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

Ver.20.1

2025 年 2 月 Rev.4

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

目次

本書をお読みになる前に

この資料の内容は 2025 年 2 月現在のものです。

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

ⓘ **Note**	補足情報などを記載しています。
Ⓟ **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

下線	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、アルテラ® SoC FPGA の開発環境である 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 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® プロセッサー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。

> ⓘ **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。

> ⓘ **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。

- 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。
  Quartus® Prime 開発ソフトウェアおよび Questa* - Intel® FPGA Edition のダウンロード方法
  Quartus® Prime 開発ソフトウェアおよび Questa* - Intel® FPGA Edition のインストール方法

- 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。
  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 ver.20.1

- 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して説明しています。

- 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。

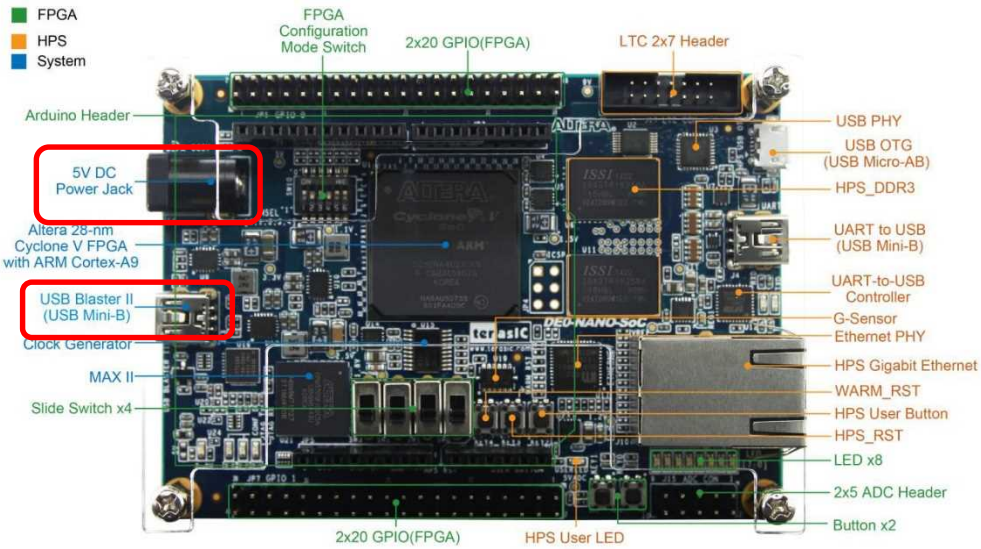


図 2-1. Atlas-SoC ボードレイアウト図

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 コネクター（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

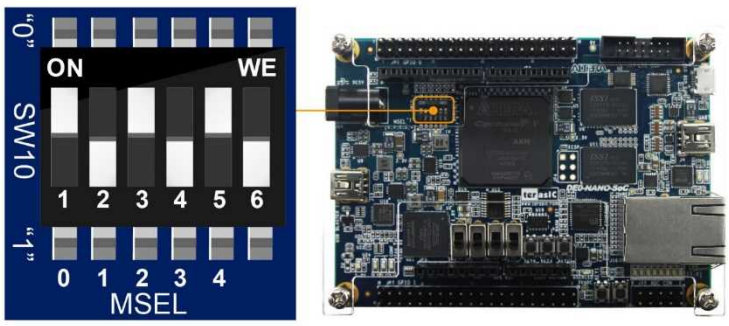


図 2-2. ジャンパーの設定

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® プロセッサーを含むハードウェアの設計を行います。

アルテラ® 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

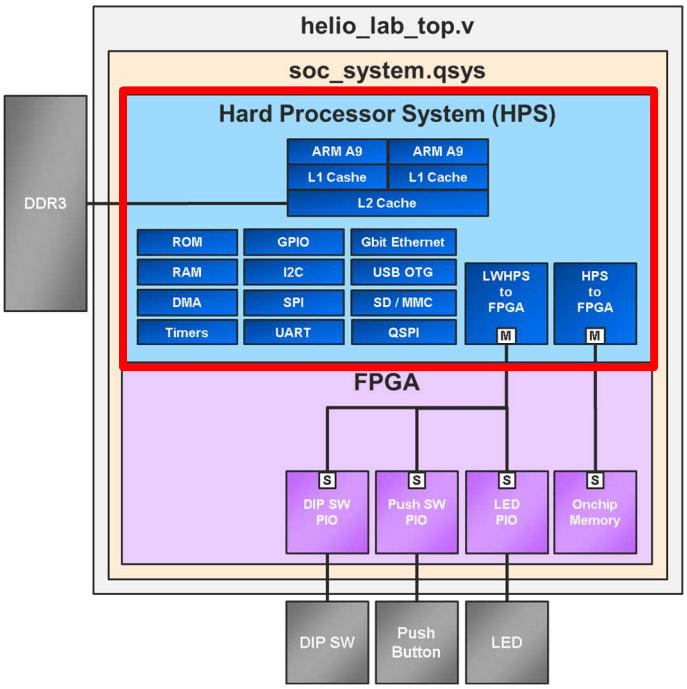


図 3-1. 演習 1 で設計する SoC システムのブロック図

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 フォルダーとして説明をします。作業フォルダー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)

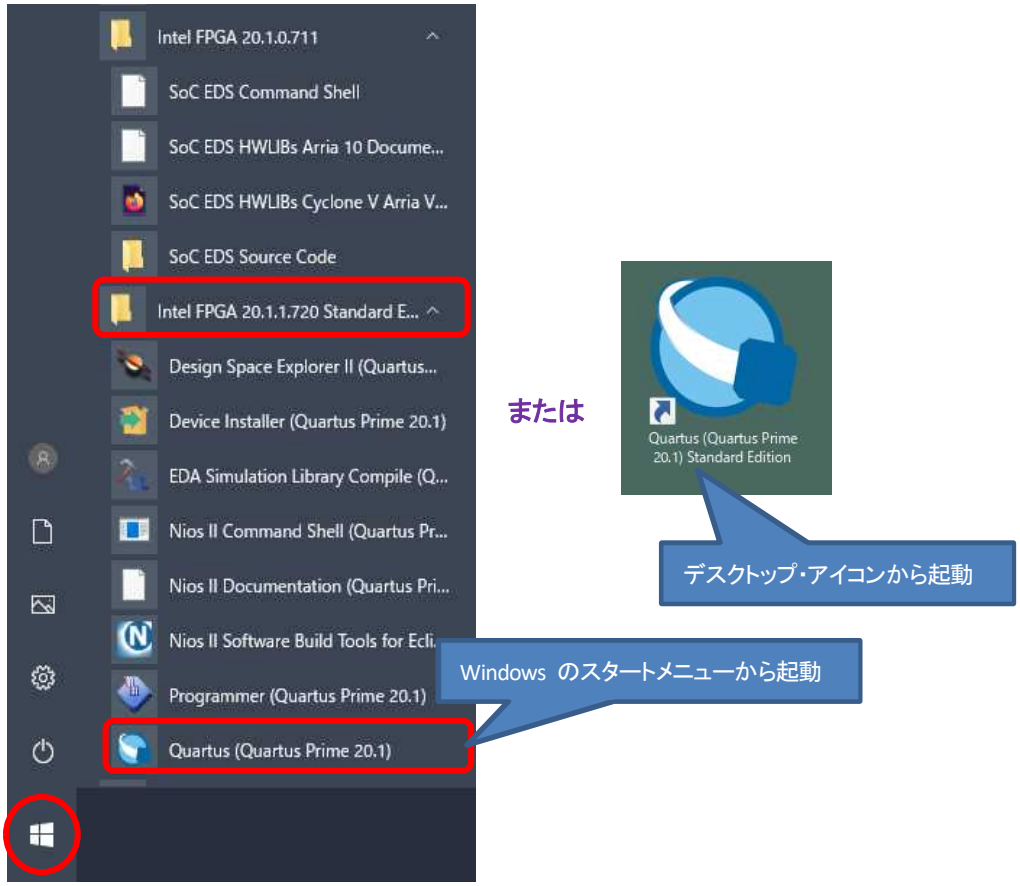


図 3-2. Quartus® Prime の起動

____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。

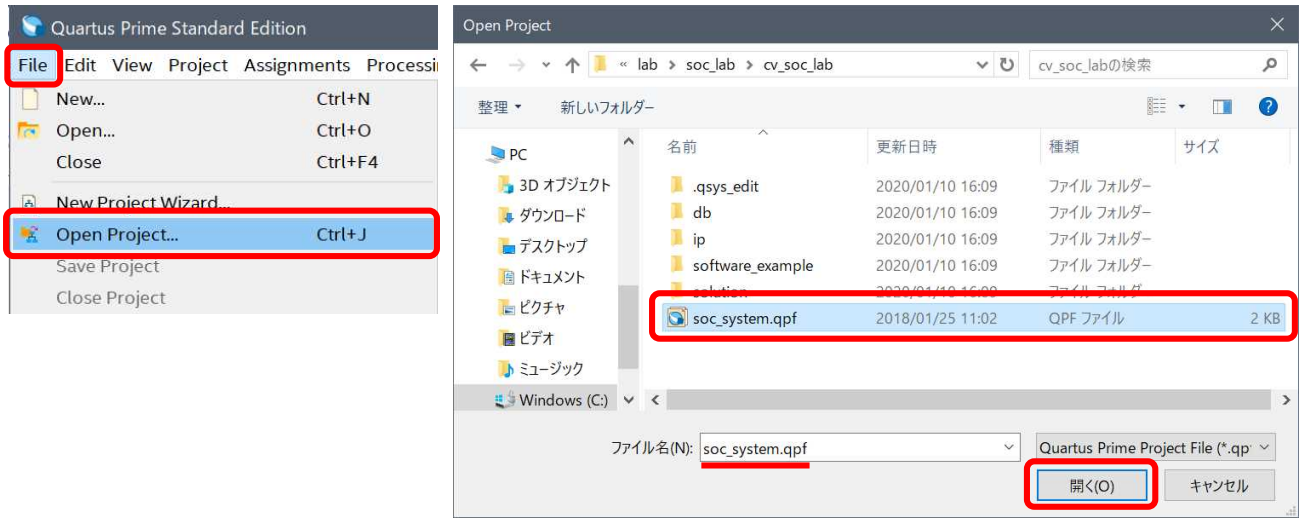


図 3-3. Quartus® Prime プロジェクトのオープン

____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。

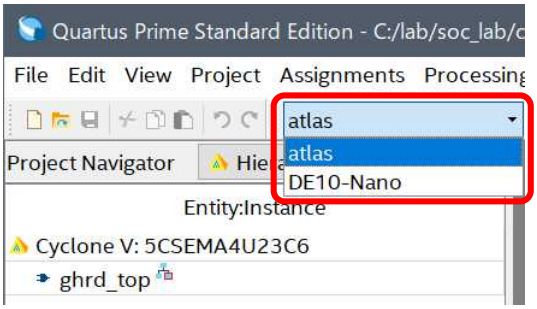


図 3-4. 使用ボードの選択

____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。

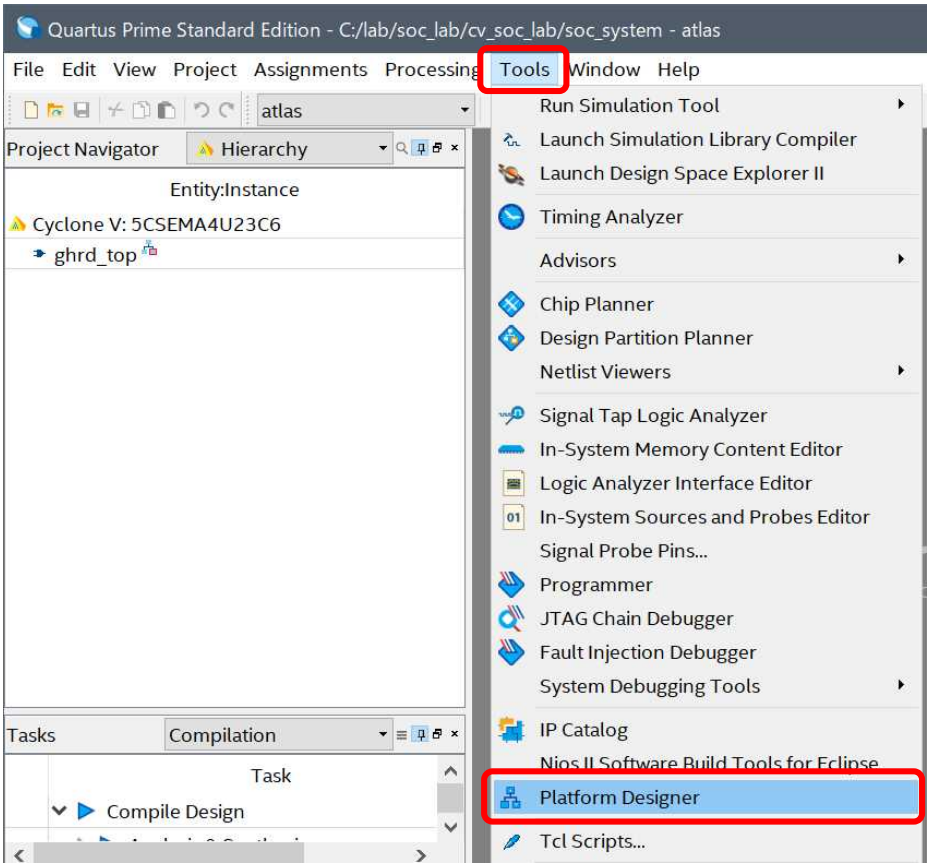


図 3-5. Platform Designer の起動

____5. soc_system.qsys ファイルを開きます。

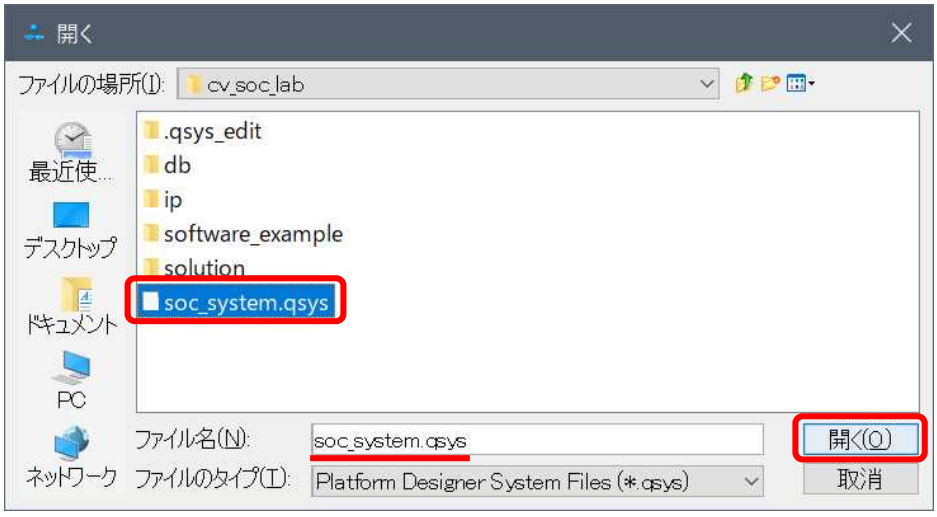


図 3-6. Platform Designer ファイルのオープン

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、 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。

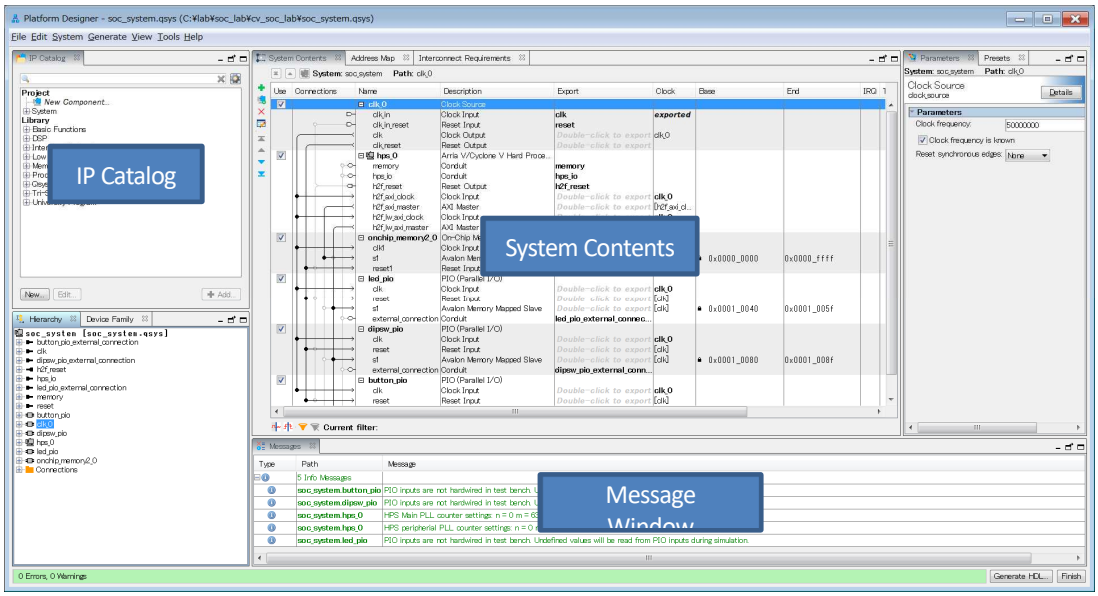


図 3-7. Platform Designer 画面

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

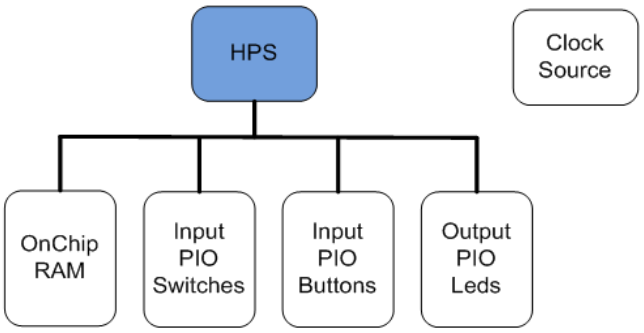


図 3-8. 設計する Platform Designer システム

■ 演習で追加するコンポーネント（青色）:

