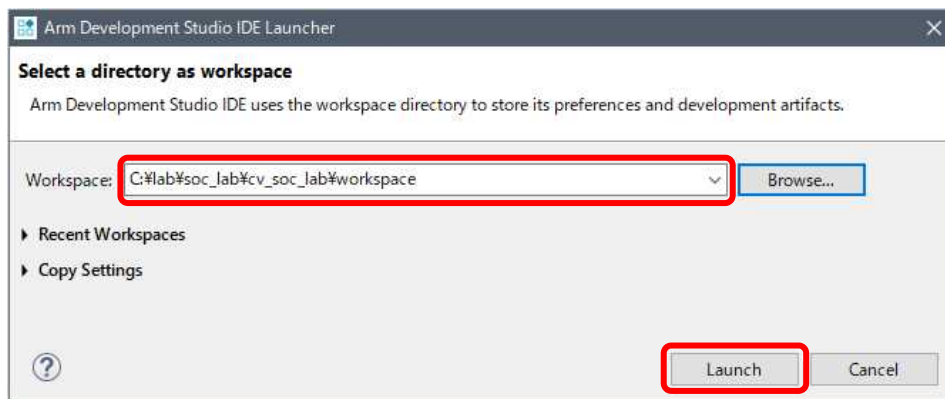


図 6-17. Workspace の作成

4. Preferences Wizard が表示された場合は、内容を確認の上で [Apply & Close]、Arm® DS の Welcome 画面が表示された場合は、[閉じる] (×マーク) をクリックして閉じます。

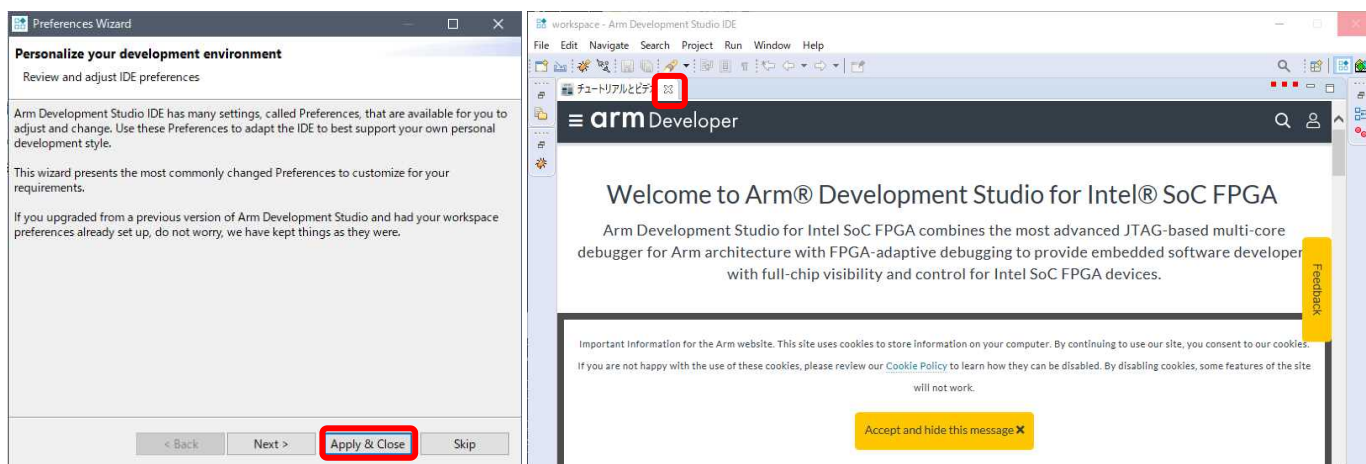


図 6-18. Preference Wizard および Welcome 画面

① Note:

現バージョンの Arm® DS (2020.1) では、Welcome 画面(チュートリアルとビデオ)をオンライン表示する際にウィンドウ操作の反応が重くなる現象が確認されています。オンライン表示が完了するまで無理に操作を行わずにお待ちください。Welcome 画面を閉じた後は、スムーズに操作頂ける状態になります。

Arm® DS をオフラインの状態では起動頂くと、オフライン用の Welcome 画面が使用されるため、この問題を回避することが可能です。

- ___ 5. Arm® DS のメニューから「File」⇒「Import」を選択します。
- ___ 6. 「General (一般)」⇒「Existing Projects into Workspae (既存プロジェクトをワークスペースへ)」を選択し [Next] をクリックします。

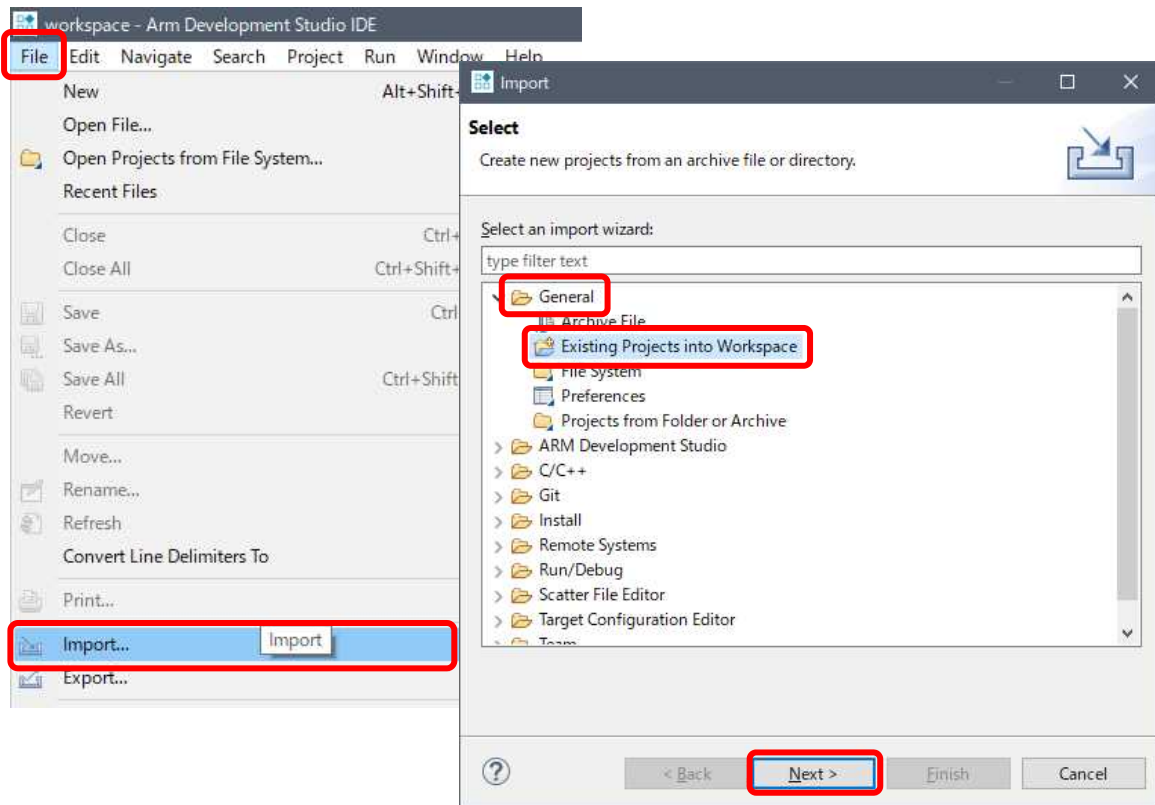


図 6-19. 「ファイル」⇒「インポート」

7. 「**Select archive file:** (アーカイブ・ファイルの選択)」オプションを選択し、[**Browse**] ボタンよりサンプル・プロジェクトを指定します。

サンプル・プロジェクトは SoC EDS に含まれており、デフォルトでは以下のインストール・フォルダーにあります。

C:\intelFPGA\20.1\embedded\examples\software\Altera-SoCFPGA-HelloWorld-Linux-GNU.tar.gz

(<SoC EDS インストール・ディレクトリー>\examples\software\Altera-SoCFPGA-HelloWorld-Linux-GNU.tar.gz をインポートしています)。