- HPS

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。

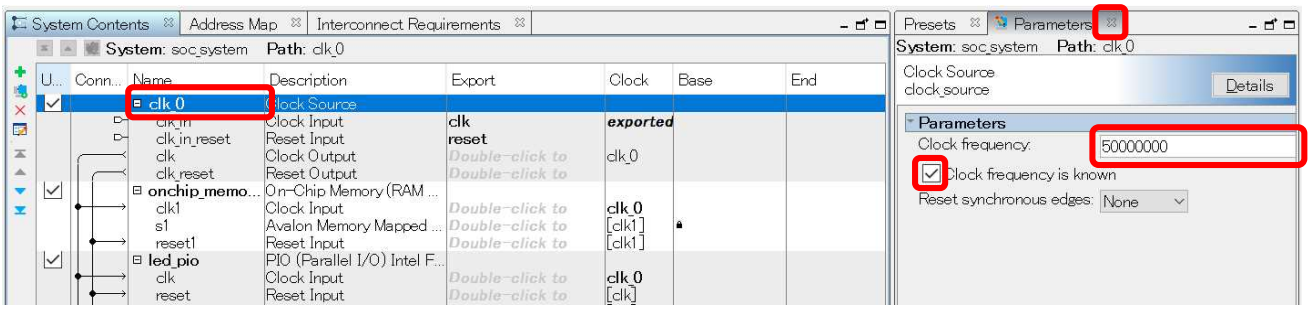


図 3-9. Clock Source の確認

____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*** タブを閉じても 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™ プロセッサー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、アルテラ® 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。

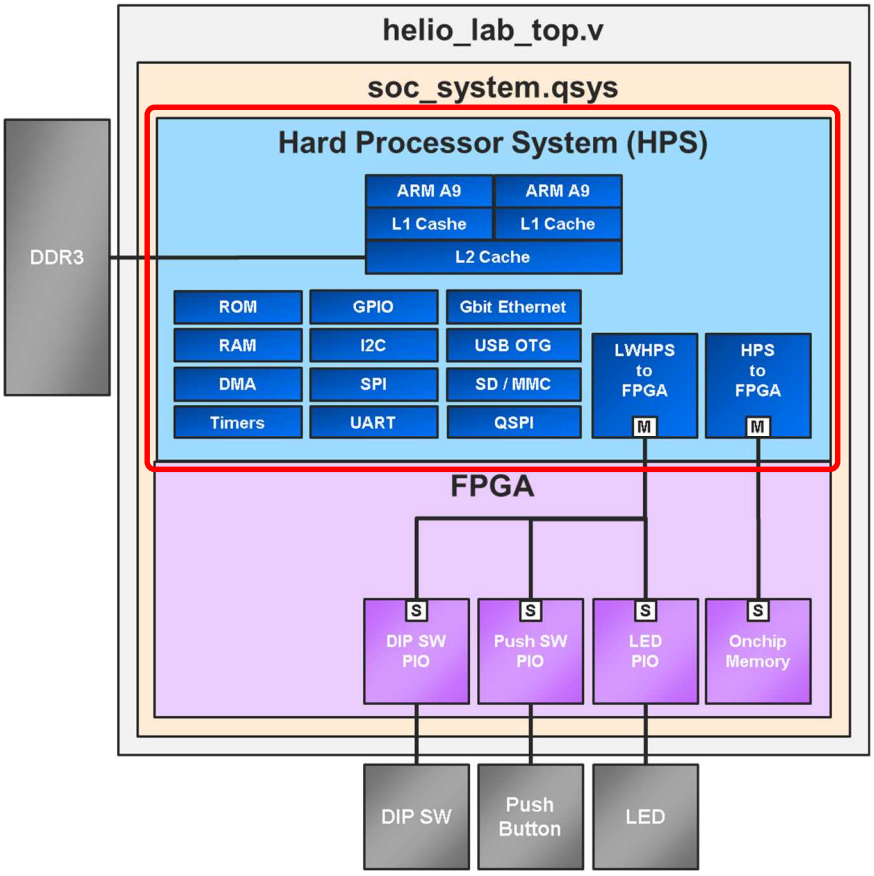


図 3-10. Platform Designer システムに追加する HPS ブロック

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。

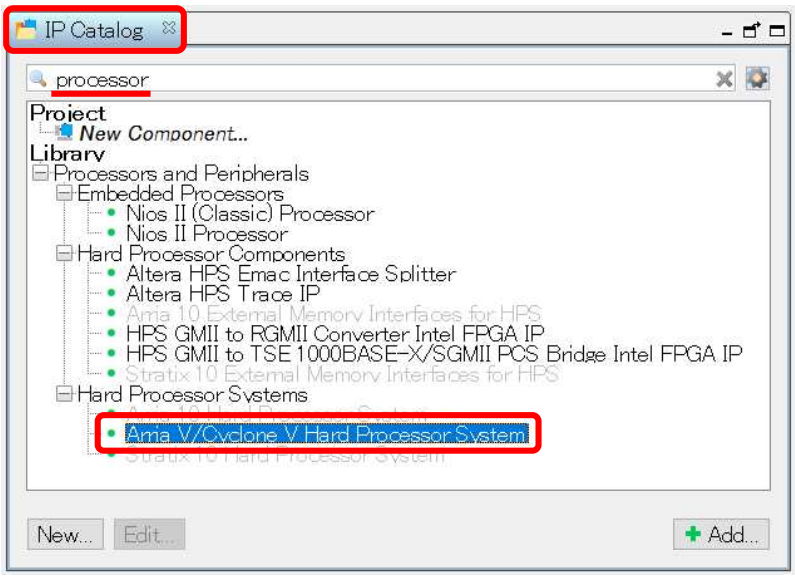


図 3-11. IP Catalog の検索ボックス

____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。

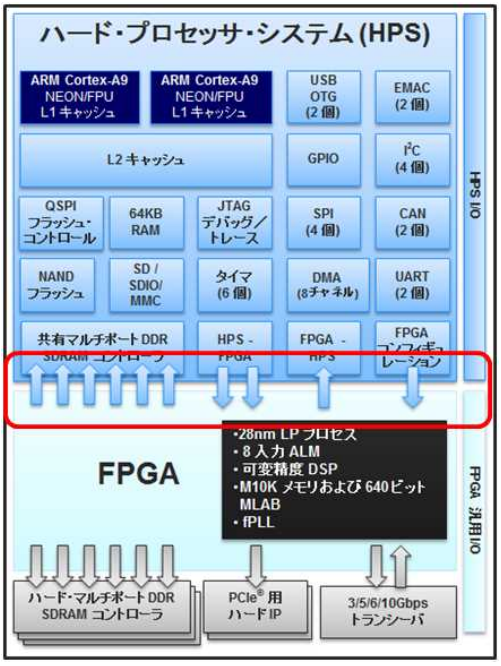


図 3-12. HPS のペリフェラルと FPGA との内部バス

____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。

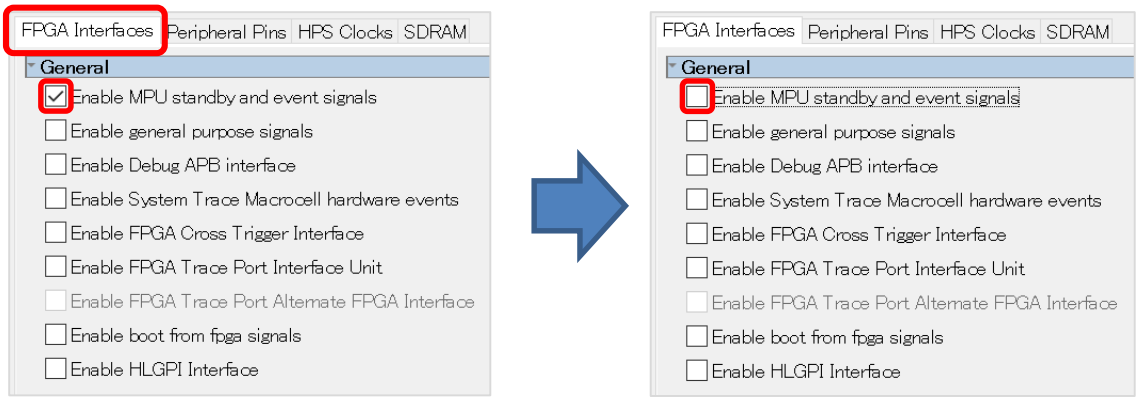


図 3-13. FPGA Interface タブの設定

> ⓘ **Note**:
> これは、マイクロ・プロセッサー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ー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。

> ⓘ **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® プロセッサー 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。

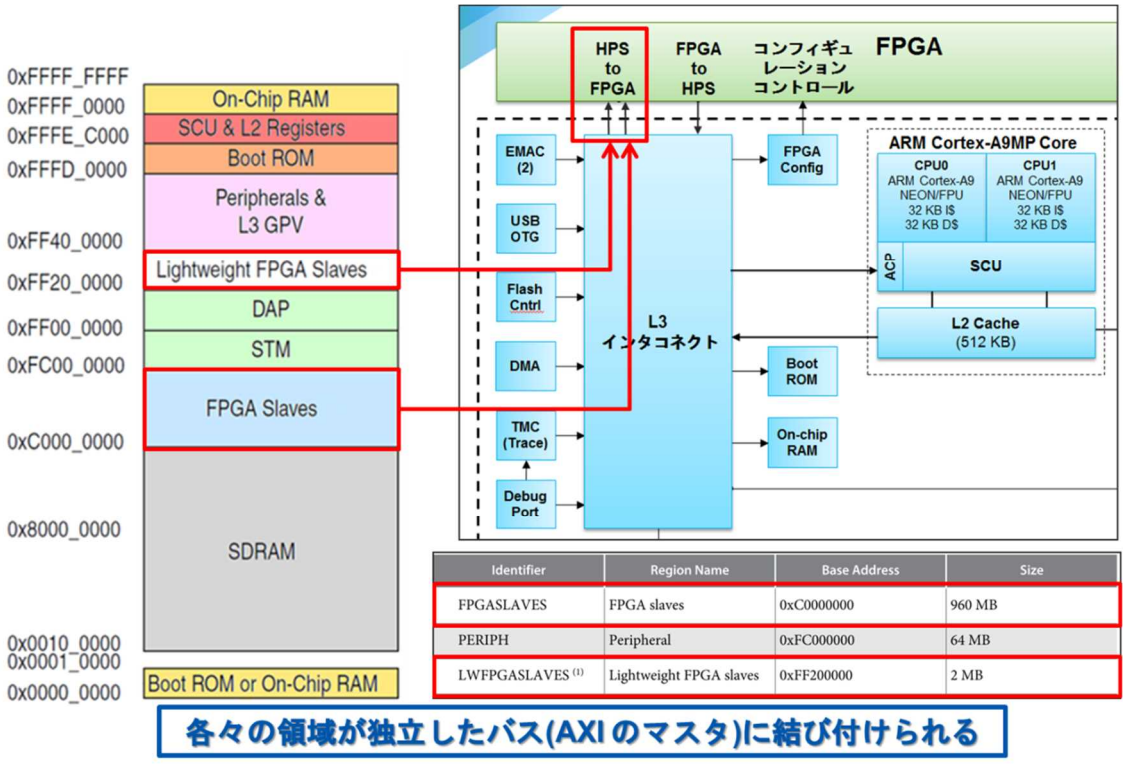


図 3-14. HPS と FPGA の内部バスと Arm から見たアドレスマップ

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（Cyclone® V SoC／Arria® V SoC 編）

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。

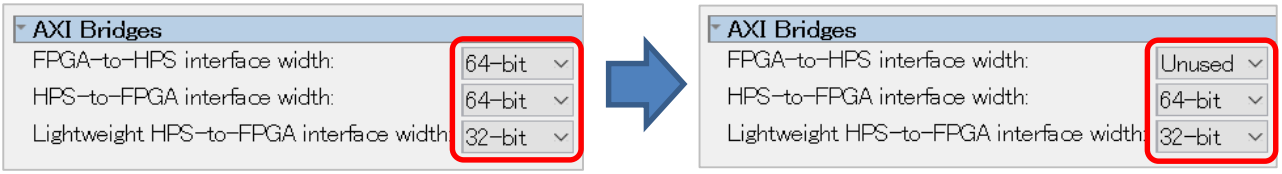


図 3-15. AXI Bridges の設定

> ⓘ **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 へ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）を介さないので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。

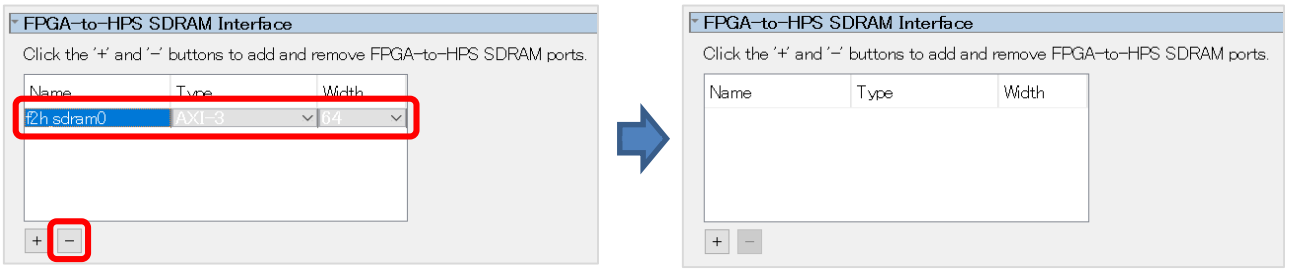


図 3-16. FPGA-to-HPS SDRAM インターフェイスの設定

____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**の下の全ての行が ***No*** と表示されていることを確認します。

> ⓘ **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® プロセッサー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）:

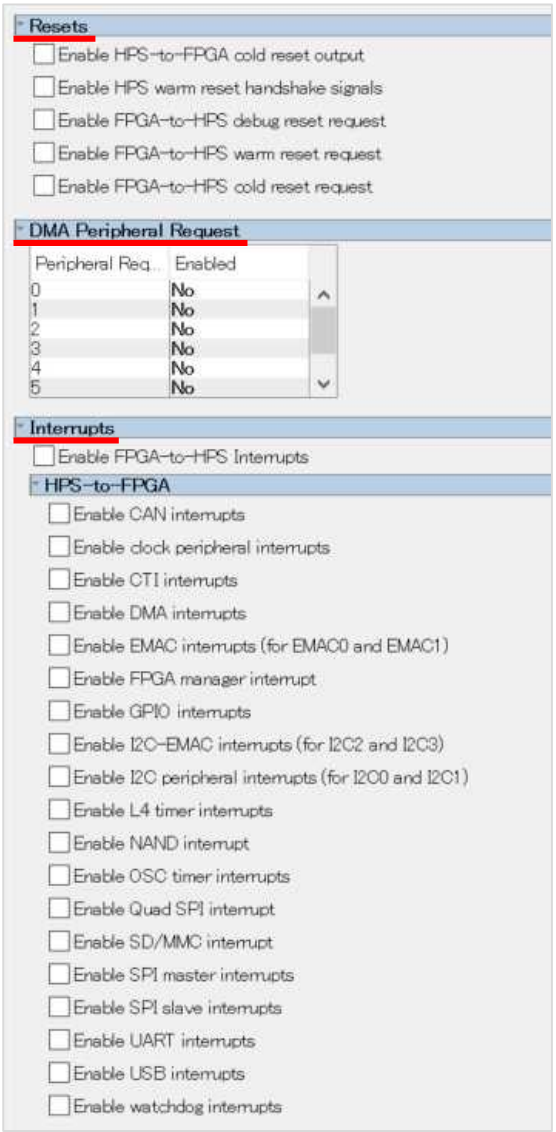


図 3-17. Resets / DMA / Interrupts の設定

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 内部にハードコーデットされている 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。

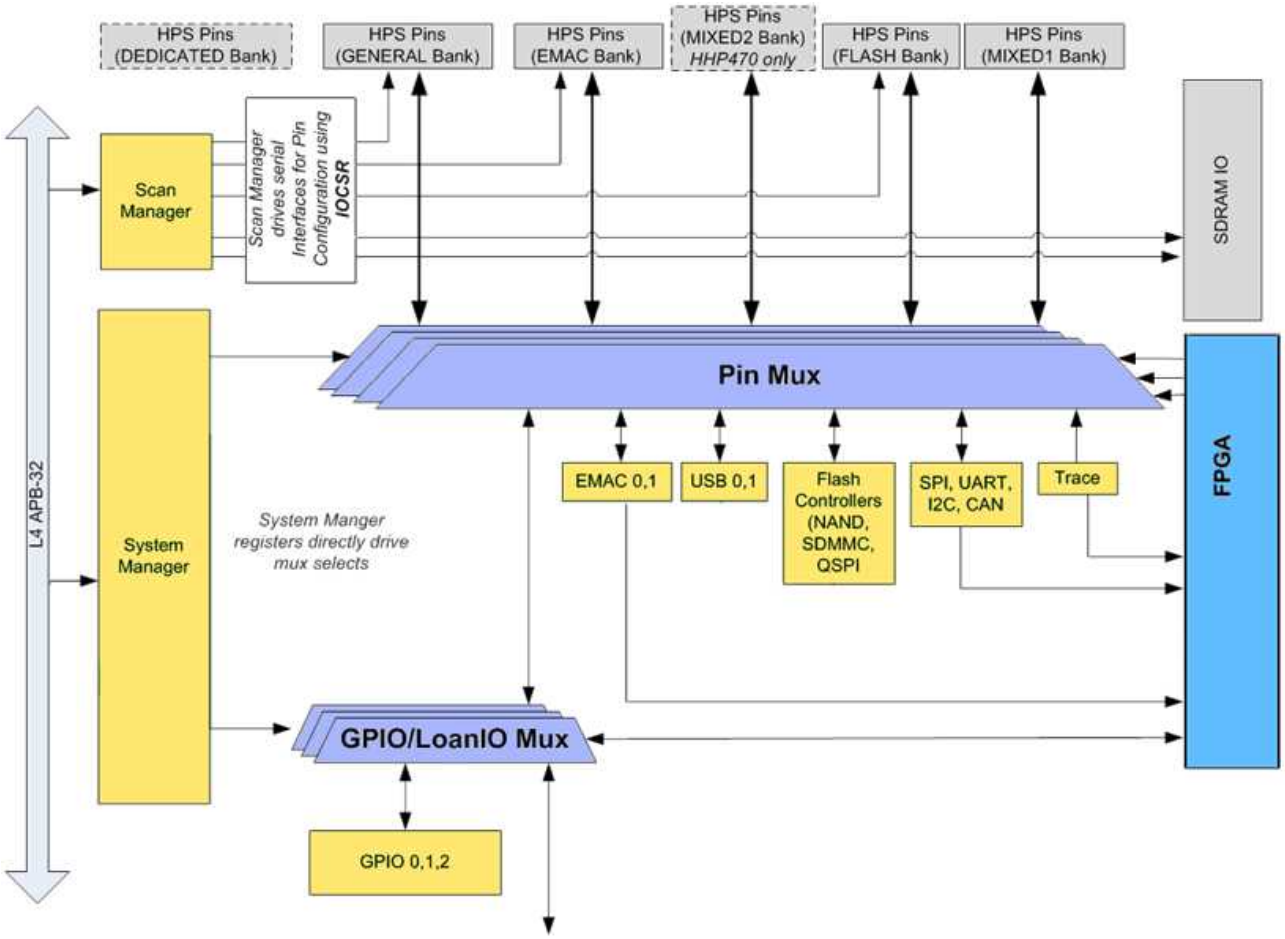


図 3-18. HPS I/O のピン・マルチプレクサー

____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。

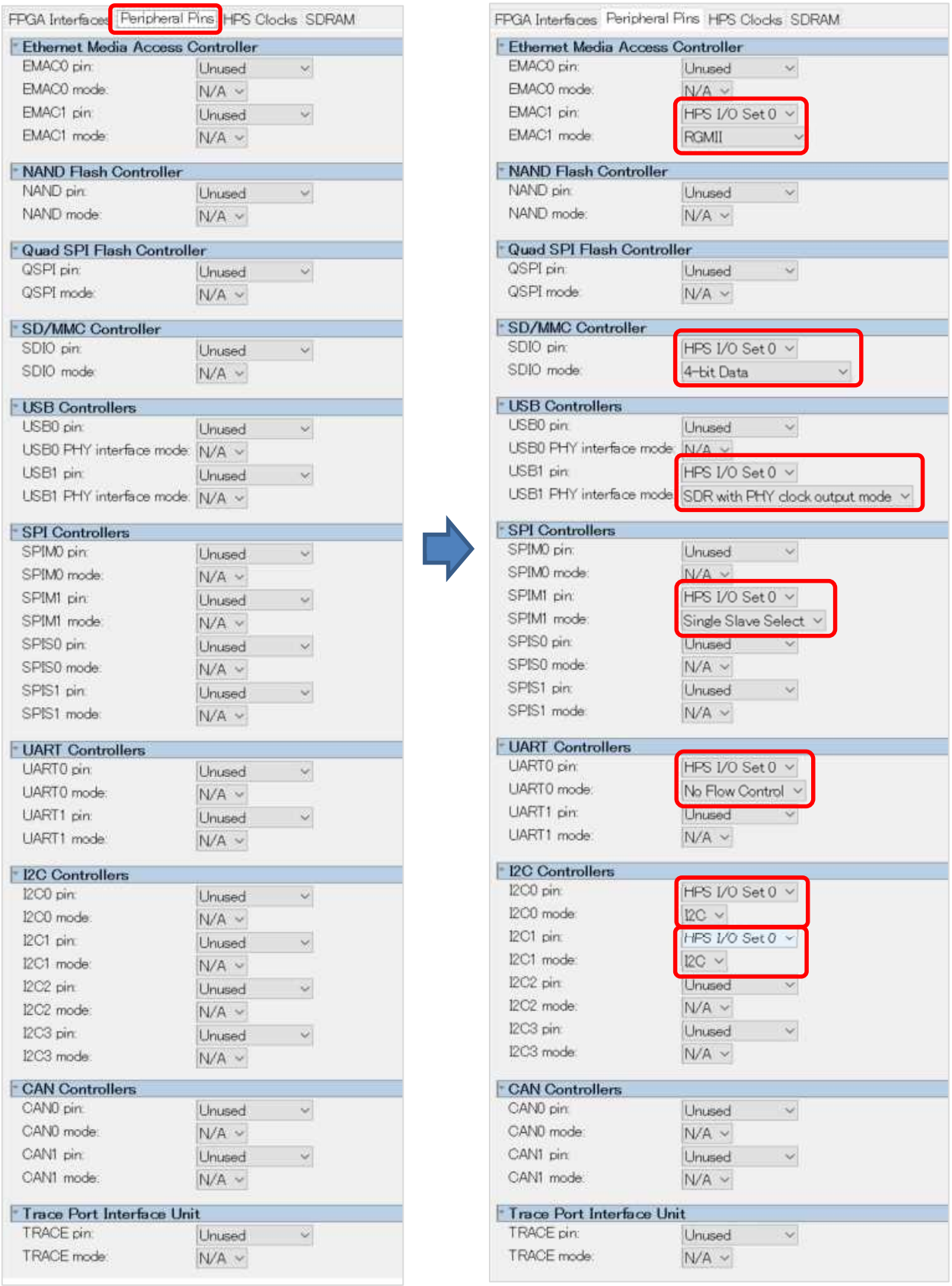


図 3-19. HPS ペリフェラルの設定

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。

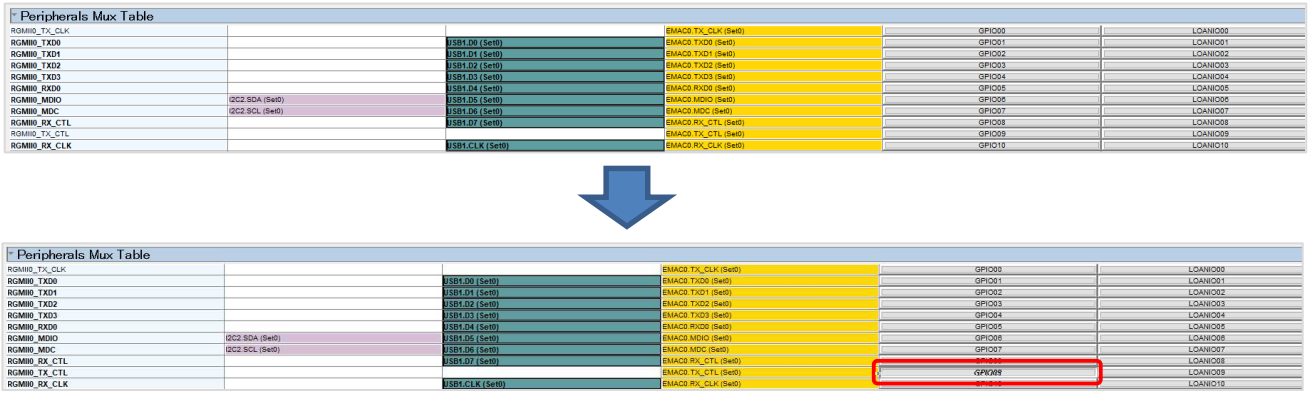


図 3-20. HPS GPIO09 の設定

> Ⓟ **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。

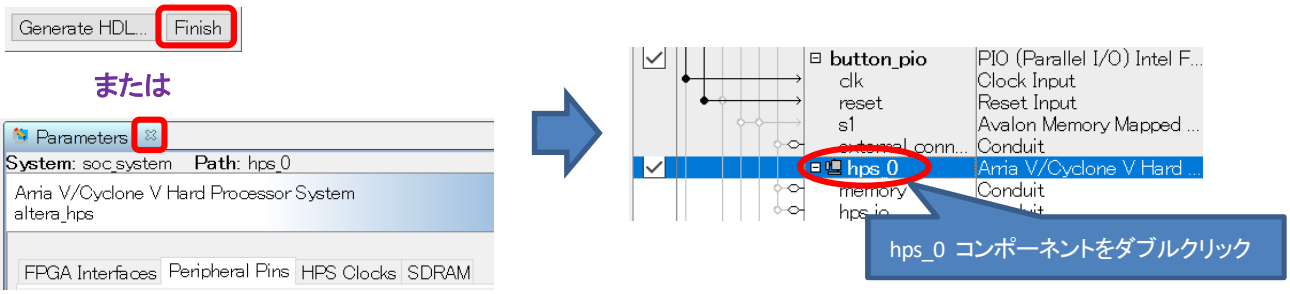


図 3-21. GPIO 設定時にクリックできなかった場合の対応方法

____17. 同様に ***GPIO35***、***GPIO40***、***GPIO53***、***GPIO54***、***GPIO61*** を有効にします。

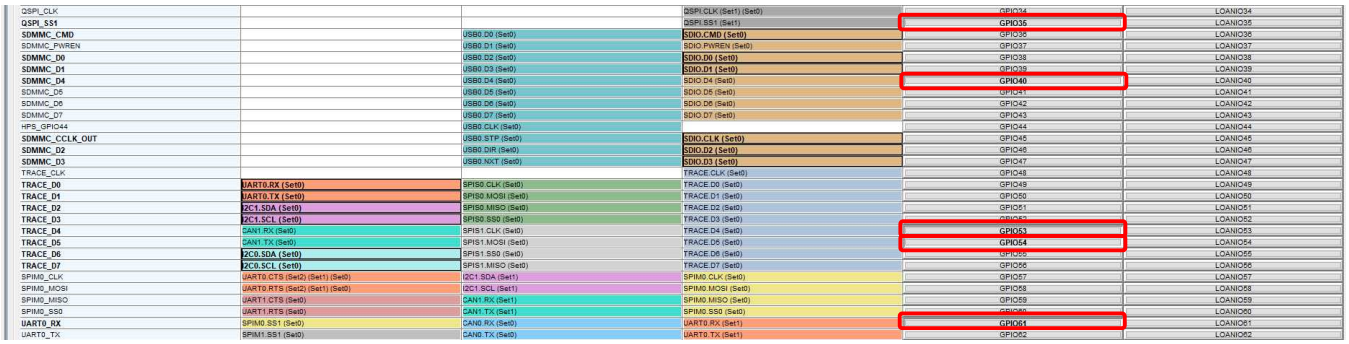


図 3-22. HPS GPIO の設定

____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）。

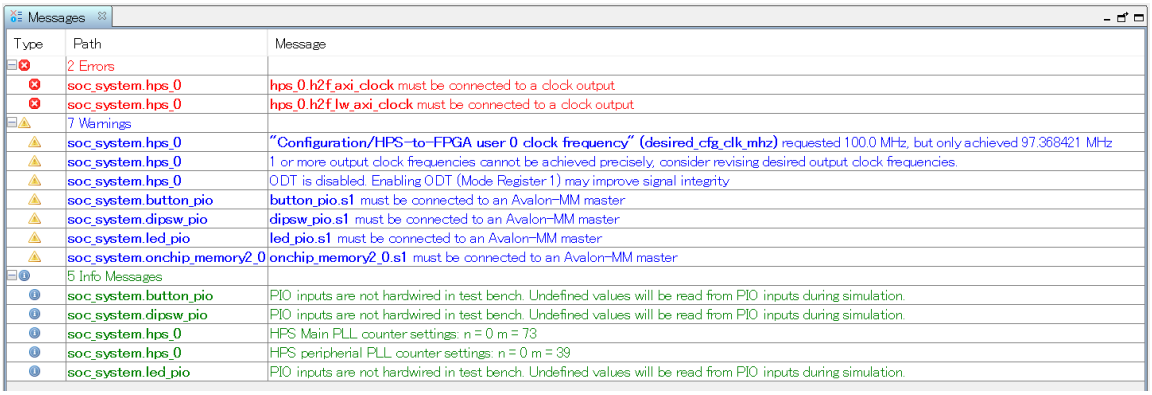


図 3-23. ピン競合エラーが無いときの表示例

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ますので、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。

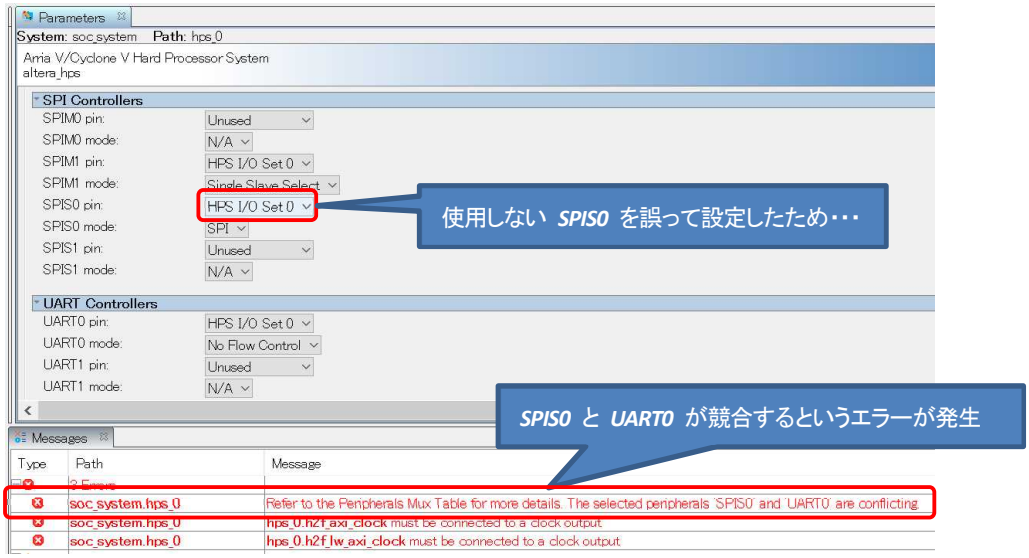


図 3-24. ピン競合エラーがあるときの表示例

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「4. 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。