選択後、[**Finish**] ボタンをクリックします。

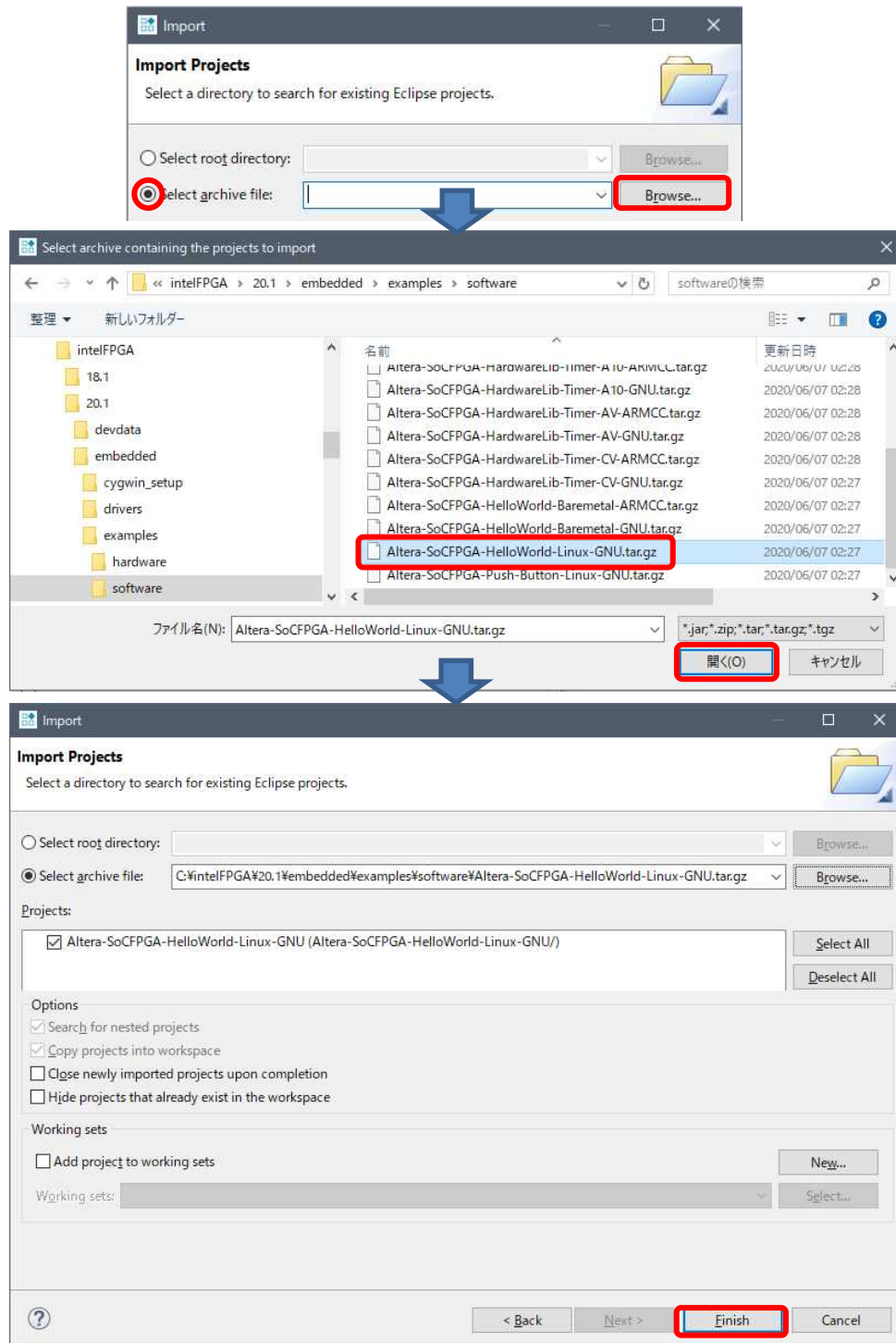


図 6-20. サンプル・プロジェクトのインポート

- ___ 8. Arm® DS ウィンドウ左側のプロジェクト・エクスプローラーに Altera-SoCFPGA-HelloWorld-Linux-GNU プロジェクトが追加され、Altera-SoCFPGA-HelloWorld-Linux-GNU をクリックして展開するとプロジェクトに含まれる各種ファイルが表示されます。プロジェクトのアイコンに赤色の×印  が付いている場合は、ツールチェーンの設定を行います。×印が無い場合は ___ 11. へ進んでください。

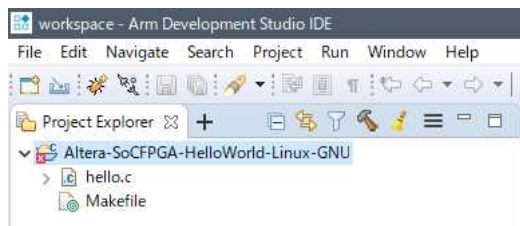


図 6-21. 追加された Altera-SoCFPGA-HelloWorld-Linux-GNU プロジェクト（ツールチェーン未検出）

- ___ 9. プロジェクト・エクスプローラーにて、Altera-SoCFPGA-HelloWorld-Linux-GNU の右クリックメニュー「Properties」を選択し、プロジェクトのプロパティを表示します。「C/C++ Build」⇒「Tool Chain Editor」の「Current toolchain:」が未選択となっているので、arm-none-linux-gnueabihf もしくは arm-linux-gnueabihf を選択し、[Apply and Close] をクリックします。ツールチェーンが選択出来た場合は ___ 11. へ進んでください。

arm-none-linux-gnueabihf もしくは arm-linux-gnueabihf が候補に無い場合は、次の ___ 10. を行った後に再度、この手順を実行してください。

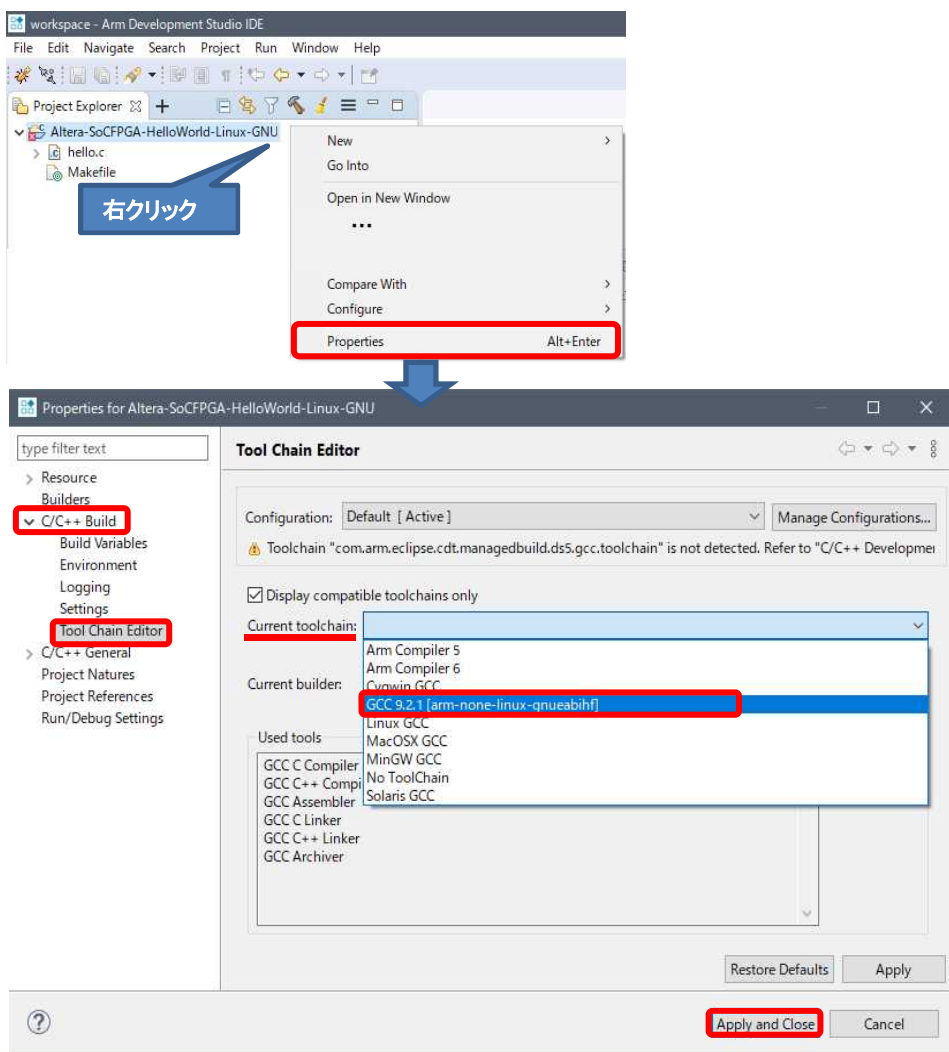


図 6-22. Linux GCC ツールチェーンの選択

10. 「**Window**」メニュー ⇒ 「**Preferences**」を選択し、「**ARM DS**」⇒ 「**Toolchains**」を表示します。Linux 用の GCC (arm-linux-gnueabihf もしくは arm-none-linux-gnueabihf) がリストに無い状態になっているので、[追加] ボタンより、ウィンドウの案内に従ってツールチェーンを追加します。追加が完了したら Arm® DS を再起動します。