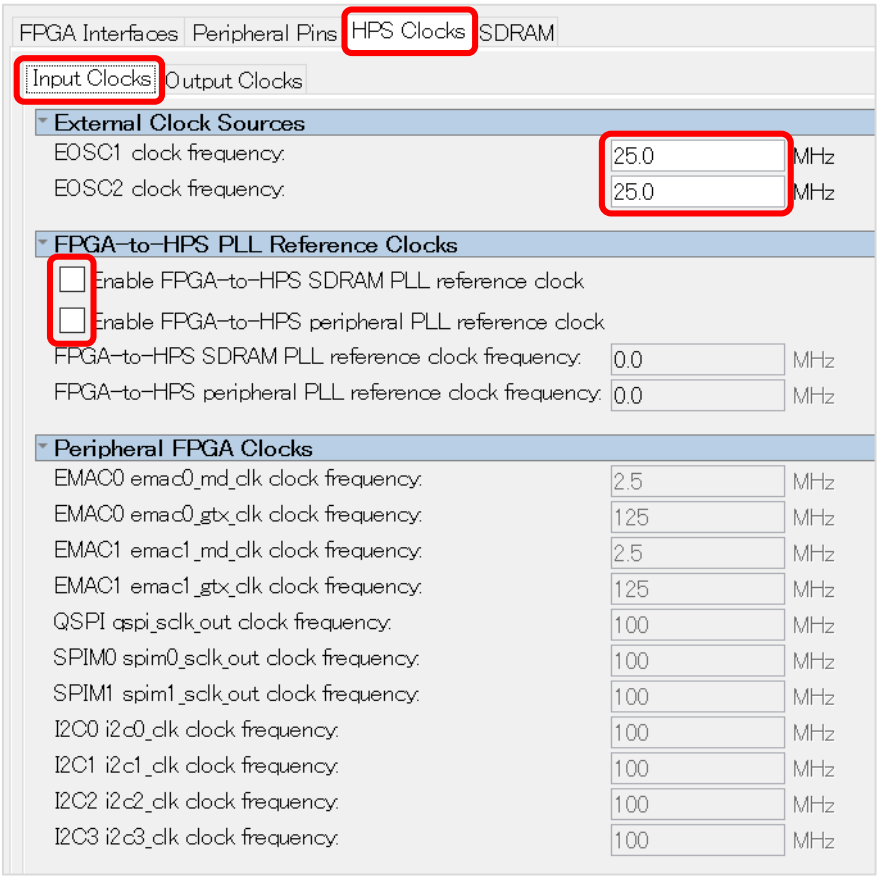


図 3-25. HPS to FPGA Clock の設定

____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。

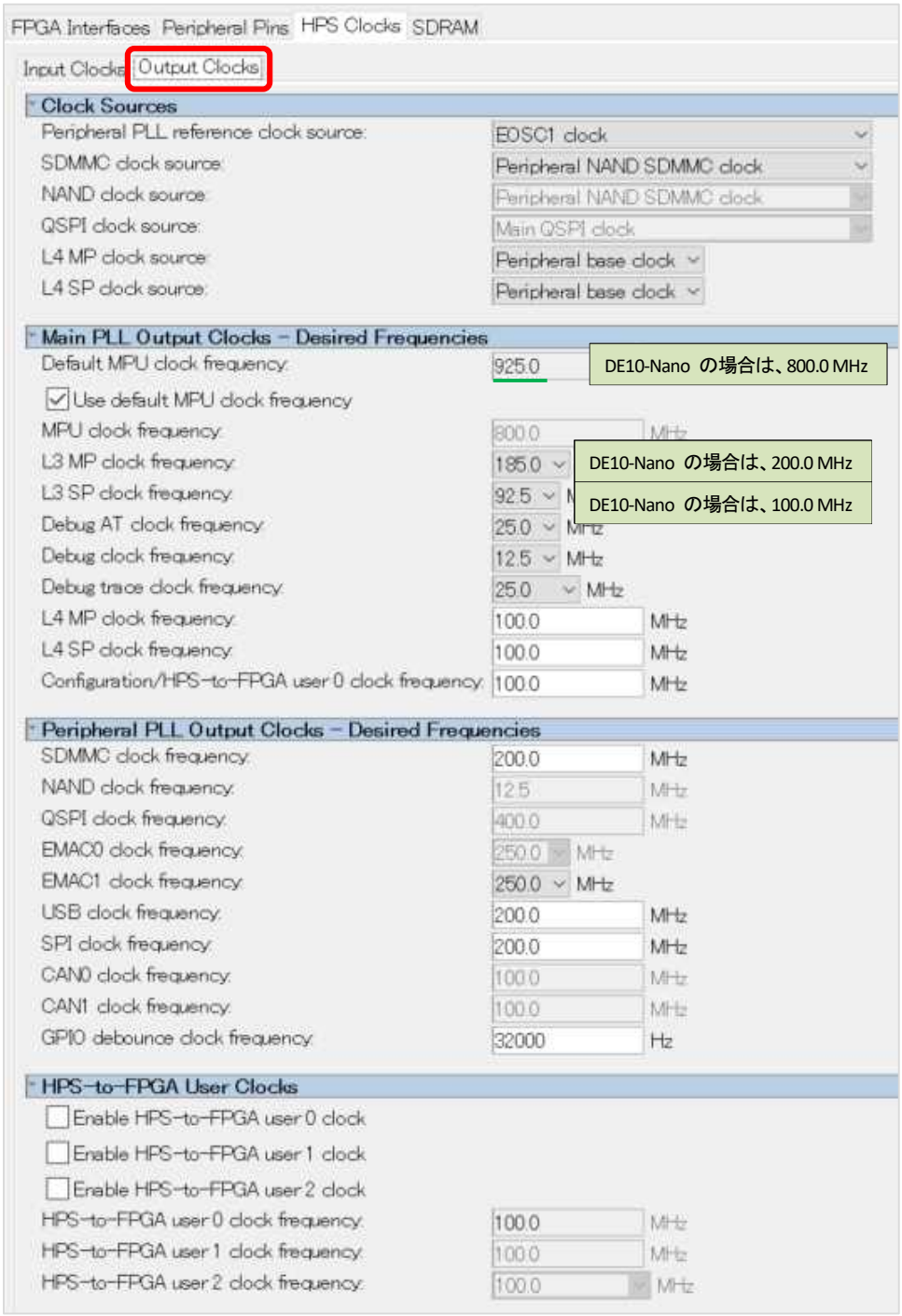


図 3-26. HPS to FPGA Clock の設定

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）。

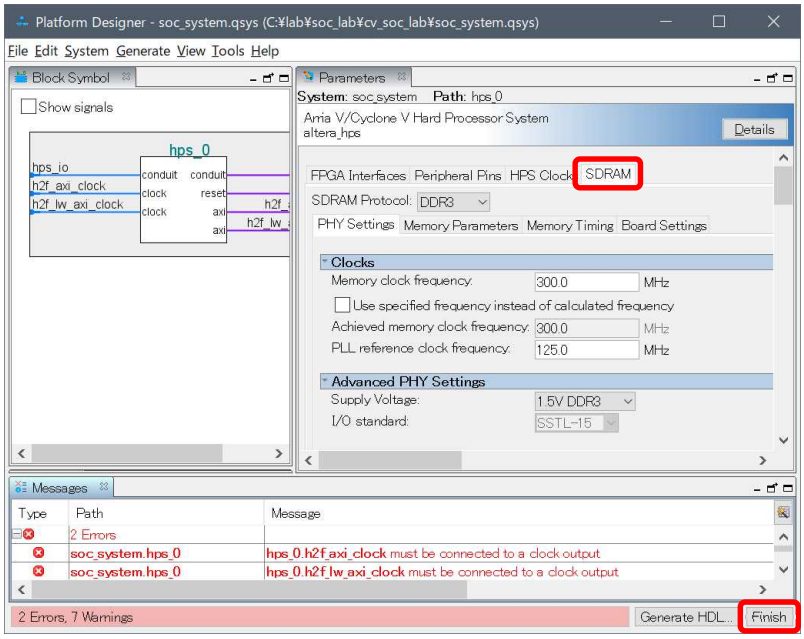


図 3-27. Parameters ウィンドウ表示の準備

____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。

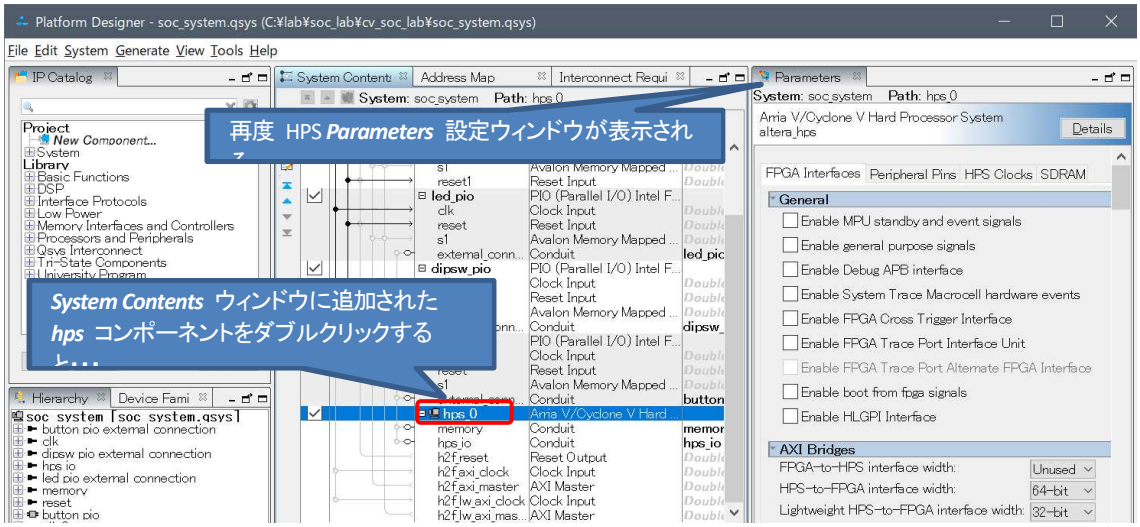


図 3-28. HPS パラメーター設定ウィンドウを再表示

____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。

> ⓘ **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。

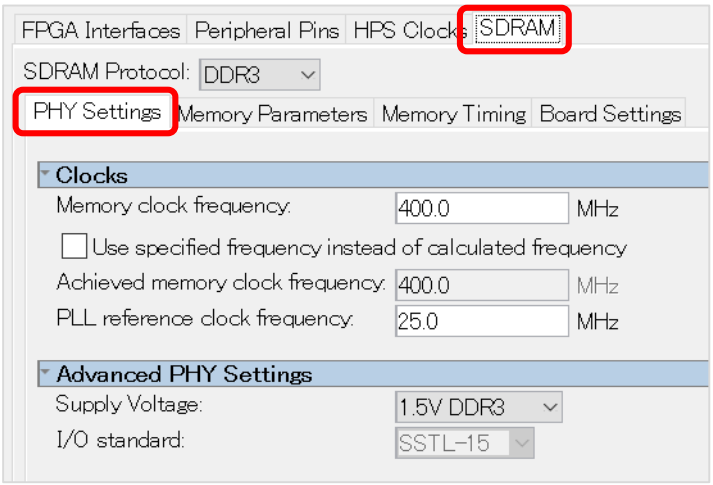


図 3-30. PHY Settings の確認

____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。

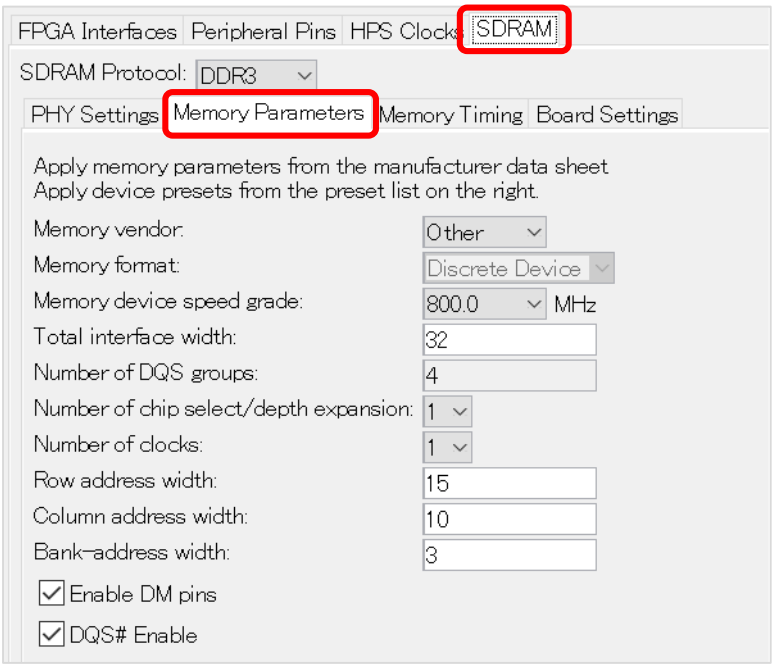


図 3-31. Memory Parameters

____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。

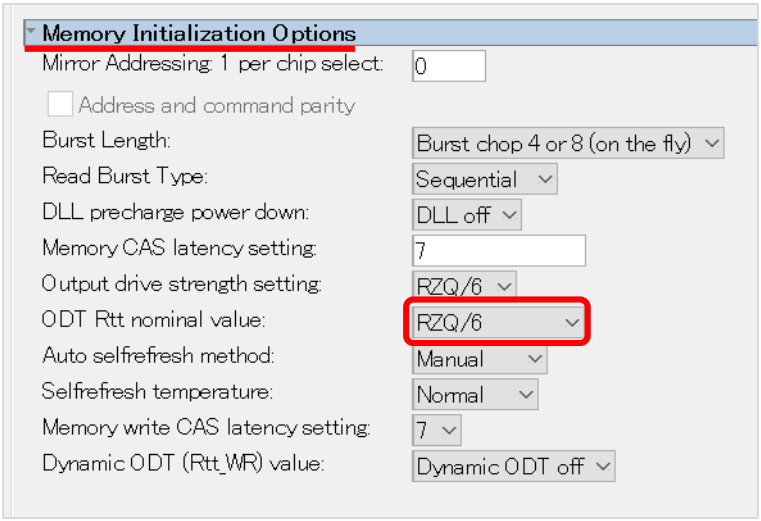


図 3-32. Memory Initialization Options

____ 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。

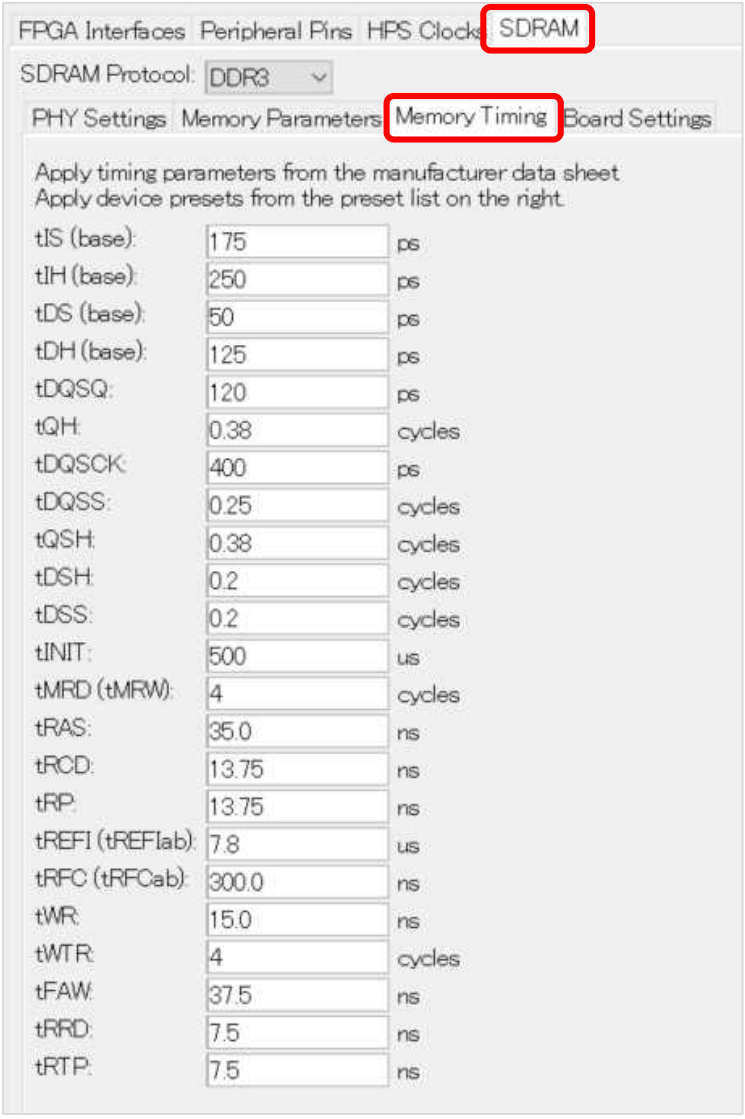


図 3-33. Memory Timing

____ 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。

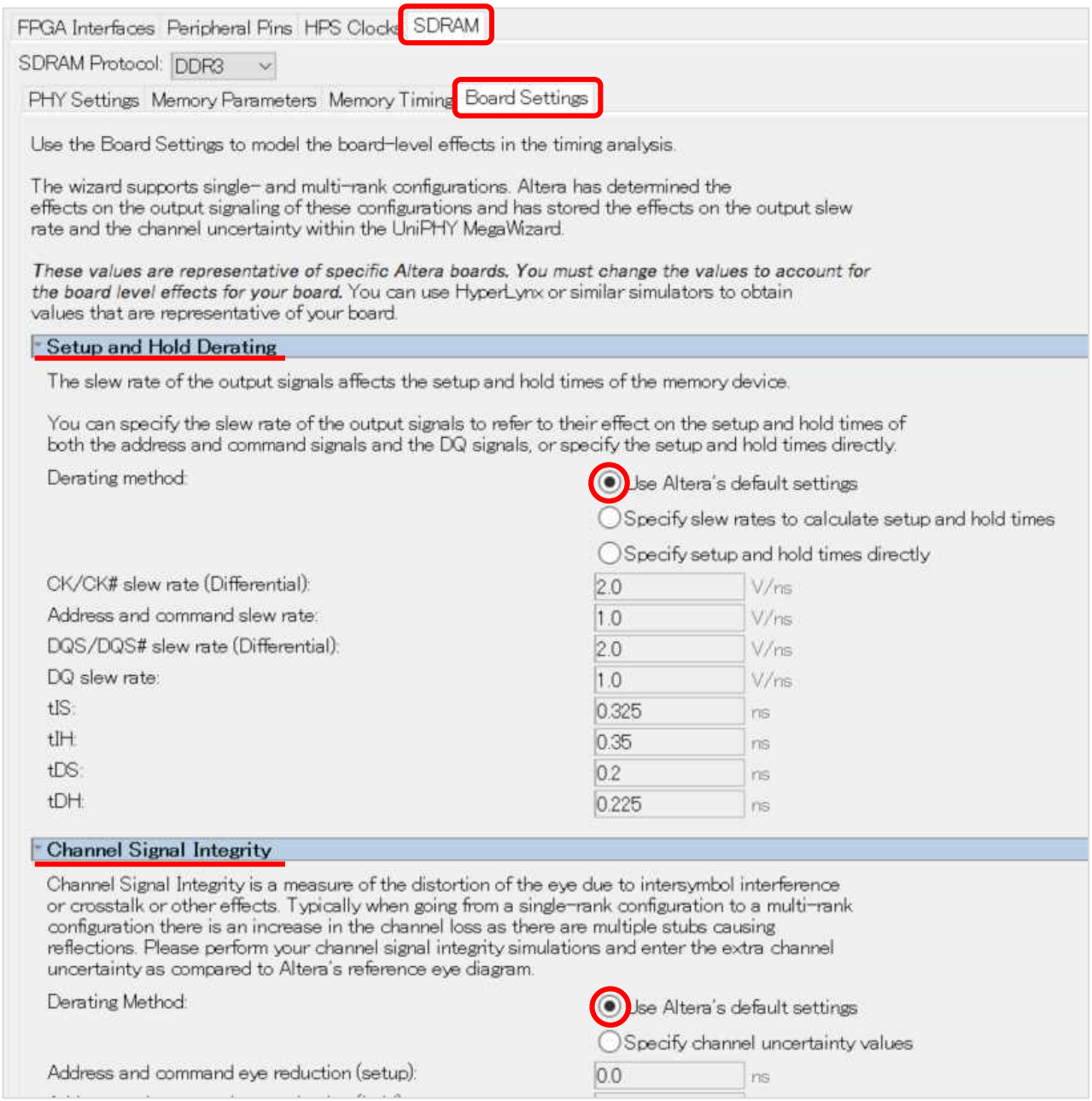


図 3-34. Board Settings（1）

____ 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。

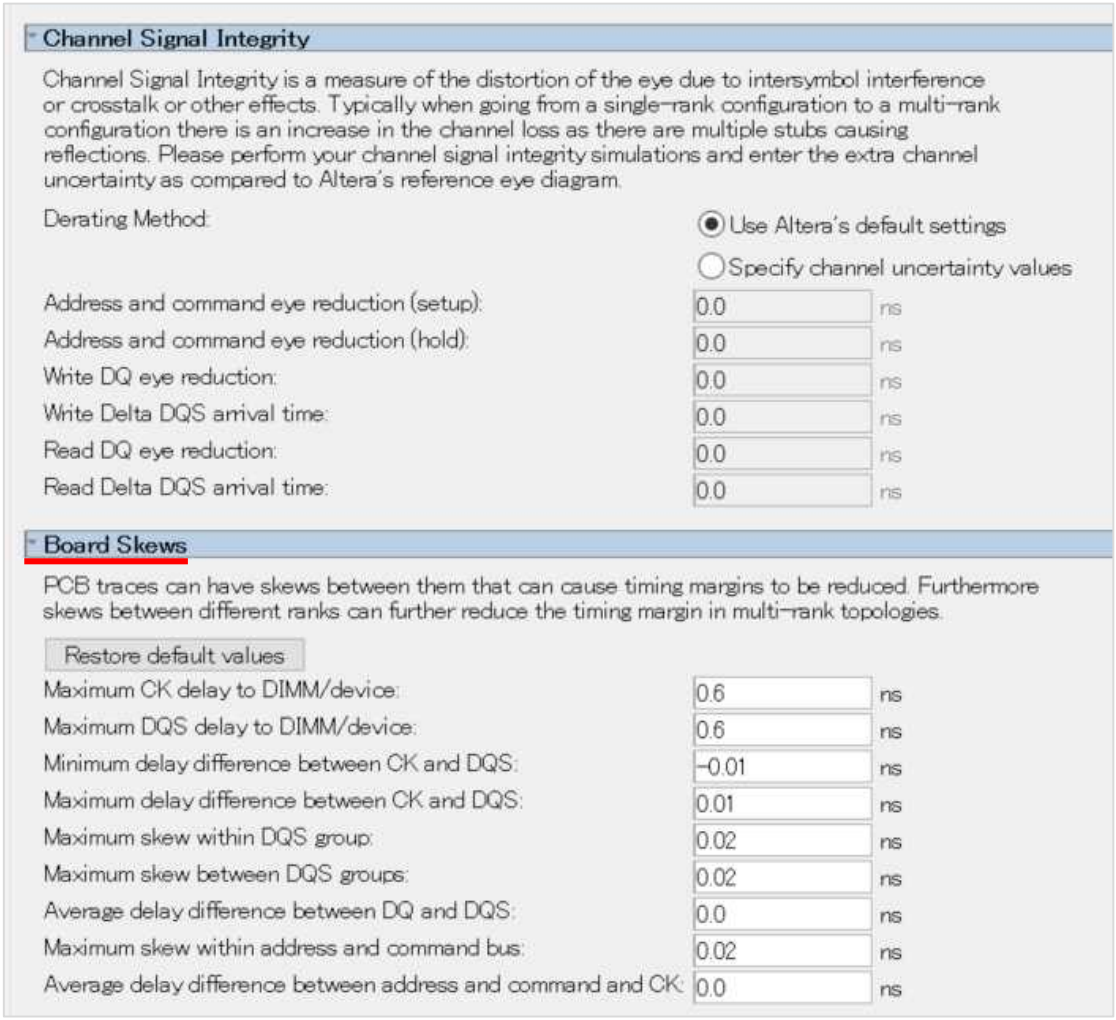


図 3-35. Board Settings（2）

____ 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。

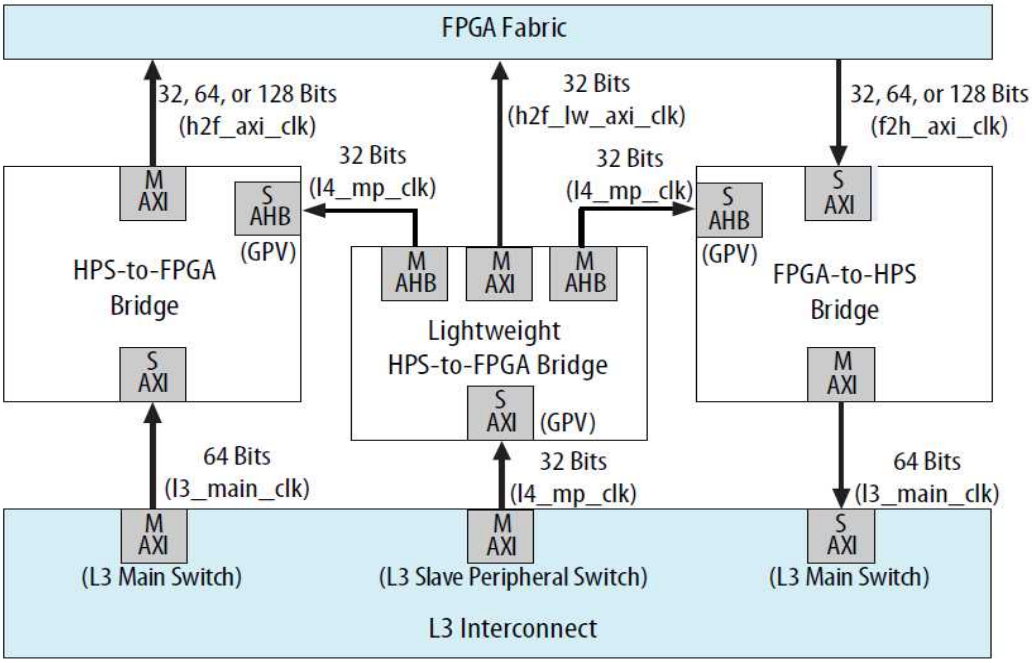


図 3-36. HPS と FPGA 間のクロック

____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_resest*** 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。

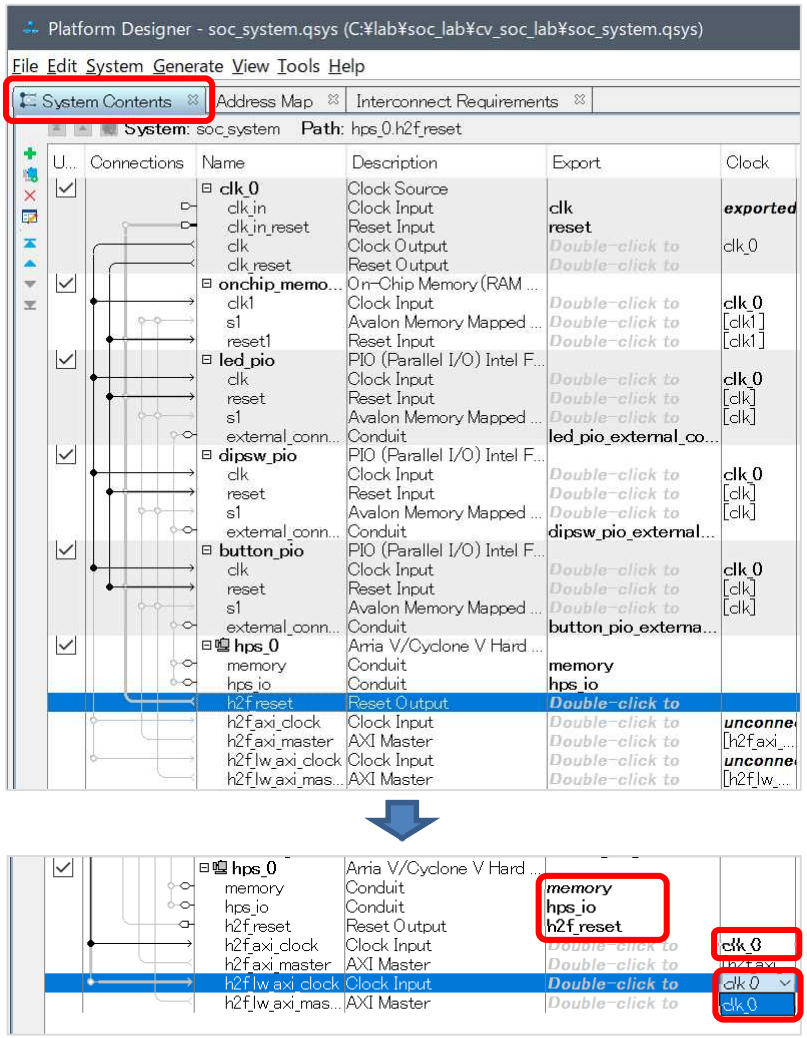


図 3-37. クロックとエクスポート信号の設定

この時点で ***clk0*** 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無くなっているはずです。

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® プロセッサー から FPGA 側の onchip_memory へアクセスすることができます。