任意のディレクトリーに、Linux GCC ツールチェーンをダウンロードおよび展開しておきます。
 <ダウンロードサイト> 例では Arm Developer を使用
[Arm Developer: arm-none-linux-gnueabihf-](https://developer.arm.com/architectures/implementations/arm-none-linux-gnueabihf)
[gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-linux-gnueabihf.tar.xz](https://developer.arm.com/architectures/implementations/arm-none-linux-gnueabihf/gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-linux-gnueabihf.tar.xz)
[Linaro: arm-linux-gnueabihf-](https://developer.arm.com/architectures/implementations/arm-none-linux-gnueabihf/linaro)
[gcc-linaro-7.5.0-2019.12-i686-mingw32_arm-linux-gnueabihf.tar.xz](https://developer.arm.com/architectures/implementations/arm-none-linux-gnueabihf/linaro/gcc-linaro-7.5.0-2019.12-i686-mingw32_arm-linux-gnueabihf.tar.xz)

ダウンロードおよび展開したツールチェーンの直下に
 含まれる bin ディレクトリーを指定
 例:
 C:\intelFPGA\toolchains\gcc-arm-9.2-2019.12-mingw-w64-i686-arm-none-linux-gnueabi\bin

図 6-23. Linux GCC ツールチェーンの追加

11. プロジェクト・エクスプローラーに Altera-SoCFPGA-HelloWorld-Linux-GNU プロジェクトが追加され、Altera-SoCFPGA-HelloWorld-Linux-GNU をクリックして展開するとプロジェクトに含まれる各種ファイルが表示されます。

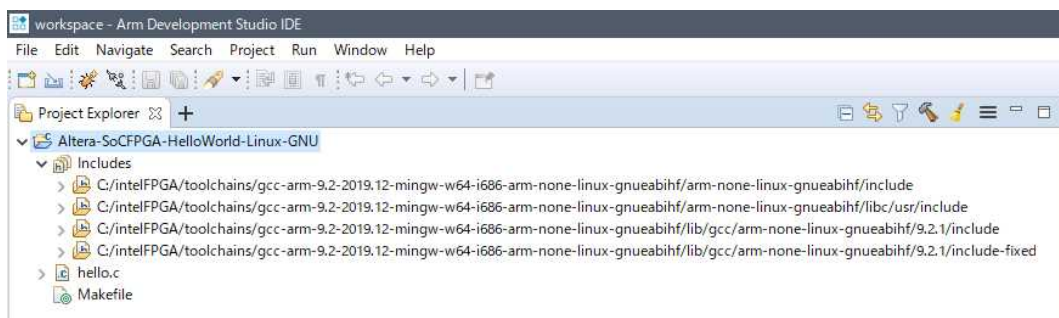


図 6-24. 追加された Altera-SoCFPGA-HelloWorld-Linux-GNU プロジェクト(正常)

12. Altera-SoCFPGA-HelloWorld-Linux-GNU アプリケーションをビルドします。

プロジェクト・エクスプローラーより Altera-SoCFPGA-HelloWorld-Linux-GNU プロジェクトをハイライトし、「Project」⇒「Build Project」を選択します。または、プロジェクト・エクスプローラー上でプロジェクトを選択し、右クリック ⇒ 「Build Project」を実行します。

プロジェクト・エクスプローラーに新たに生成された hello 実行可能ファイルが出力されます。

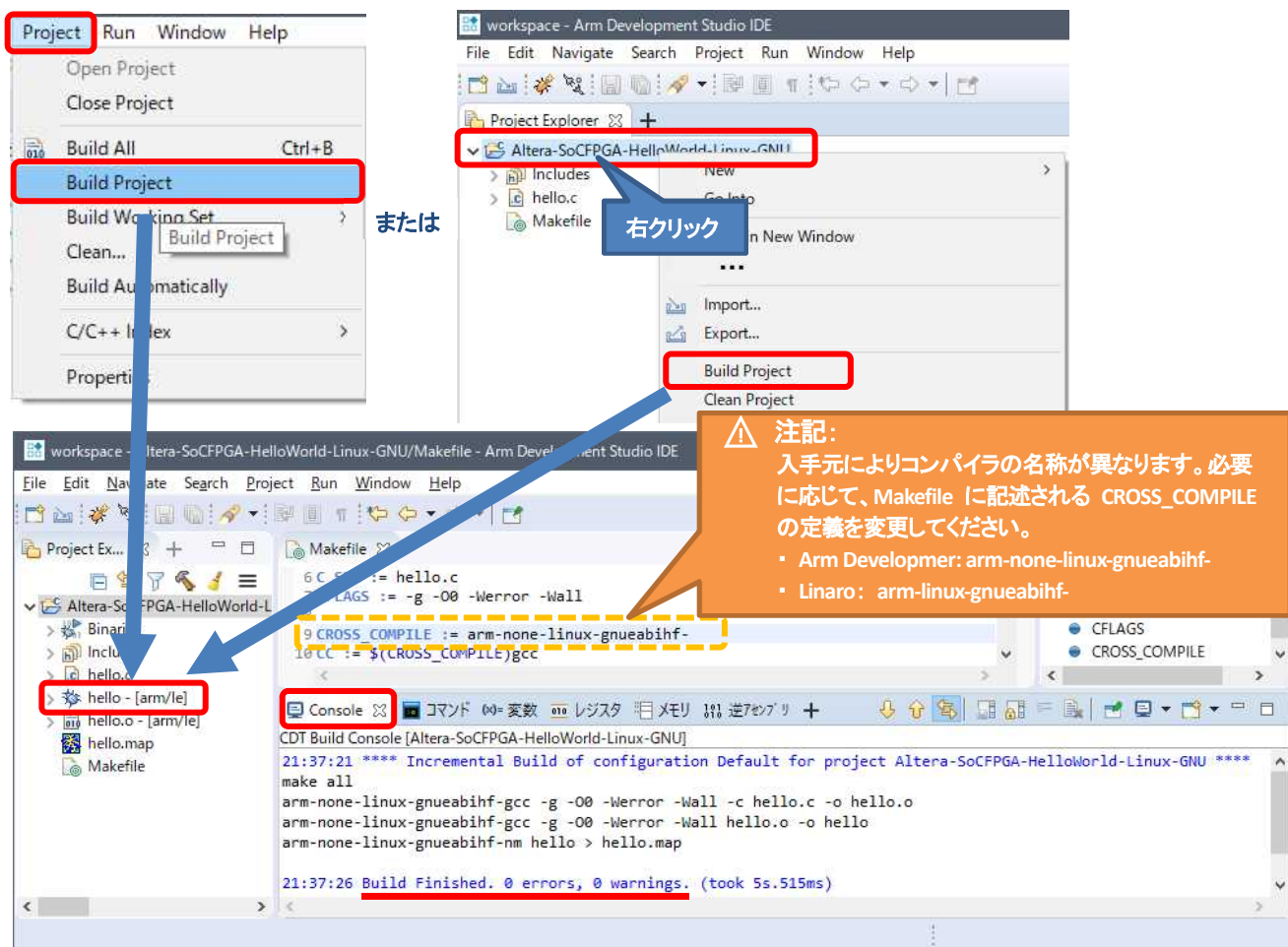


図 6-25. プロジェクトのビルド

6-6. リモート・システム・エクスプローラー (RSE) の設定

Arm® DS では、リモート・システム・エクスプローラー (RSE) を使用する事で、Linux アプリケーション・プログラムをターゲット上で実行・デバッグすることが可能です。

- ___ 1. 「**Window**」メニュー ⇒ 「**Perspective**」⇒ 「**Open Perspective**」⇒ 「**Other**」を選択します。