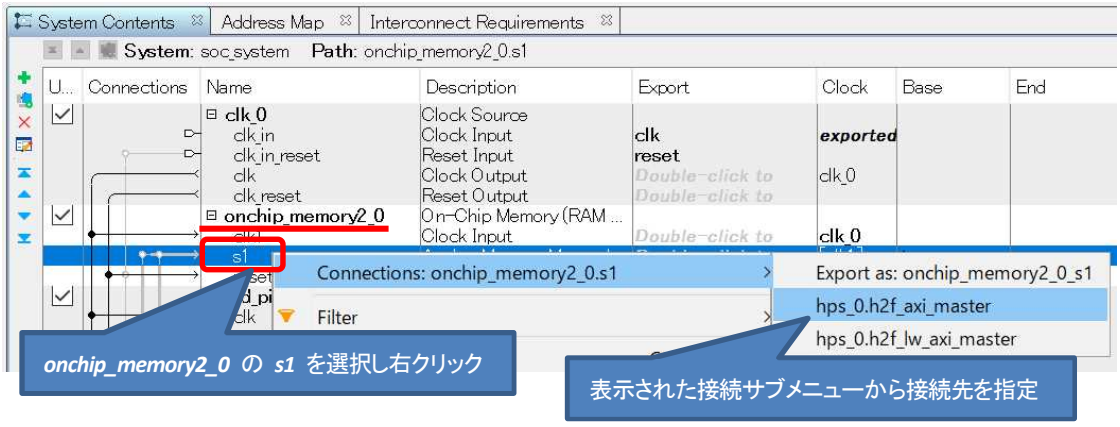


図 3-38. コンポーネント間の接続

____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。

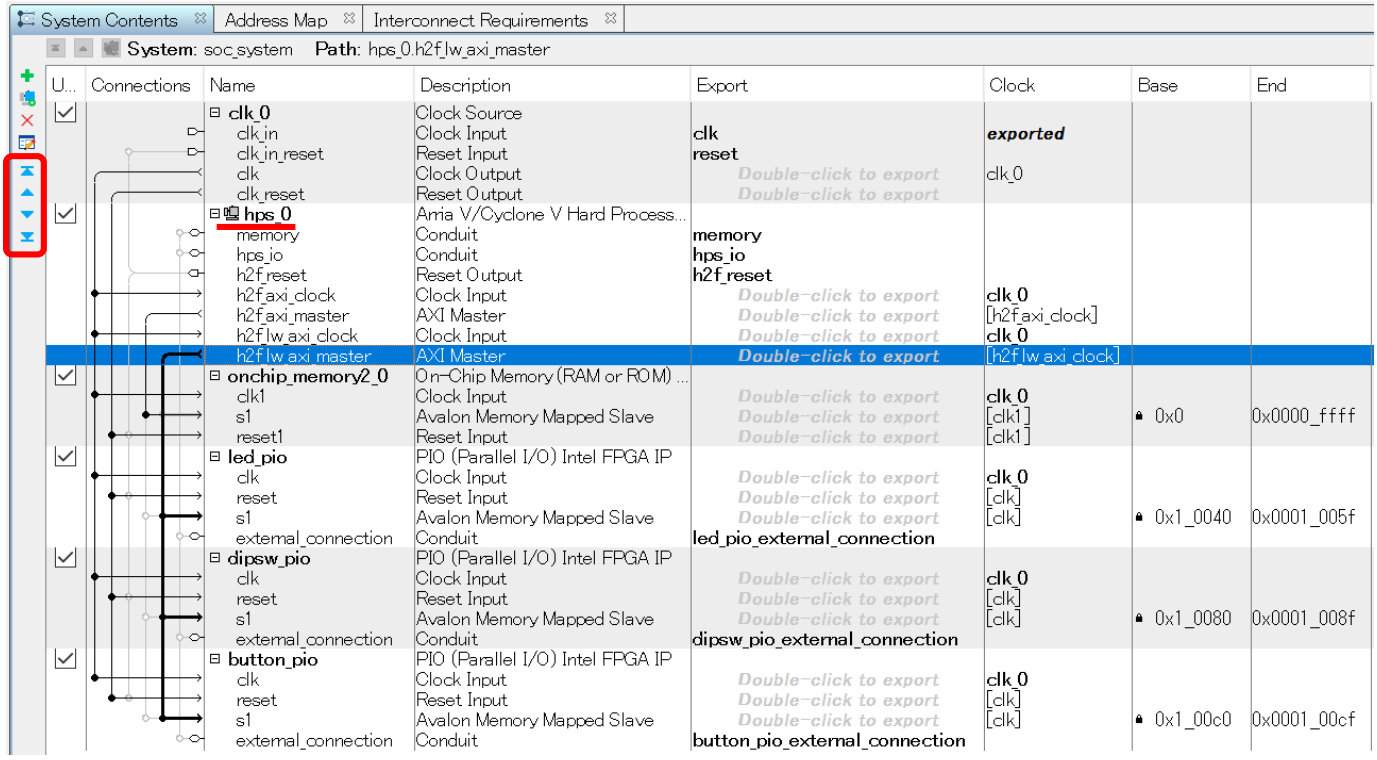


図 3-39. 設定完了後の Platform Designer システム

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。

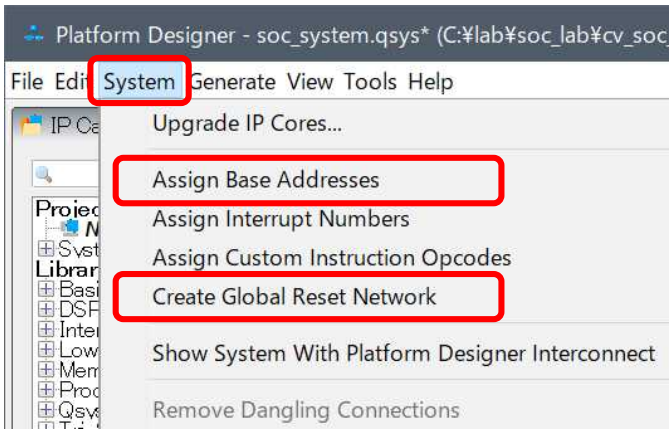


図 3-40. リセットの一括接続とベースアドレスの自動割り当て

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。

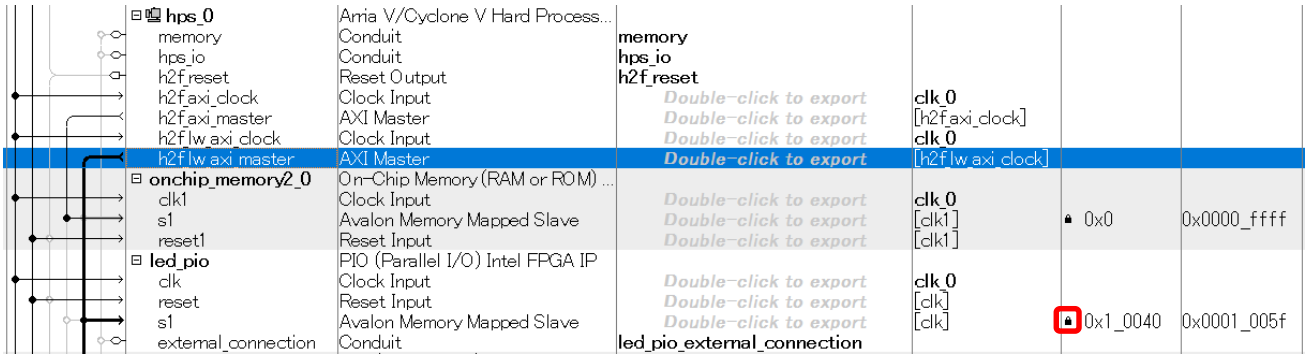


図 3-41. ベースアドレスの固定

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。

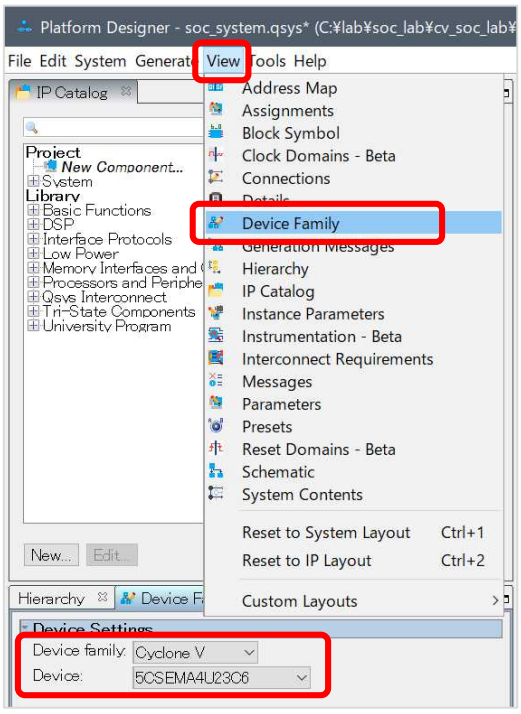


図 3-42. Device Family タブ

____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。

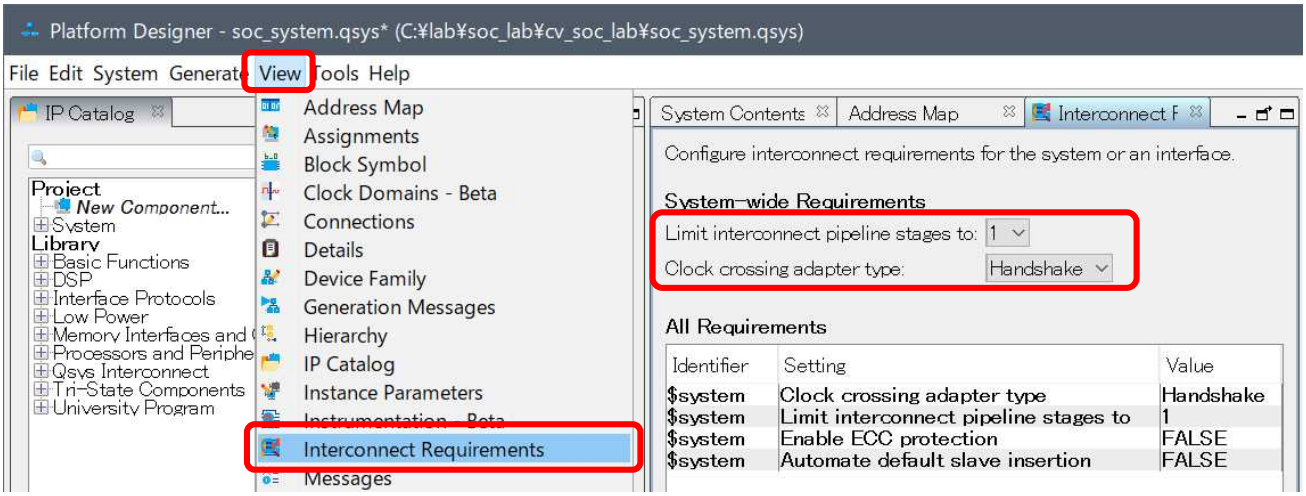


図 3-43.　プロジェクト・パラメーターの設定

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。

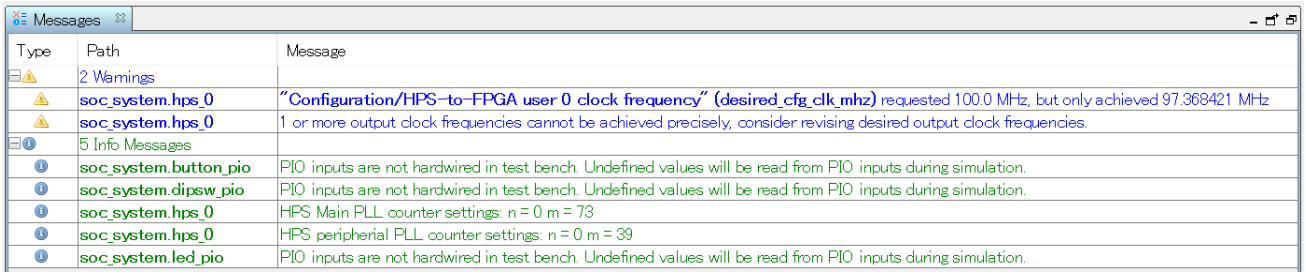


図 3-44. Message ウィンドウの表示

____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。

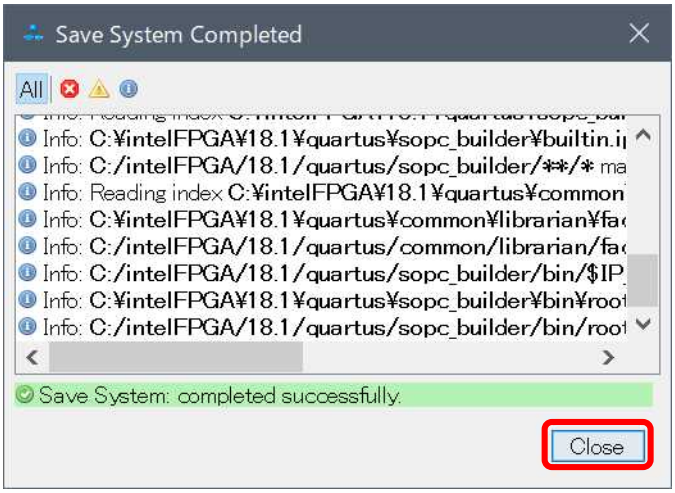


図 3-45. Platform Designer システムの保存

____3. ***Generate*** メニュー ⇒ ***Generate HDL*** を選択します。

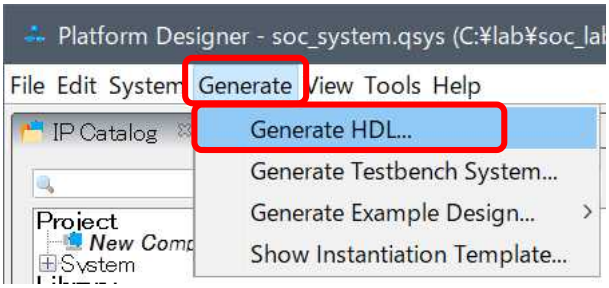


図 3-46. Platform Designer システムの生成

____4. ***Generation*** ウィンドウの設定を確認し、[***Generate***] を実行します。

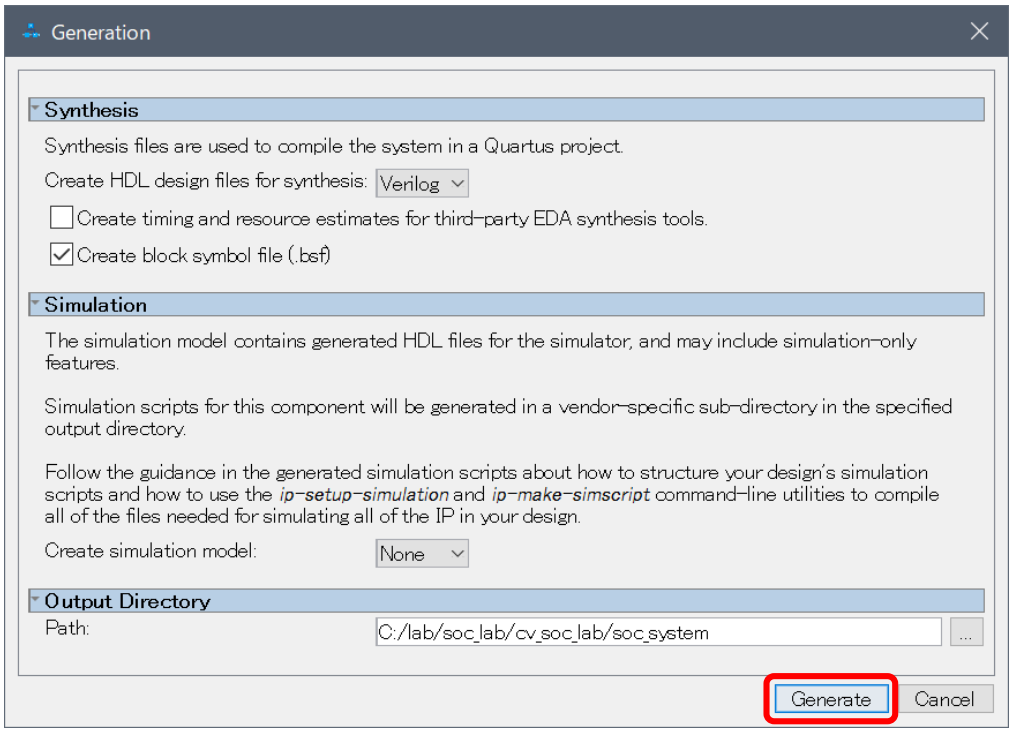


図 3-47. Platform Designer システムの Generate 実行画面

____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。

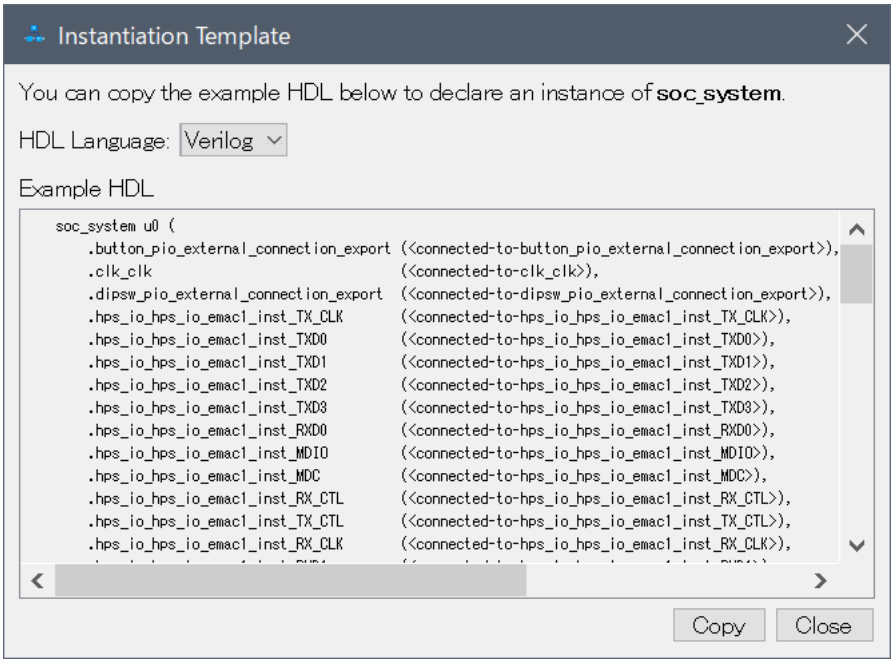


図 3-48. Platform Designer システムのインスタンス例

____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。

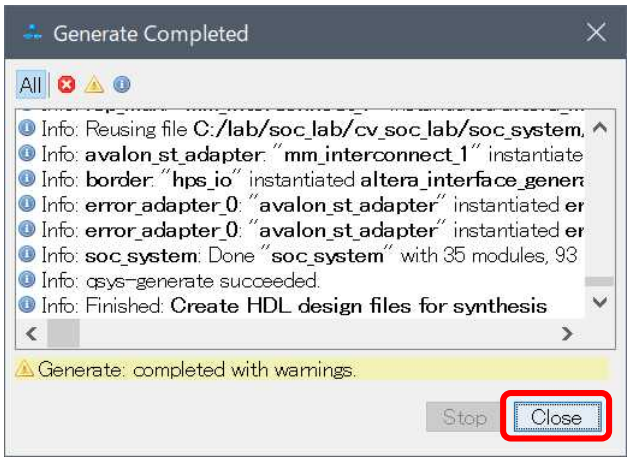


図 3-49. Platform Designer システムの生成完了

____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*** カテゴリーが選択された状態で表示されます）。

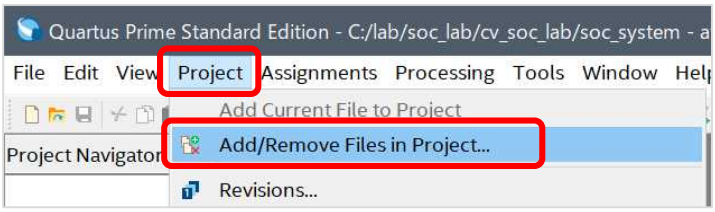


図 3-50. Add/Remove Files in Project を選択

____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*** フォルダーを参照します。

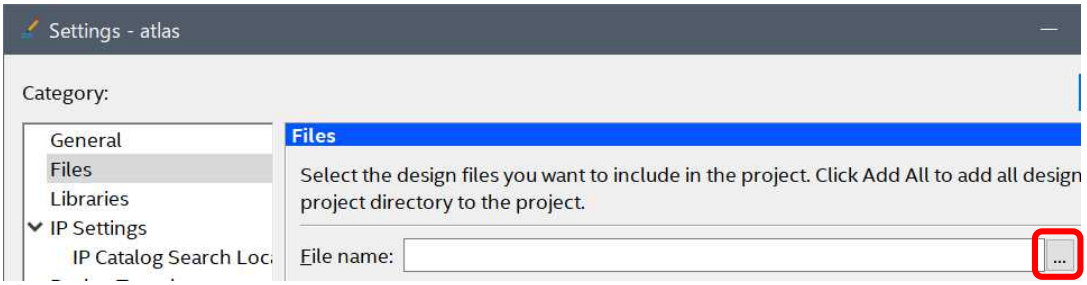


図 3-51. Settings ダイアログボックス

____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。

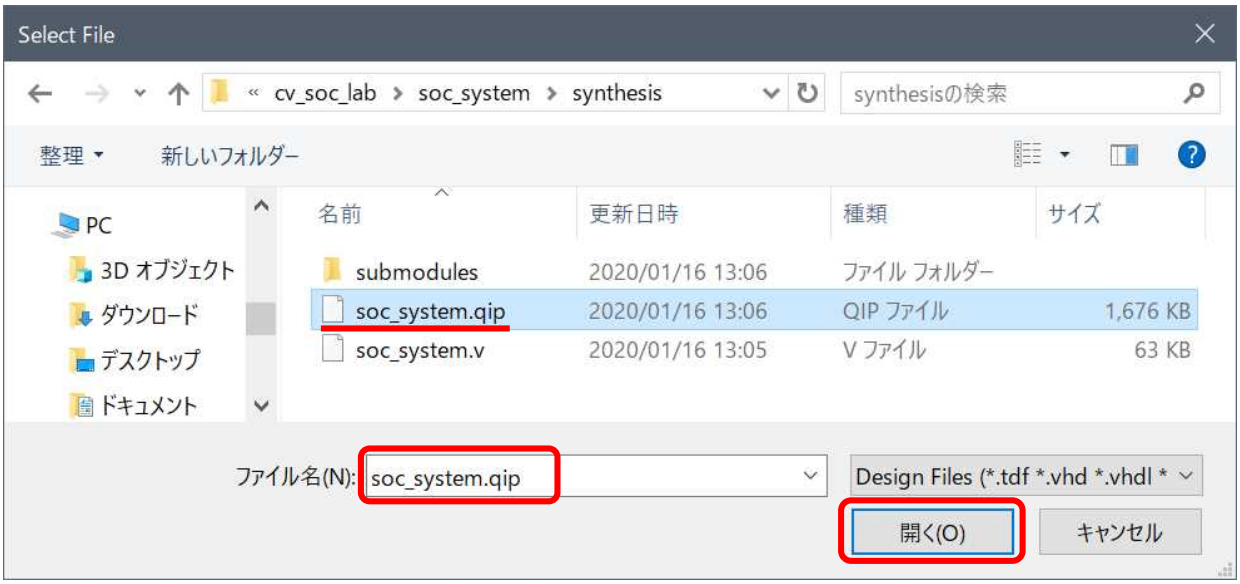


図 3-52. Qip ファイルの指定

____10. ファイルが追加されたことを確認します。

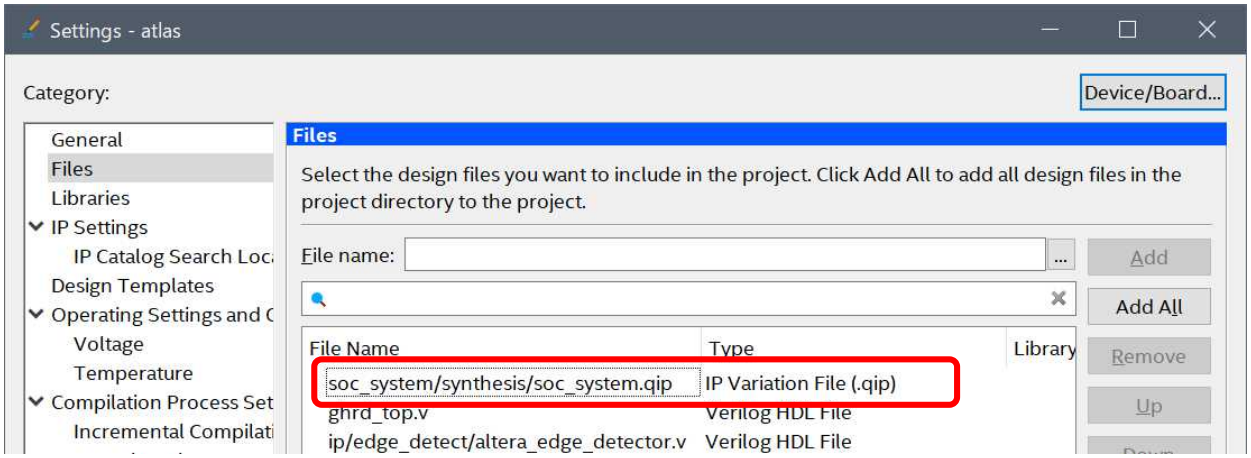


図 3-53. Qip ファイルの登録

____11. ***Settings*** ダイアログボックスを［***OK***］ボタンで閉じます。

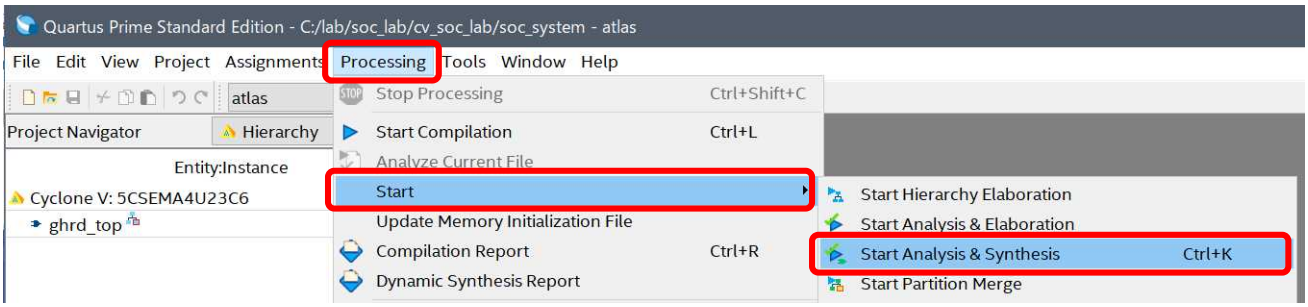
3-11. ステップ 11 ： ピン・アサインメントの設定と Quartus® Prime プロジェクトのコンパイル

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）。



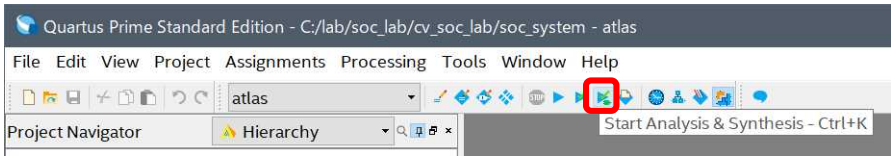


図 3-54. Start Analysis & Synthesis

____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。

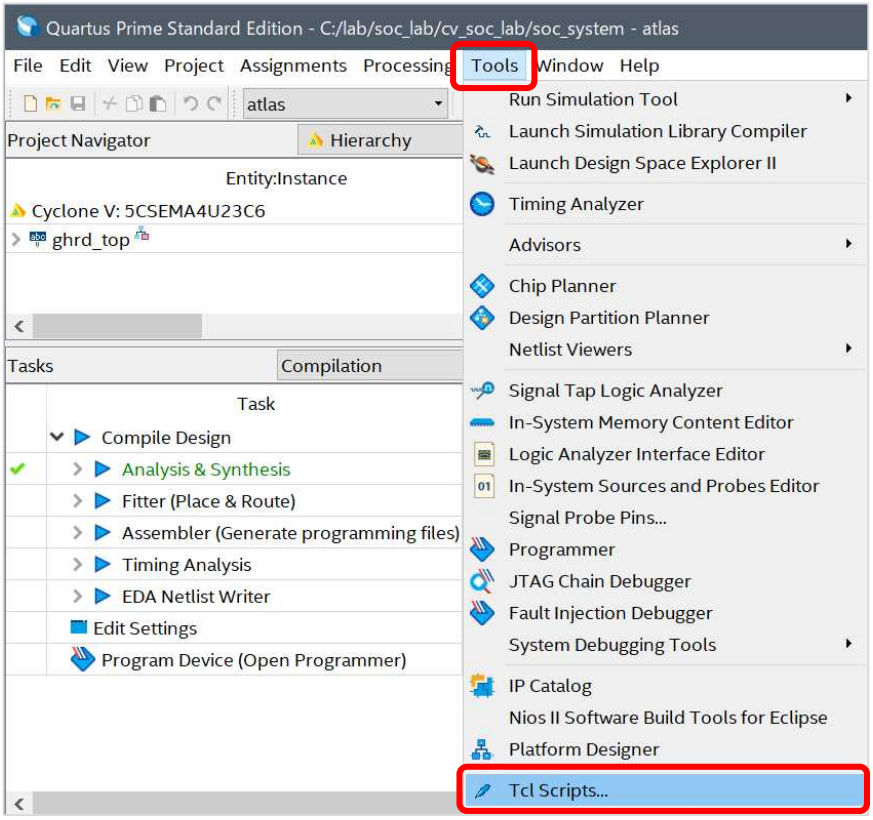


図 3-56. Tcl Scripts ウィンドウの起動

____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。

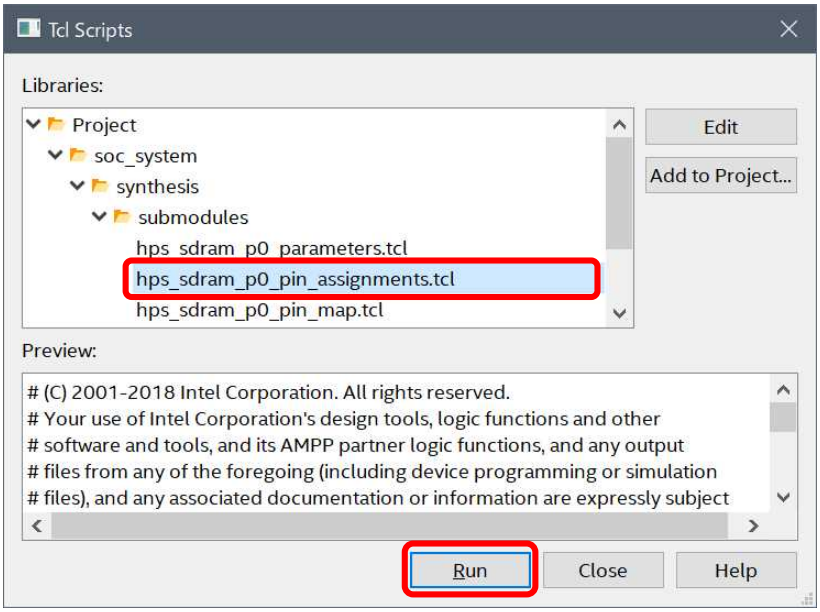


図 3-57. Tcl Script の実行

____5. 完了したら［***OK***］ボタンをクリックします。

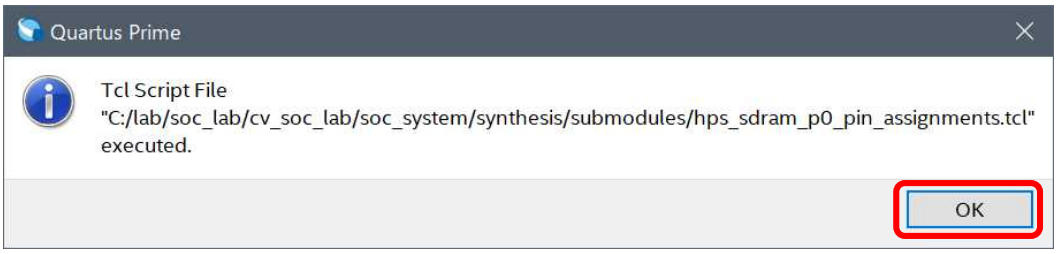


図 3-58. Tcl Script の完了

____6. Tcl Scripts ウィンドウを［***Close***］します。

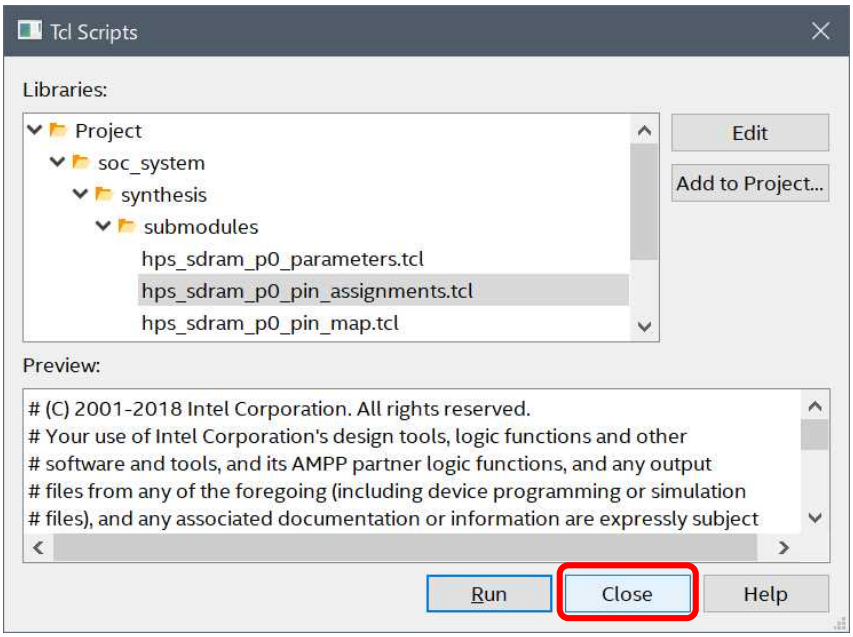


図 3-59. Tcl Scripts ウィンドウの Close

____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。

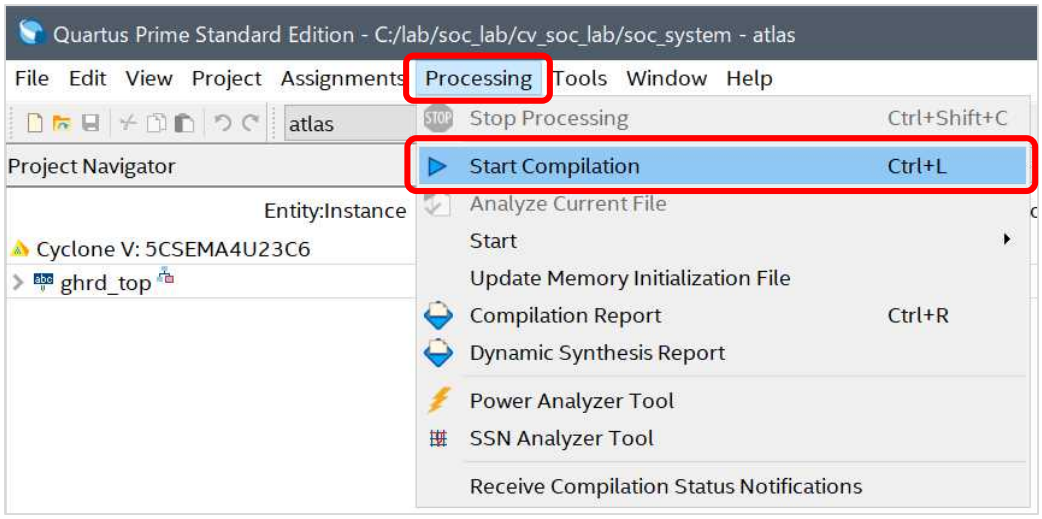


図 3-60. Start Compilation の実行

____8. コンパイルの完了を確認します。

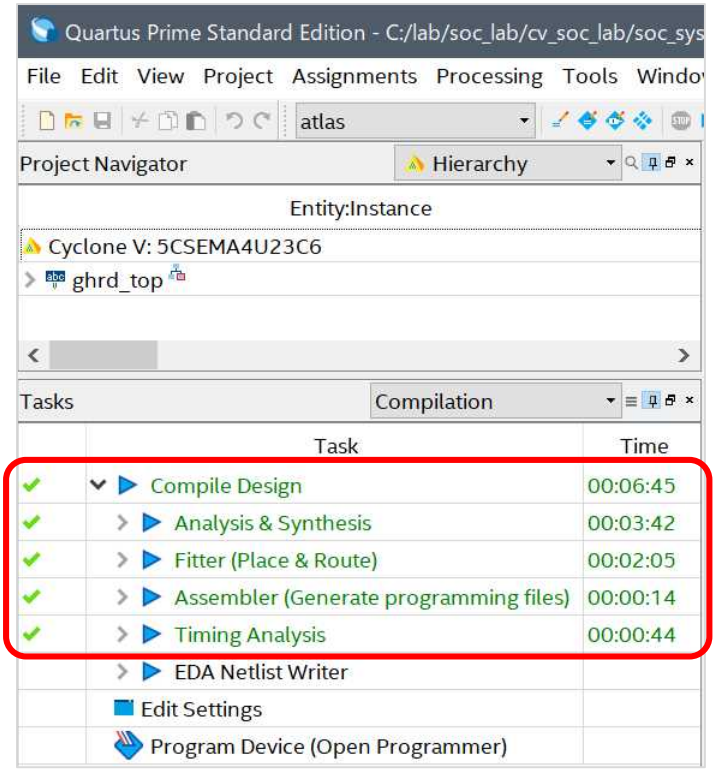


図 3-61. コンパイルの完了

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® プロセッサー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、「3. 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。

> Ⓟ **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）をベースにアルテラ® 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。

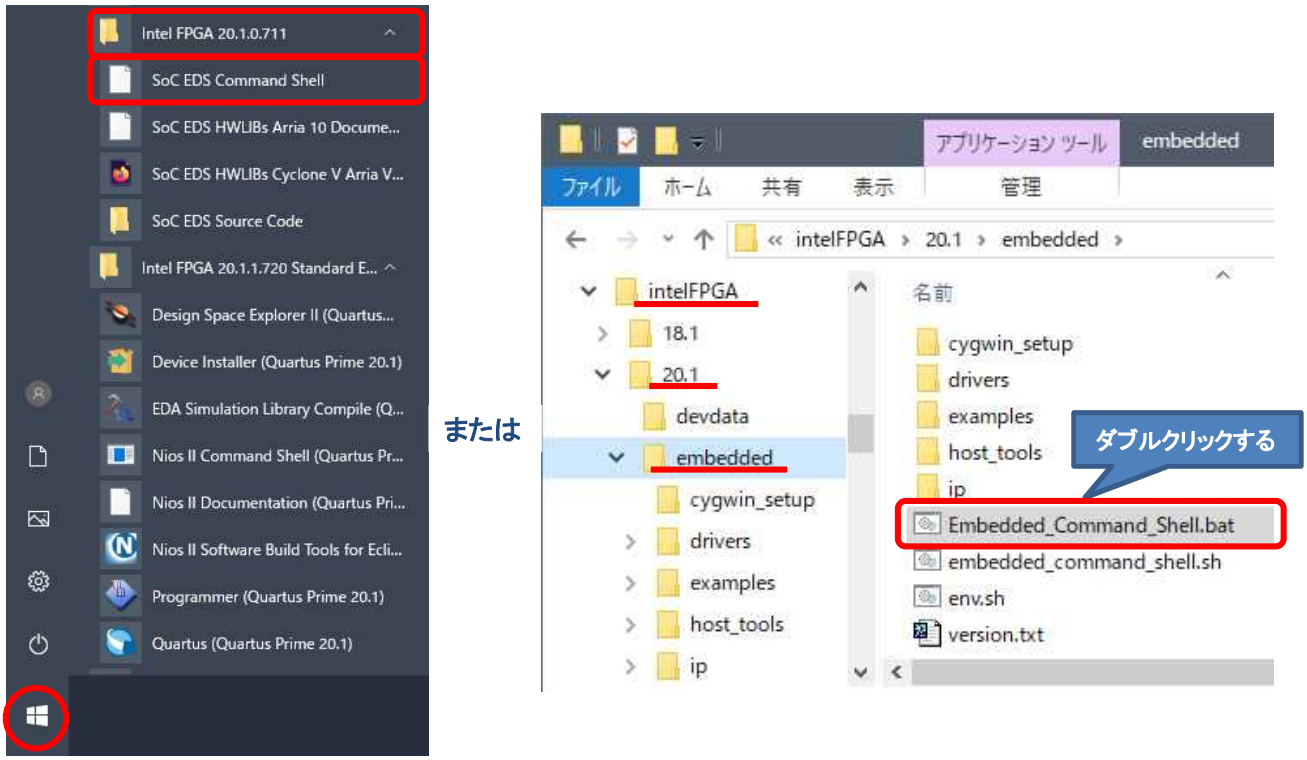


図 4-1. Embedded Command Shell の起動

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 生成に必要となる 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 ↵
```

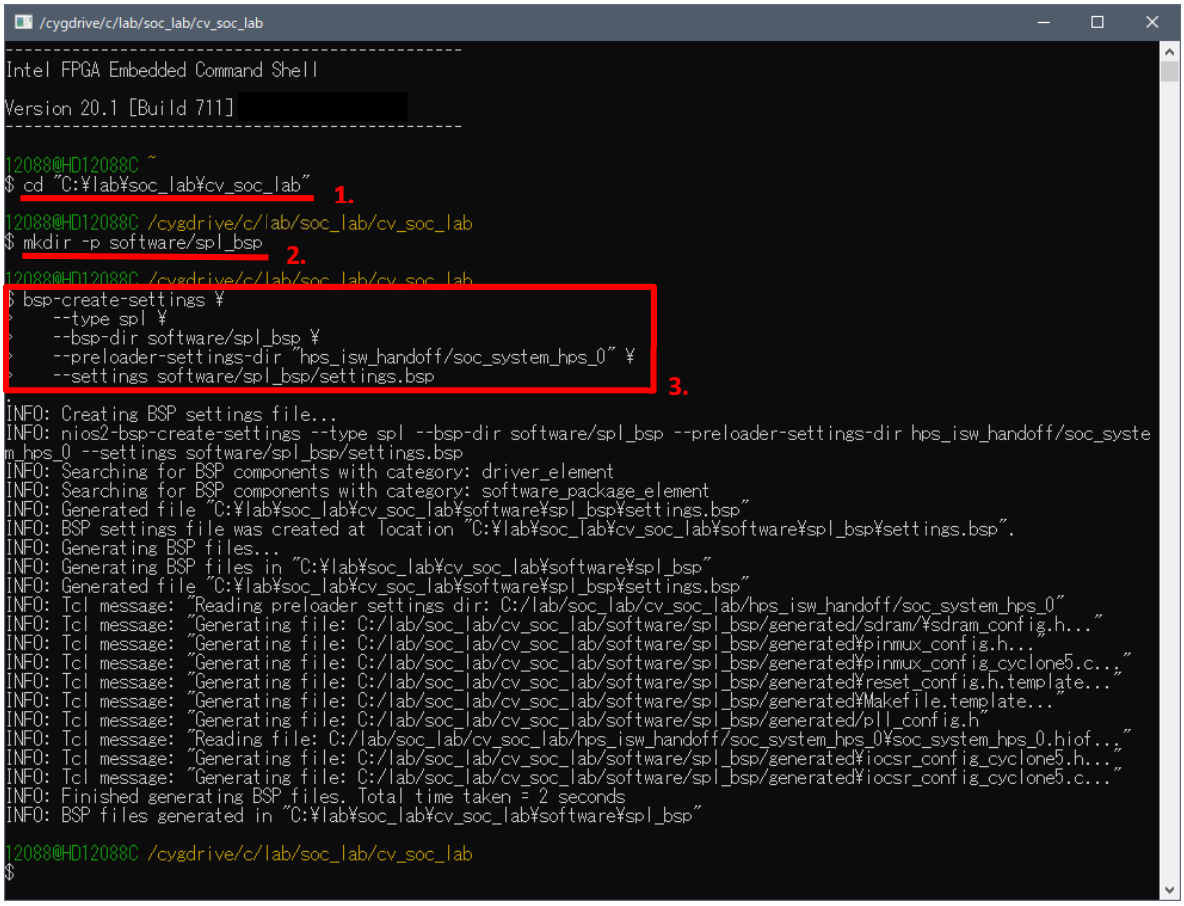


図 4-2. Bsp プロジェクトの生成

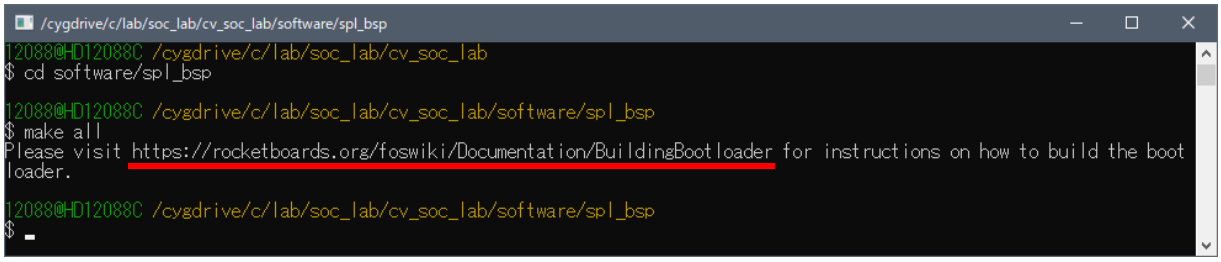
⚠ **注記:**

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。



参照: Building Bootloader for Cyclone V and Arria 10

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 Prelodaer 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

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

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。

> ⚠ **注記:**
> ※ 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。
>
> ※ このドキュメントでは Ubuntu 18.04 LTS を使用した説明となっていますが、Ubuntu のバージョンについては、ご使用の環境に合わせて読み替えてください。

> ⓘ **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。

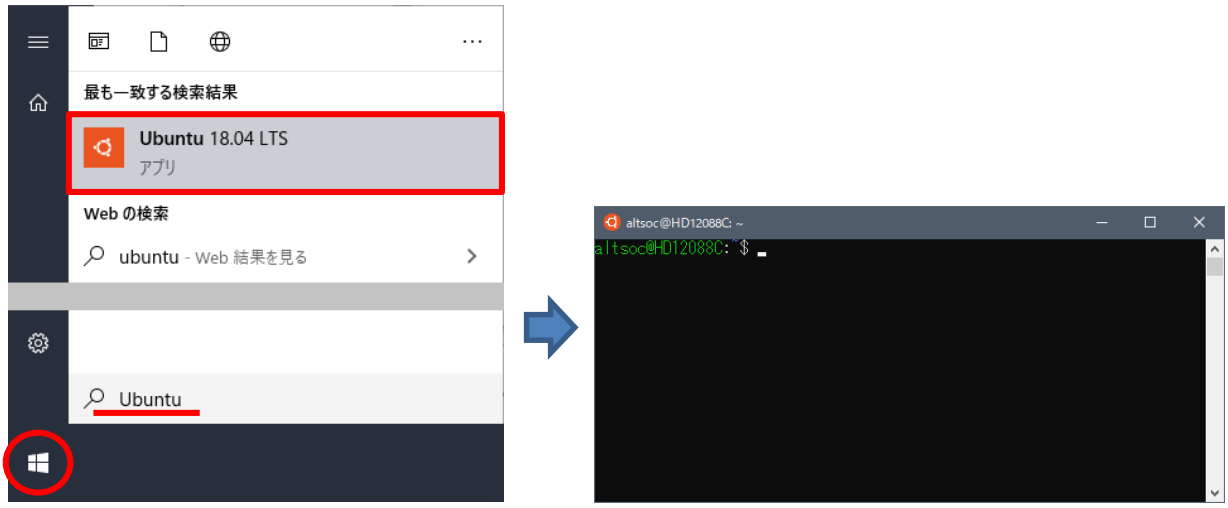


図 4-3. Embedded Command Shell の起動

> ⓘ **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ため時間が掛かります。
>
> **※ 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

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

____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
```

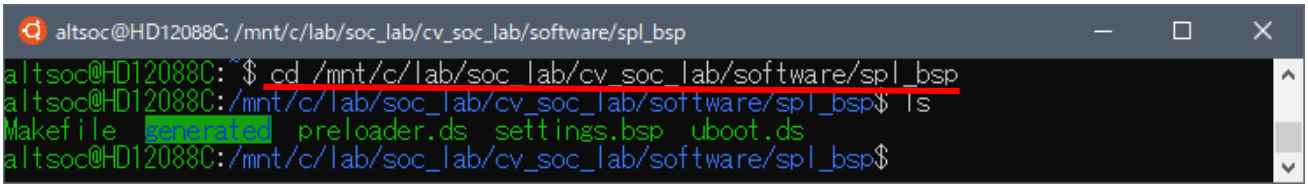


図 4-4. Bsp プロジェクト・ディレクトリーへの移動

____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 -xzvf ACDS20.1STD_REL_GSRD_PR.tar.gz
$ mv u-boot-socfpga-ACDS20.1STD_REL_GSRD_PR/ uboot-socfpga
```

> ⓘ **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 -xzvf /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/Preloder 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
```