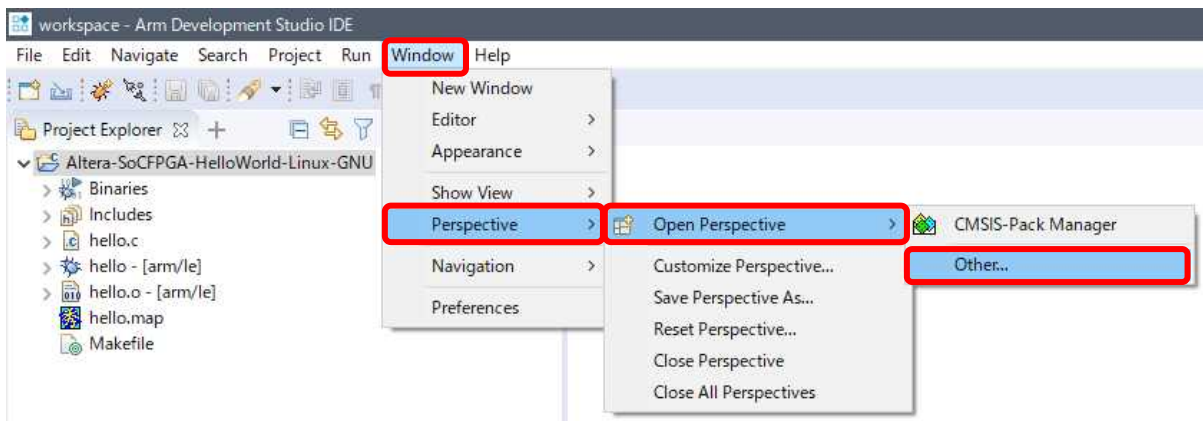


図 6-26. 「パースペクティブを開く」⇒「その他」を選択

- ___ 2. 「**Remote System Explorer (リモート・システム・エクスプローラー)**」を選択して [Open] をクリックします。

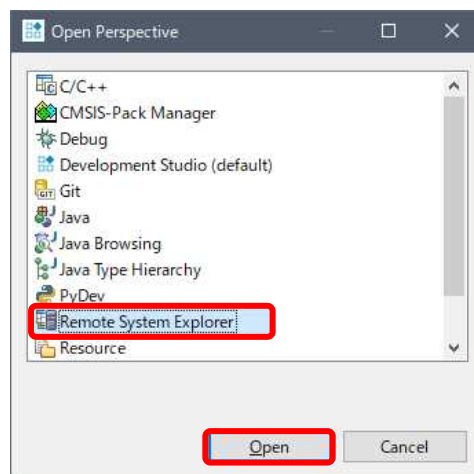


図 6-27. 「リモート・システム・エクスプローラー」の選択

- ___ 3. リモート・システム・エクスプローラーのビューで  ボタンまたは、空白部分を右クリックして「新規接続」を選択します。

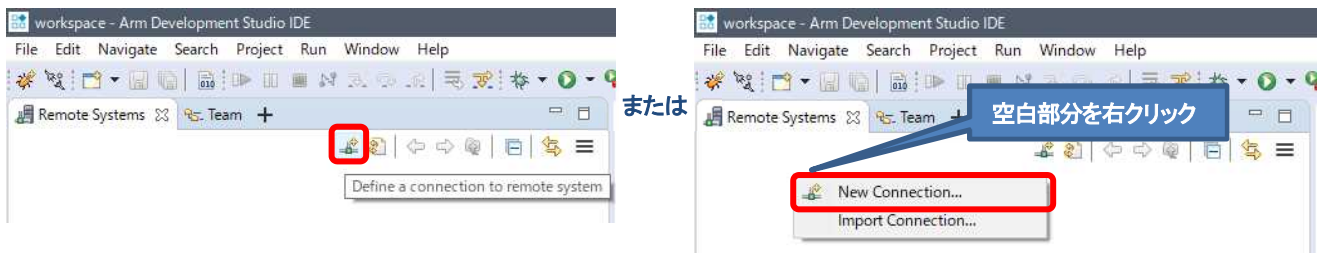


図 6-28. 「リモート・システム・エクスプローラー」での新規接続

- ___ 4. リモート・システム・タイプの選択のビューで「SSH Only」を選択し [Next] をクリックします。

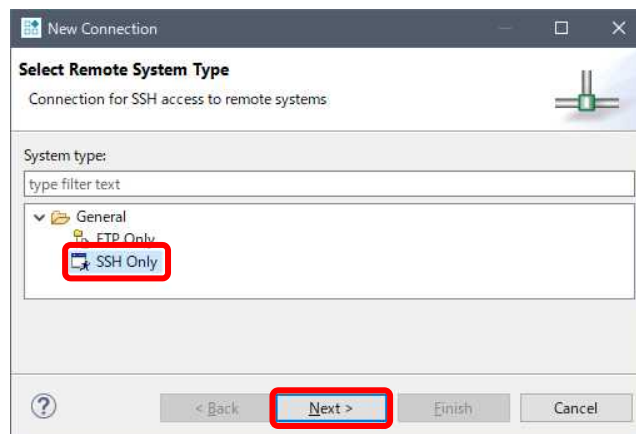


図 6-29. 「SSH のみ」を選択

- ___ 5. 「Host name: (ホスト名)」の欄には設定しておいたボードの IP アドレス（この例では 192.168.1.30）を入力し、「Connection name: (接続名)」と「Description: (記述/説明)」には“Atlas SoC”または“DE10 Nano”と入力し「Verify host name (ホスト名を検証)」にチェックを入れて [Finish] ボタンをクリックします。

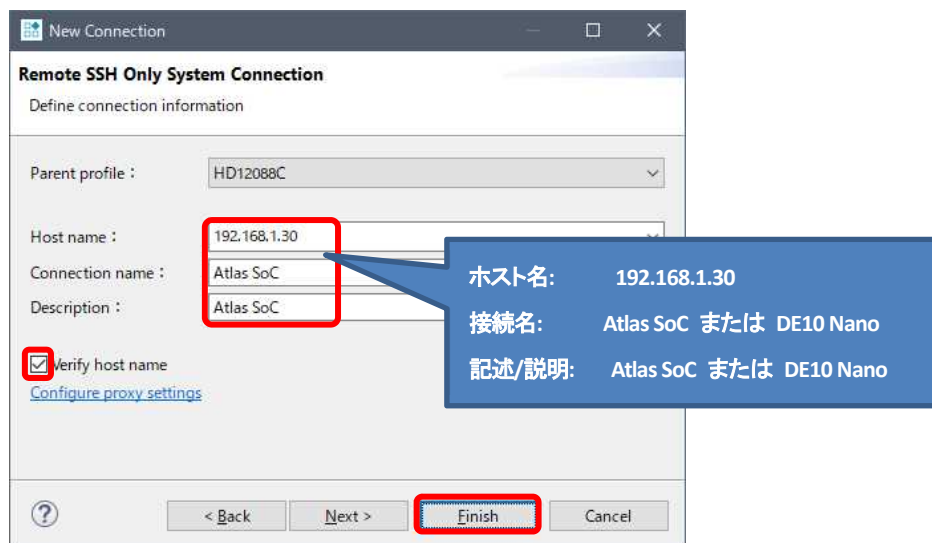


図 6-30. 接続設定

___ 6. リモート・システム・エクスプローラーのビューで、「Atlas SoC」(または「DE10 Nano」) ⇒ 「Sftp Files」 ⇒ 「Root」をクリックすると、ユーザー ID とパスワードを入力するウィンドウが表示されます。

___ 7. 「User ID:」には「root」、「Password」には設定したパスワードを入力して [OK] をクリックします。

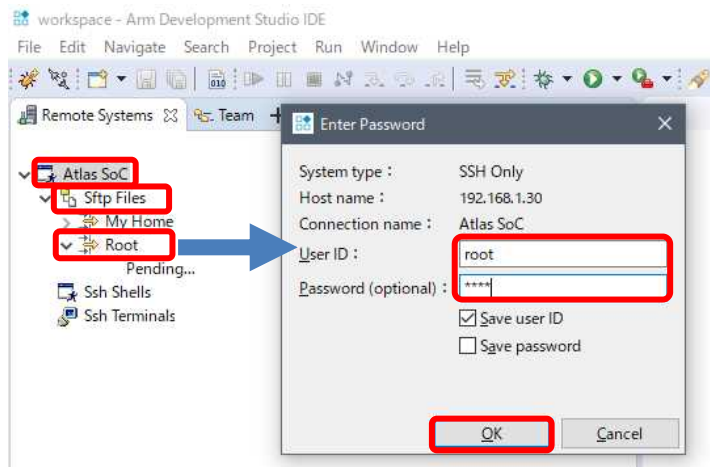


図 6-31. ユーザー ID とパスワードを入力

___ 8. 下図の警告が出た場合は [Yes] をクリックします。

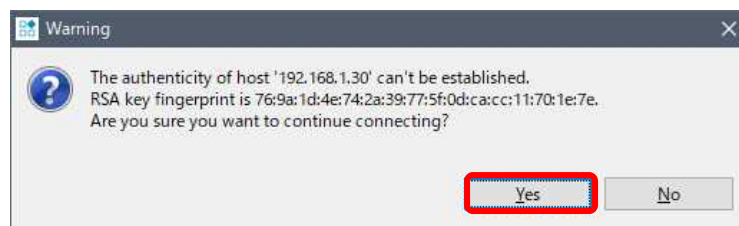


図 6-32. 警告表示

___ 9. 接続が成功すると、リモート・システム・エクスプローラーに現在のボード上のファイル群が表示されます。

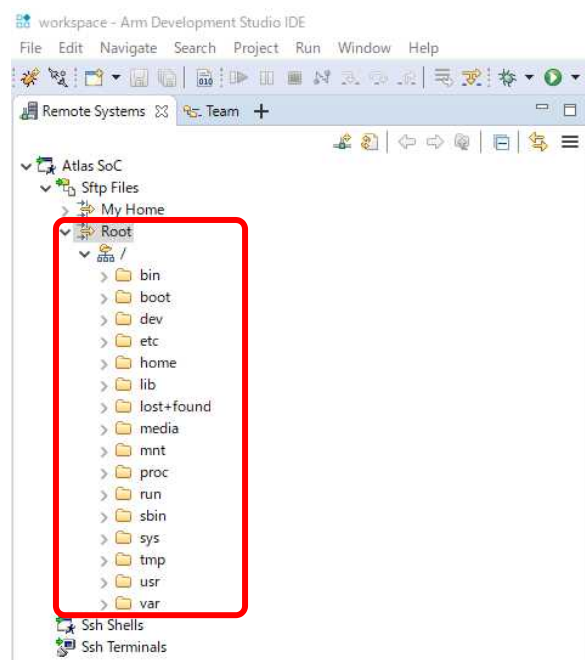


図 6-33. リモート・システム・エクスプローラーに現在のボード上のファイル群が表示

10. エラーが出て接続できない場合は、ホスト PC のプロキシ設定の問題が考えられます。この場合は「コントロール パネル」⇒「インターネット オプション」をクリックし、「接続」タブの「LAN の設定」をクリックします。
11. “LAN にプロキシサーバーを使用する” にチェックが入っている場合は、このチェックを外して [OK] をクリックします。

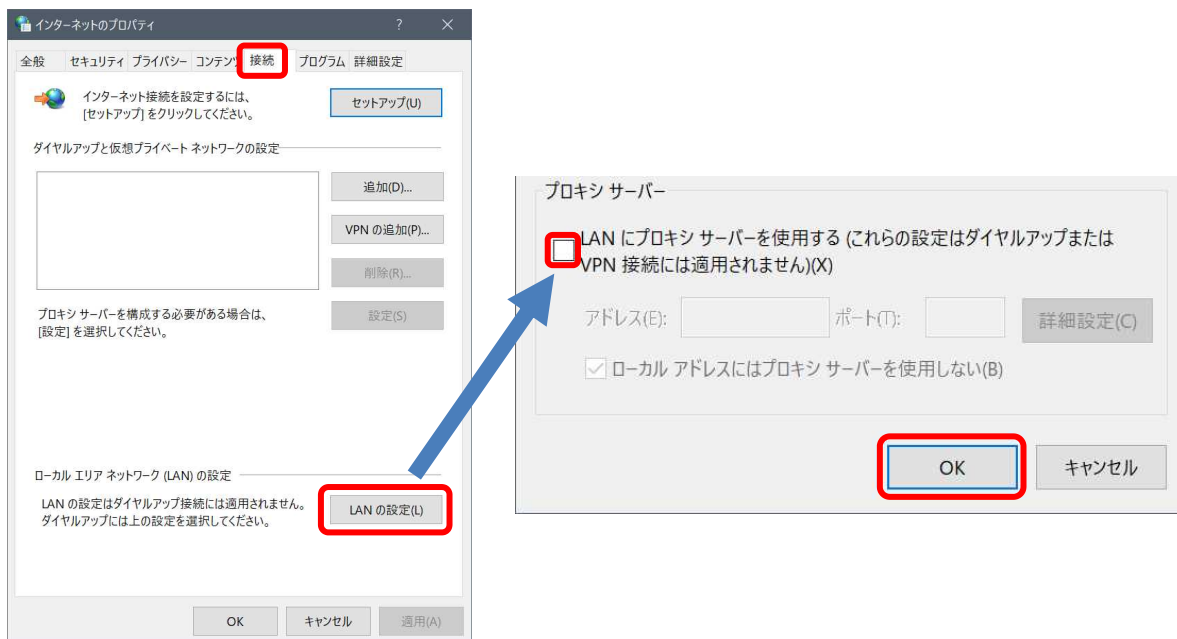


図 6-34. プロキシサーバーの設定

12. 再度 Atlas SoC（または DE10 Nano）のルートへの接続を試みてください。

6-7. Linux アプリケーションの実行・デバッグ

ここからは、デバッガ設定方法と実行・デバッグ方法について確認します。

- ___ 1. メニューバー(ウィンドウ右上)の  ボタンをクリックして Development Studio のメイン・パースペクティブに戻ります。
- ___ 2. プロジェクト・エクスプローラー・タブより **Altera-SoC FPGA-HelloWorld-Linux-GNU** を右クリックして、「**Debug As (デバッグ)**」⇒「**Debug Configurations (デバッグの構成)**」を選択します。

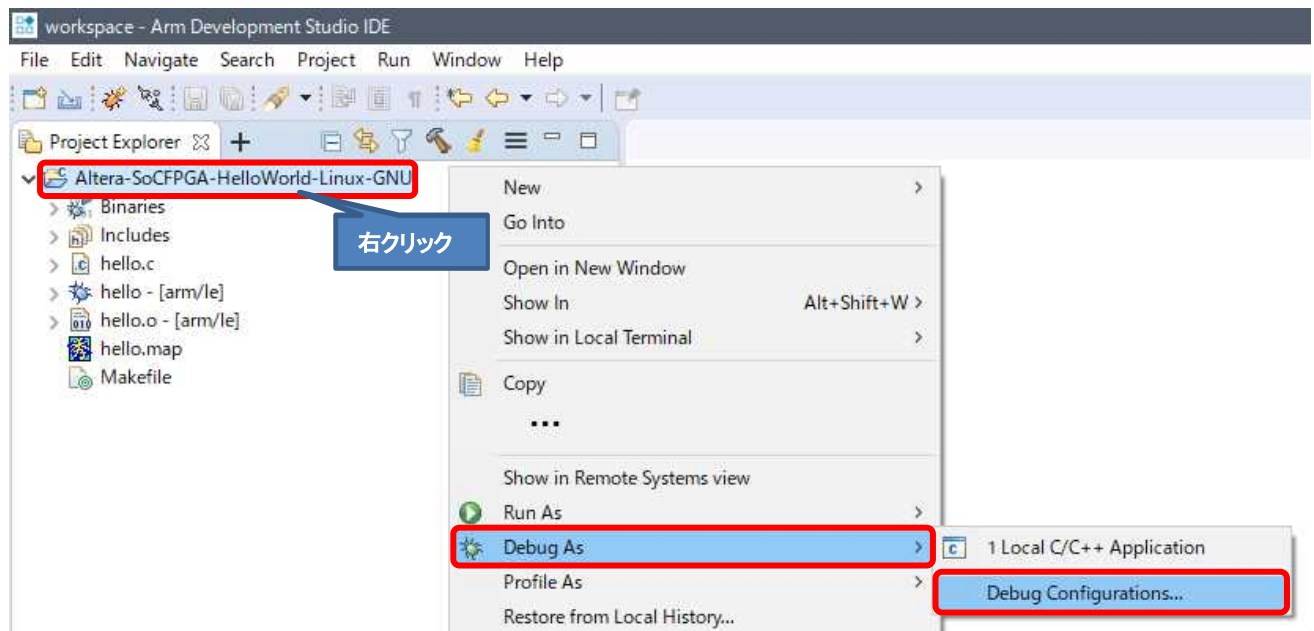


図 6-35. 「デバッグ」⇒「デバッグの構成」を選択

- ___ 3. 「汎用 ARM C/C++ アプリケーション」を右クリックし「**New Configuration**」を選択して、新しいデバッグ・コンフィグレーションを作成します。

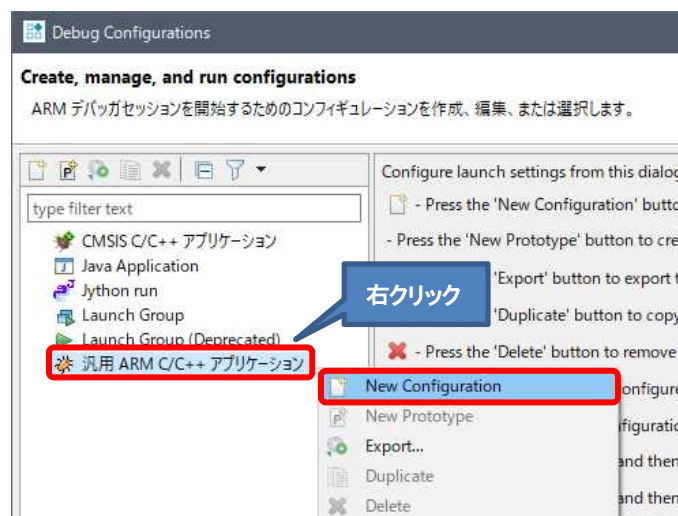


図 6-36. 新しいデバッグ・コンフィグレーションを作成

4. 「名前」フィールドに“HelloWorld”と入力します。
5. 「接続」タブの「ターゲットの選択」フィールドにおいて、
Intel SoC FPGA ⇒ Cyclone V SoC (Dual Core) ⇒ Linux Application Debug ⇒ Download and debug application
を選択します。
6. 「接続」フィールドでは、生成した RSE 接続（この例では Atlas SoC）を選択し、その他はデフォルト値を使用します。

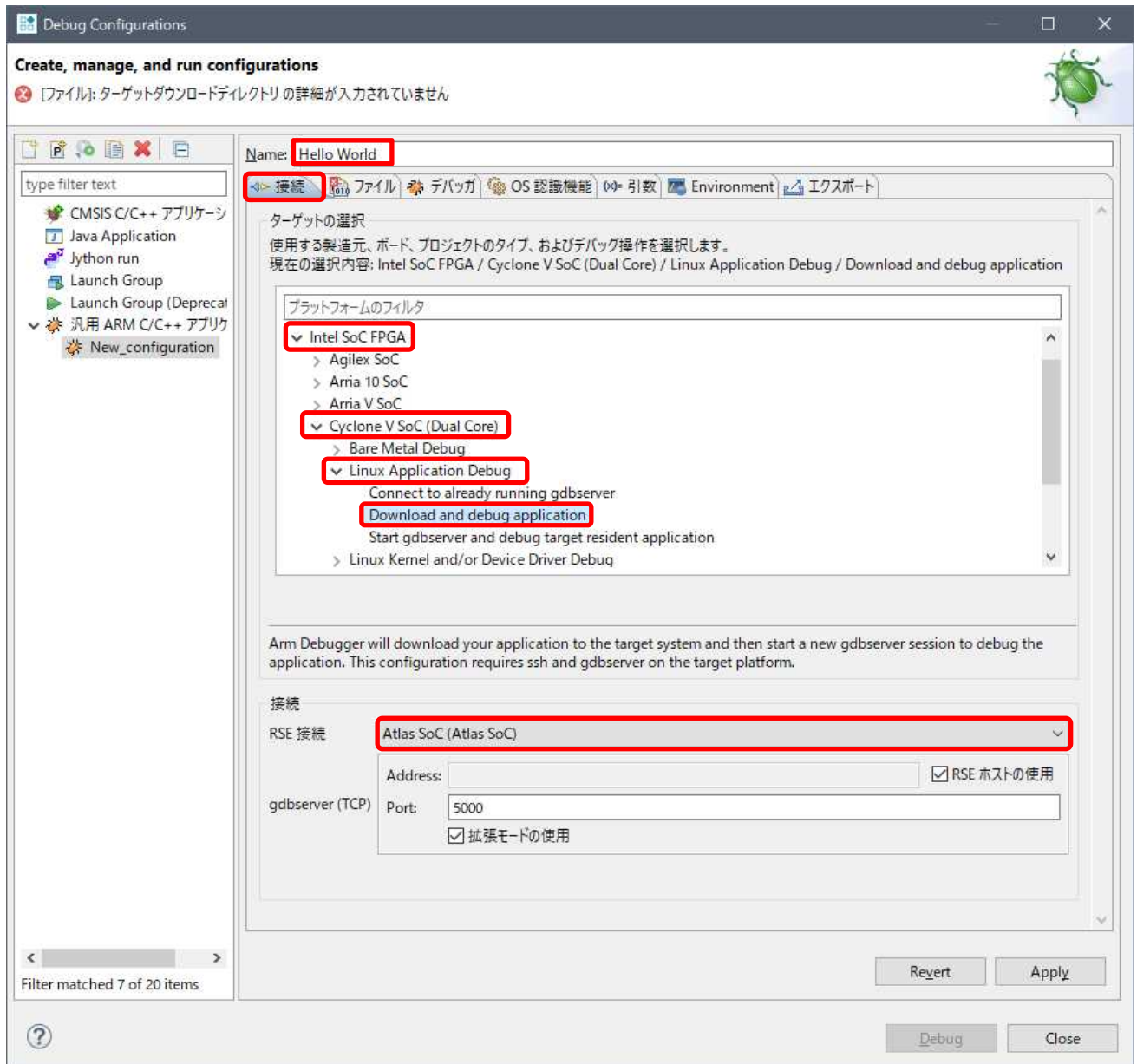


図 6-37. デバッグ構成の設定 (1)

7. 「ファイル」タブの「ダウンロードするホスト上のアプリケーション:」に Hello World の実行体を設定します。
「ワークスペース」ボタンを使用して hello を選択し [OK] をクリックします。

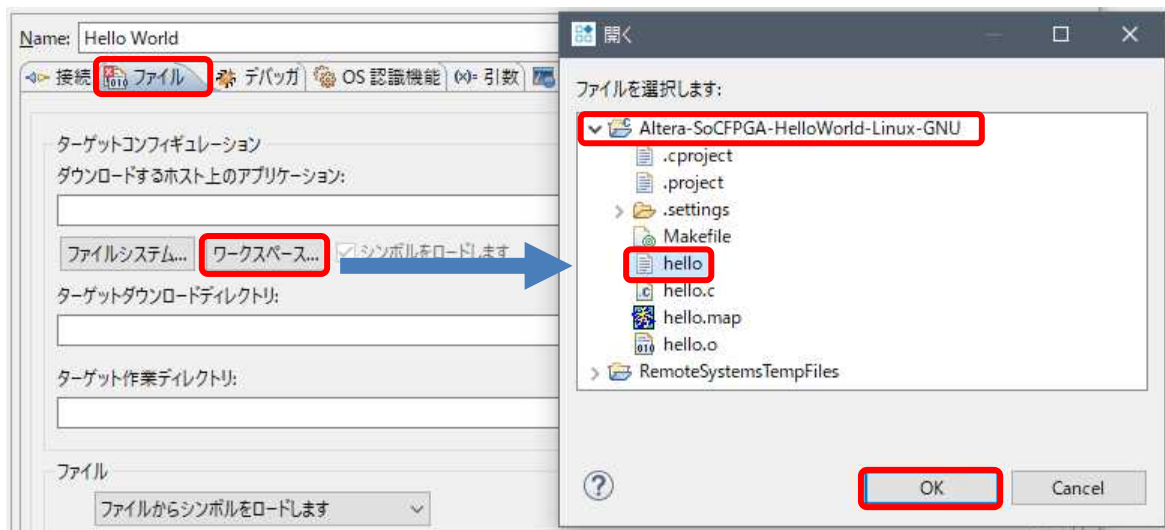


図 6-38. デバッグ構成の設定 (2)

8. 「ターゲットダウンロードディレクトリ:」と「ターゲット作業ディレクトリ:」には “/home/root” を設定します。

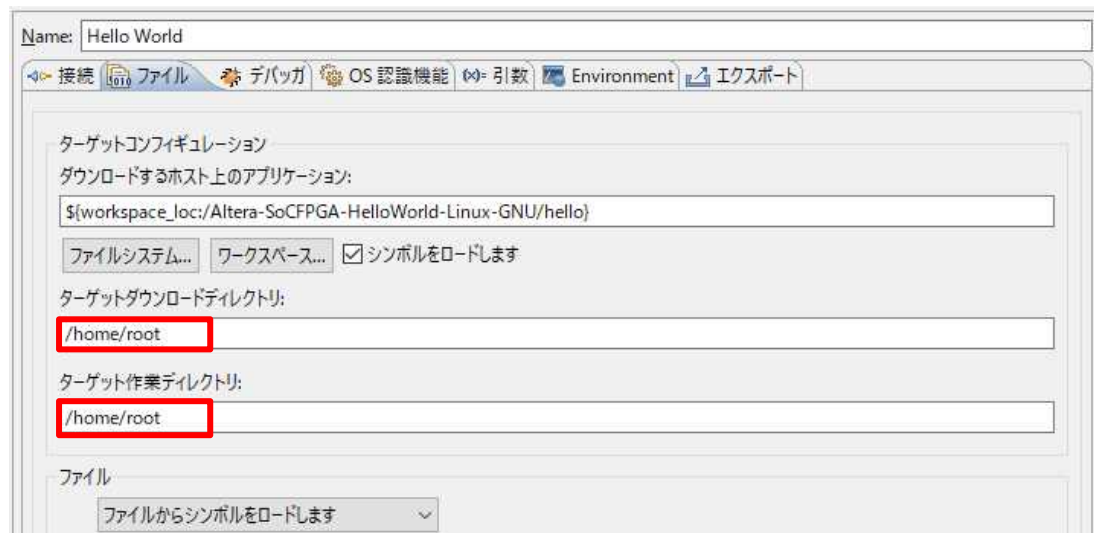


図 6-39. デバッグ構成の設定 (3)

9. 「デバッガ」タブで、実行制御フィールドは「シンボルからデバッグします」を選択し、シンボル名に “main” と入力します。

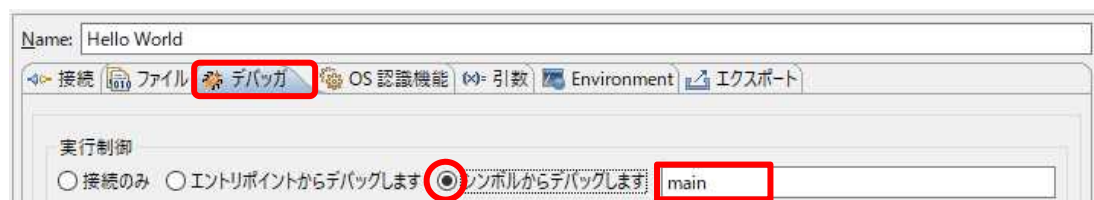
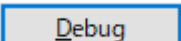


図 6-40. デバッグ構成の設定 (4)

10. 「デバッグ」ボタン  をクリックしてデバッグセッションを開始します。

11. デバッグセッションが開始すると App Console に以下のようなログが表示されます。



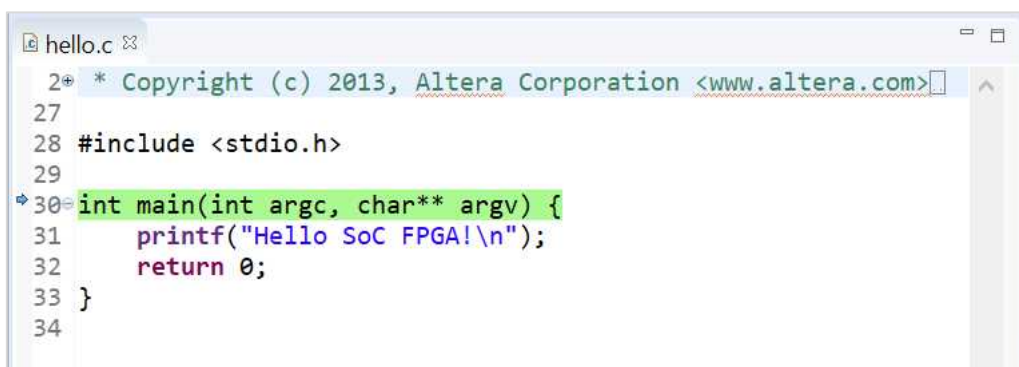
```

デバッグセッションの準備中
# cd "/home/root"
# export LD_LIBRARY_PATH=".:/home/root:$LD_LIBRARY_PATH"
# gdbserver :5000 "/home/root/hello"
Process /home/root/hello created; pid = 1384
Listening on port 5000
デバッグセッションが開始され、gdbserver に接続中です
Remote debugging from host 192.168.1.31

```

図 6-41. デバッグセッション開始時の App Console

12. アプリケーションは、ロードされてから main 関数でブレークします。



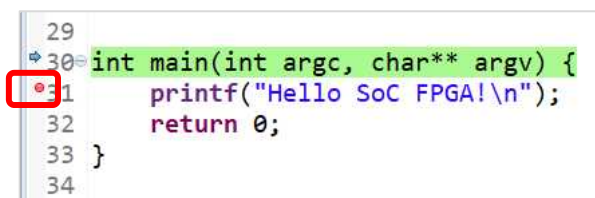
```

hello.c
27 #include <stdio.h>
29 int main(int argc, char** argv) {
31     printf("Hello SoC FPGA!\n");
32     return 0;
33 }
34

```

図 6-42. Main 関数でブレーク

13. ソースコードの左余白をダブルクリックすると、赤い点 ● で示すようにデバッガがそこにブレークポイントを設定します。



```

29 int main(int argc, char** argv) {
31     printf("Hello SoC FPGA!\n");
32     return 0;
33 }
34

```

図 6-43. ブレークポイントの設定

14. [続行] ボタン  (または F8) を押すと、アプリケーションが実行されてブレークポイントで停止します。



```

29 int main(int argc, char** argv) {
31     printf("Hello SoC FPGA!\n");
32     return 0;
33 }
34

```

図 6-44. ブレークポイントで停止

15. ソースコードの左余白に赤い点 ● で示されたブレークポイントをダブルクリックすると、ブレークポイント設定が解除されます。

16. [ソース行のステップ実行] ボタン  (または F5) を押すと、実行コードが 1 ライン進みます。

17. 「Window」メニュー ⇒ 「Show View」⇒ 「Other」より、デバッグ時に便利な各種ビューの追加が行えます。
「ARM デバッグ」以下にリストされる項目から表示したいビューを選択し、[Open] をクリックします。

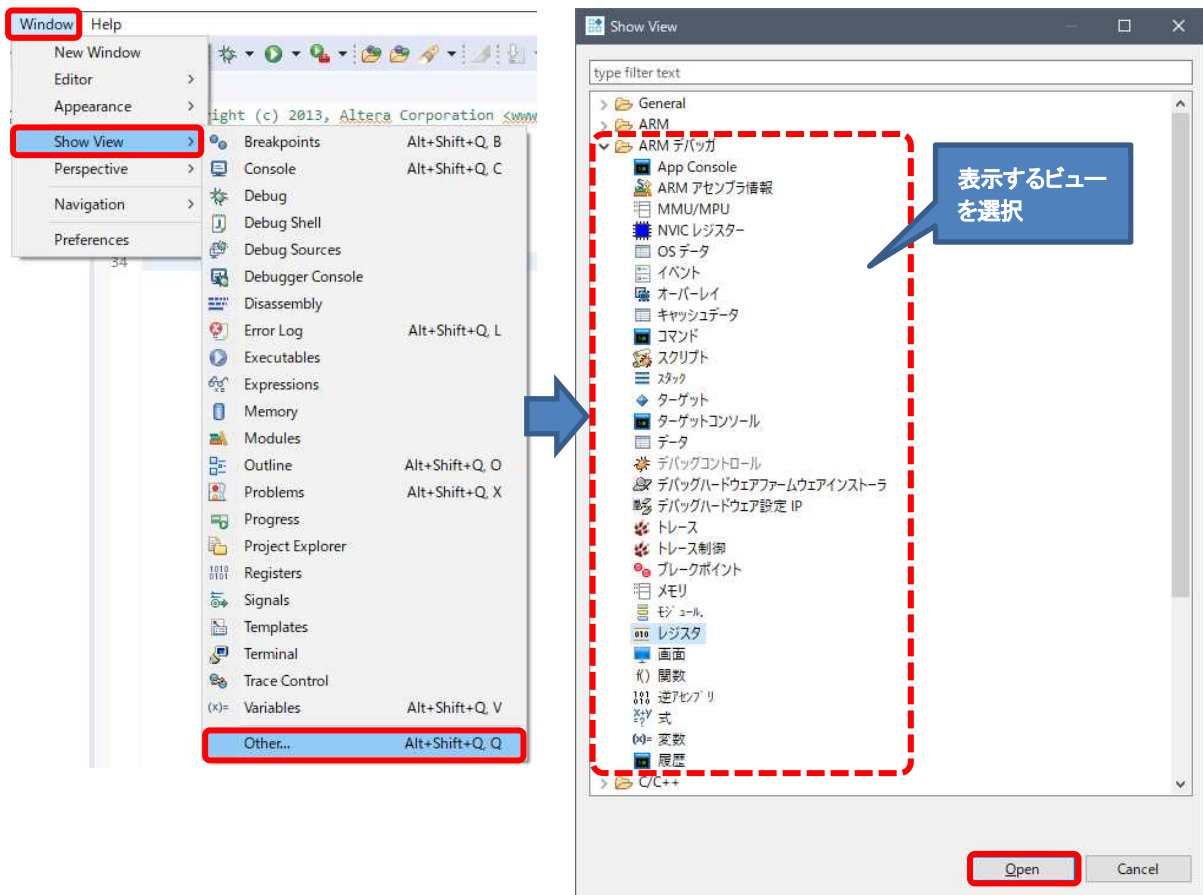


図 6-45. ビューの追加

18. 「レジスタ」ビューは、ターゲット・レジスタの内容を表示します。また、書き込み可能なレジスタの値を変更できます。

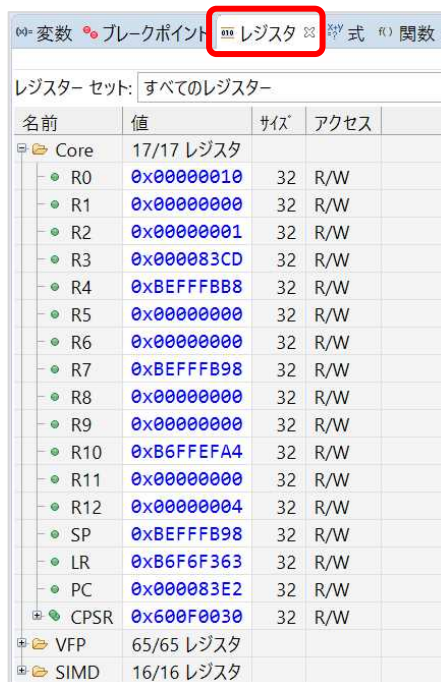


図 6-46. 「レジスタ」ビュー

19. 「変数」ビューは、現在有効範囲にある変数の内容を表示します。また、現在有効範囲にある変数の値を変更できます。



図 6-47. 「変数」ビュー

20. 「App Console」(アプリケーション・コンソール) ビューは、Arm C ライブラリーでのセミホスティングの実装によって提供されるコンソール I/O 機能を使用できます。アプリケーション内の print 文の内容が表示されます。

21. 「続行」ボタン  を押すと、アプリケーションが続行され Hello SoC FPGA! と表示されます。

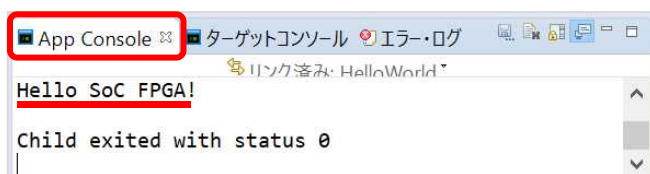


図 6-48. 「App Console」(アプリケーション・コンソール) ビュー

22. 「main() からデバッグ」 をクリックすると、アプリケーションの先頭 main に戻ってブレークします。

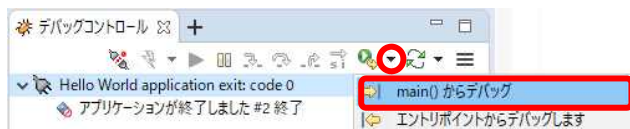


図 6-49. デバッグコントロール「main() からデバッグ」

23. 再度「続行」ボタン  を押すと、アプリケーションが先頭から実行され、「App Console」ビューに Hello SoC FPGA! と表示されます。

24. 「ターゲットから切断」ボタン  をクリックしてデバッグセッションを終了します。

25. 「すべての接続の削除」ボタン  をクリックしてデバッグセッションを削除します。

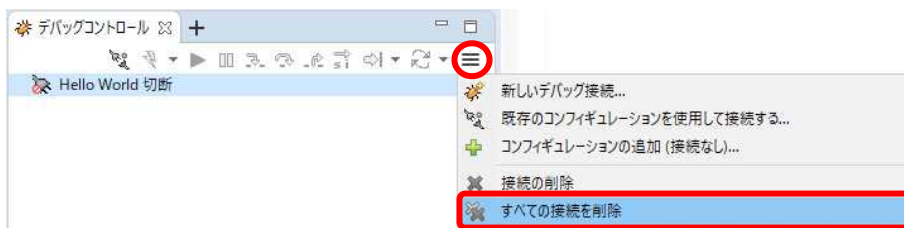


図 6-50. デバッグコントロール「すべての接続を削除」

26. ウィンドウ右上の  ボタンを右クリックして、「Reset」を選択すると画面配置が初期に戻ります。

おめでとうございます。これで全ての演習が完了しました。

7. 今後の参考資料について

本演習ではインテル® SoC FPGA の開発環境であるインテル® Quartus® Prime 開発ソフトウェアやシステム構成ツールである Platform Designer システム統合ツール、およびソフトウェア開発環境である SoC EDS の基本的な操作を学ぶことを中心に紹介しました。今後さらなる知識向上につなげたい場合はさまざまな情報源がありますのでこちらをご利用ください。

また同様の内容として「SoC はじめてガイド」をご参照いただくと、さらに理解が深まると思いますのでご利用ください。

参考:

- SoC はじめてガイド
 - [インテル の SoC FPGA](#)
 - [SoC はじめてガイド – HPS-FPGA 間のアクセス方法](#)
 - [SoC はじめてガイド – Preloader Generator の使用方法](#)
 - [SoC はじめてガイド – DS-5 によるベアメタル・アプリケーション・デバッグ](#)
- SoC 関連情報
 - [マクニカ・ホームページ技術情報: SoC 関連記事／ドキュメント](#)
 - [マクニカ・ホームページ技術情報: SoC 関連 FAQ](#)
 - [アルティマ技術サポート: SoC 関連記事／ドキュメント](#)
 - [アルティマ技術サポート: SoC 関連 FAQ](#)
- デバイスやツールについての説明
 - [Intel FPGA and SoC](#)
- SoC デバイスで Linux を使用する上で参考となる各種ドキュメントやプロジェクト
 - [RocketBoards.org](#)
- 代理店からの各種情報サイト
 - [マクニカ・ホームページ技術情報](#)
 - [アルティマ技術サポート](#)

改版履歴

Revision	年月	概要
v18.1 r1	2020 年 1 月	初版
v18.1 r2	2020 年 6 月	(5 ページ) 「インテル® SoC Embedded Development Suite」を「インテル® SoC FPGA エンベデッド開発スイート」に変更 (6 ページ) ・「Quartus® Prime Standard Edition v18.1」を「インテル® Quartus® Prime Standard Edition v18.1」に変更 ・「SoC Embedded Development Suite v18.1」を「インテル® SoC FPGA エンベデッド開発スイート Standard Edition v18.1」に変更
v20.1 r3	2020 年 12 月	SoC EDS v20.1 向けの内容にアップデート

免責およびご利用上の注意

弊社より資料を入手されましたお客様におかれましては、下記の使用上の注意を一読いただいた上でご使用ください。

1. 本資料は非売品です。許可無く転売することや無断複製することを禁じます。
2. 本資料は予告なく変更することがあります。
3. 本資料の作成には万全を期していますが、万一ご不明な点や誤り、記載漏れなどお気づきの点がありましたら、本資料を入手されました下記代理店までご一報いただければ幸いです。
[株式会社マクニカ 半導体事業 お問い合わせフォーム](#)
4. 本資料で取り扱っている回路、技術、プログラムに関して運用した結果の影響については、責任を負いかねますのであらかじめご了承ください。
5. 本資料は製品を利用する際の補助的な資料です。製品をご使用になる際は、各メーカー発行の英語版の資料もあわせてご利用ください。