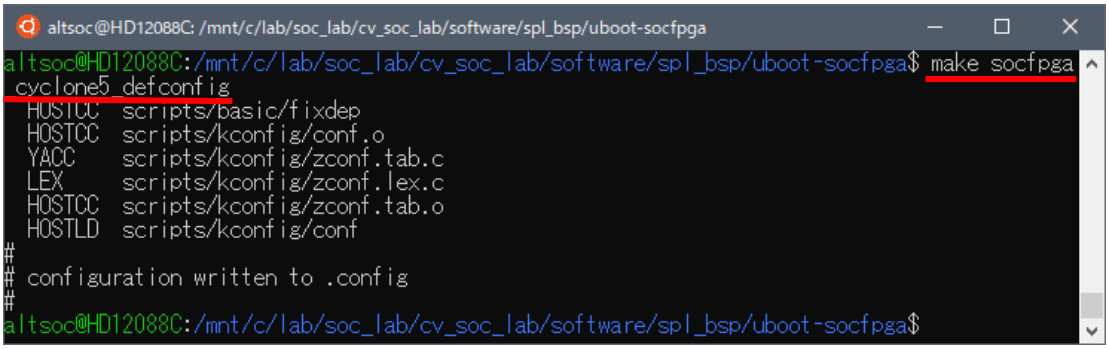


図 4-5. U-Boot/Preloader ソースツリーのコンフィギュレーション

____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
```

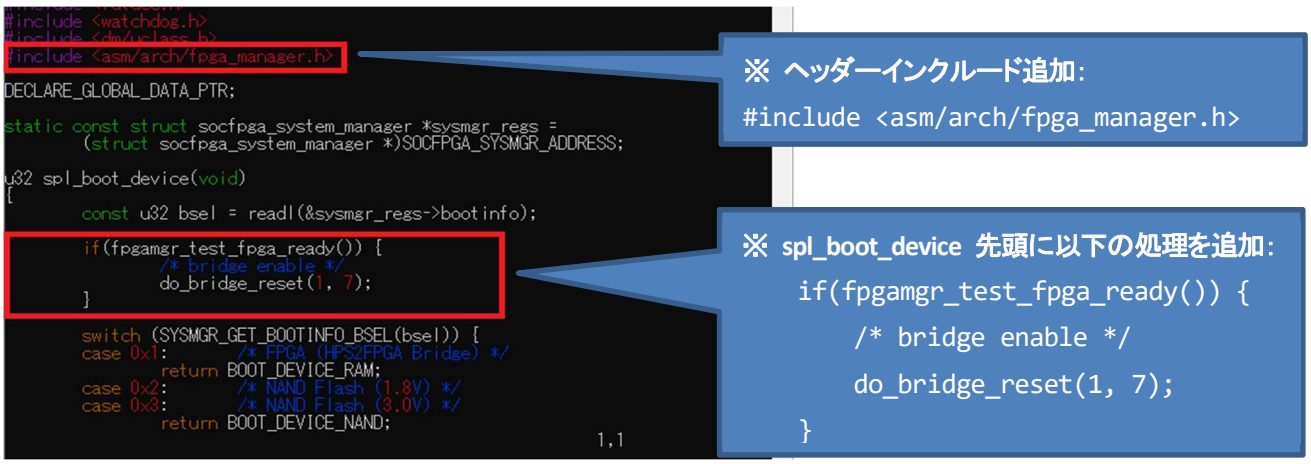


図 4-6. Arch/arm/mach-socfpga/spl_gen5.c の編集

```
$ vi arch/arm/mach-socfpga/misc_gen5.c ↵
```

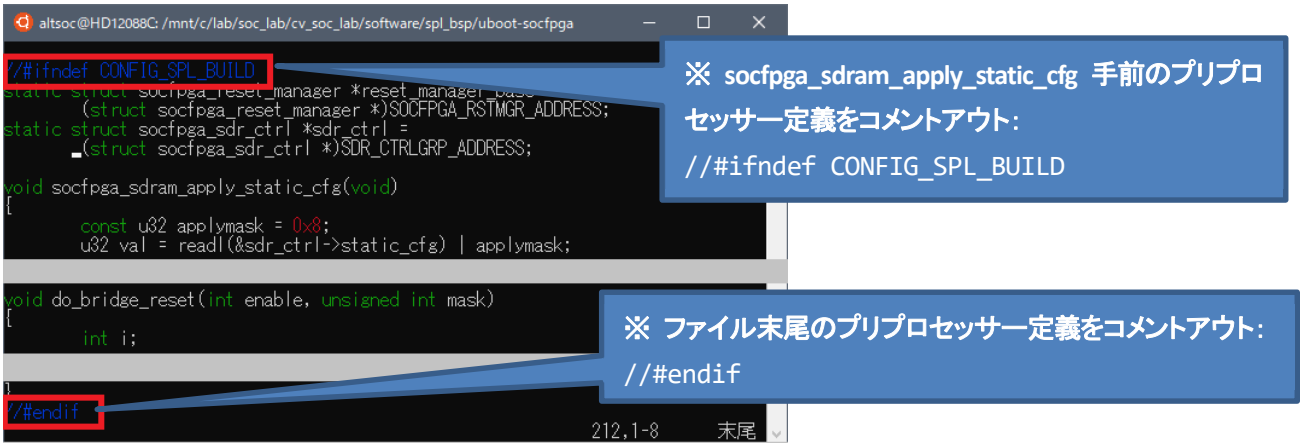


図 4-7. Arch/arm/mach-socfpga/misc_gen5.c の編集

> ⓘ **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 「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。

```
$ make -j 24 ↵
```

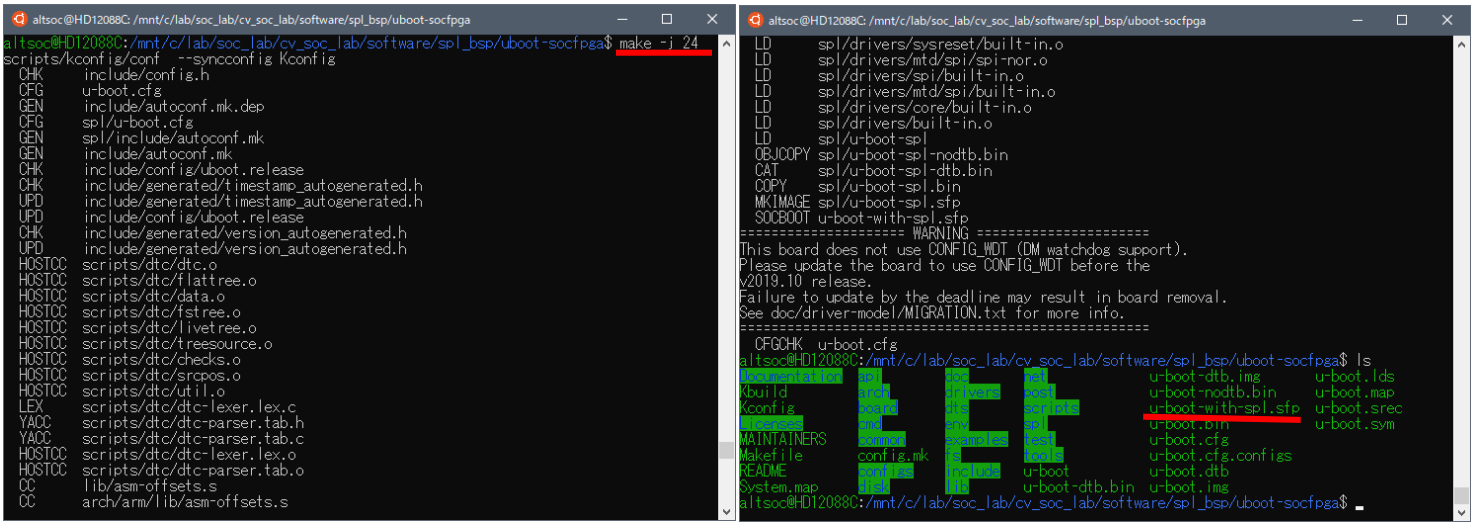


図 4-8. U-boot および Preloader の生成

____ 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 メモリーの設定を必要としません。

  上記の理由から、アルテラ® 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、「3. 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® プロセッサーから 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:**

- このドキュメントでは Arm® DS バージョン 2020.1 を使用した説明になっています。Arm® DS のバージョンについては、ご使用の環境に合わせて読み替えてください。
- 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。
  ライセンスに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。

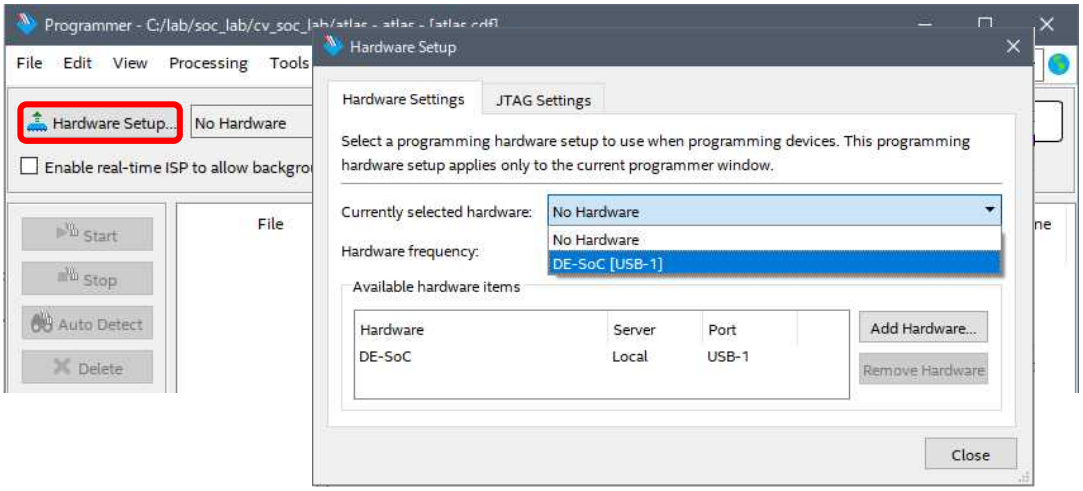


図 5-1. Hardware Setup

____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。

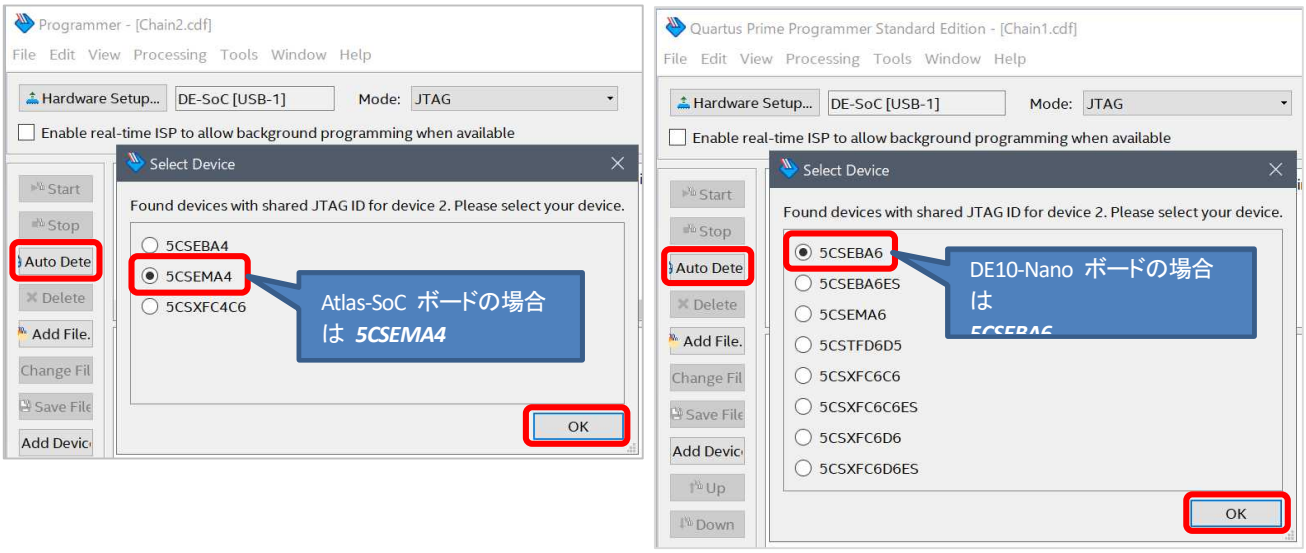


図 5-2. デバイスの選択

____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。

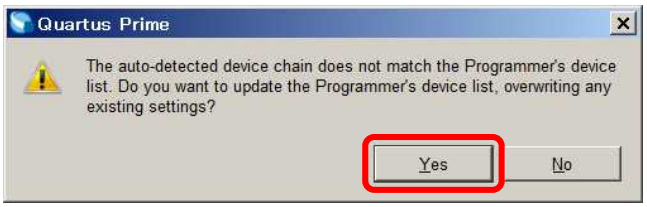


図 5-3. ダイアログボックス

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。

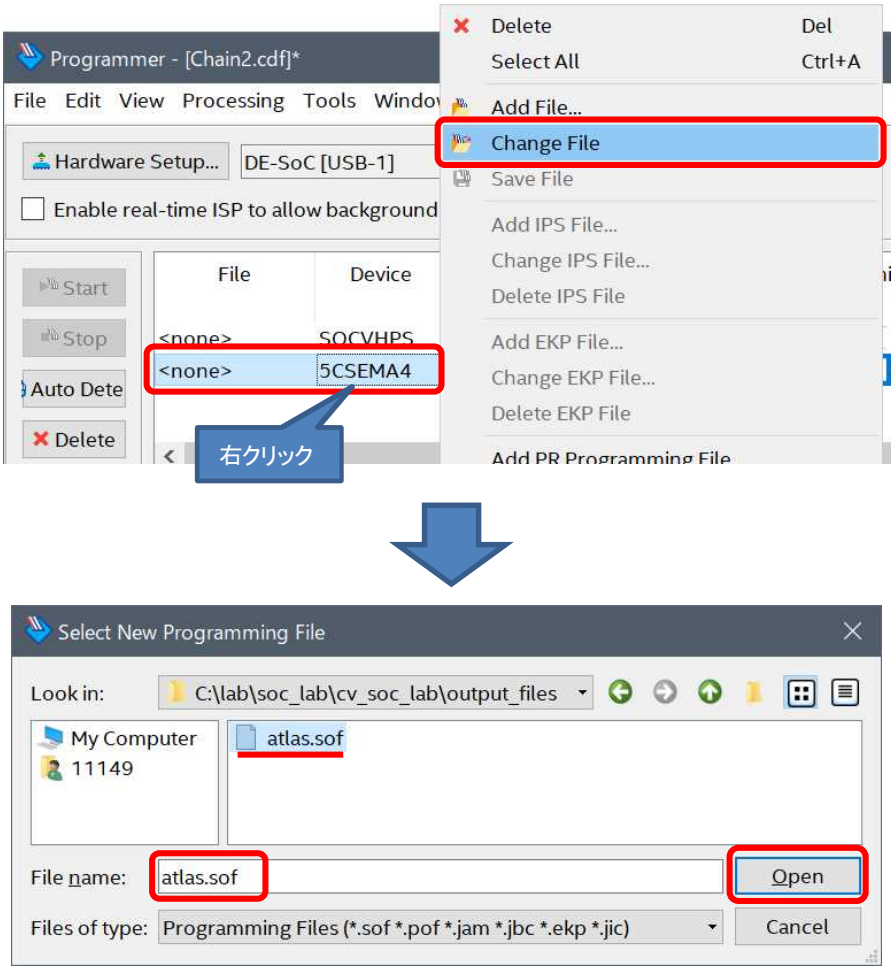


図 5-4. sof ファイルの選択

____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 バーが 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。

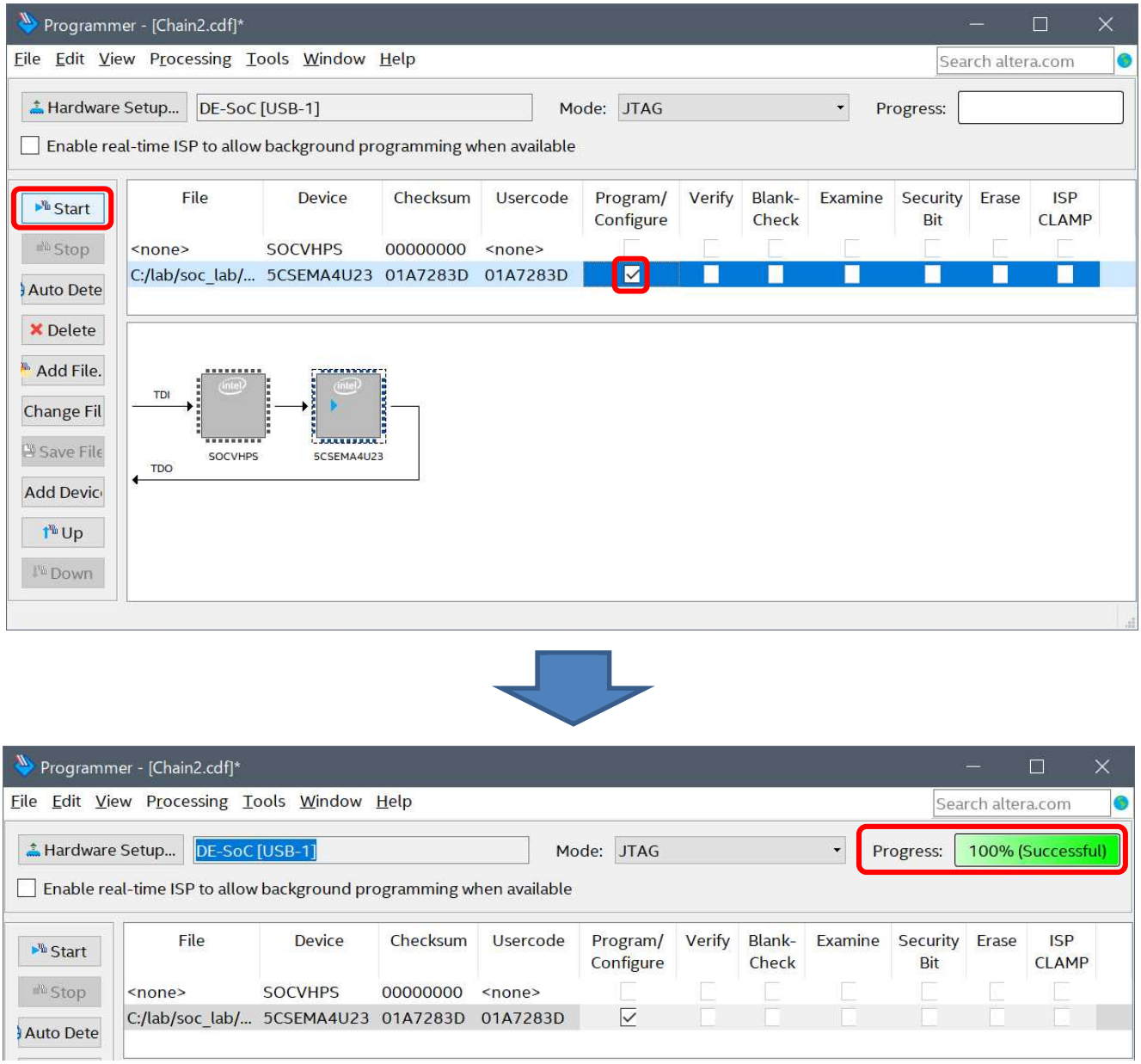


図 5-5. sof のダウンロード

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。

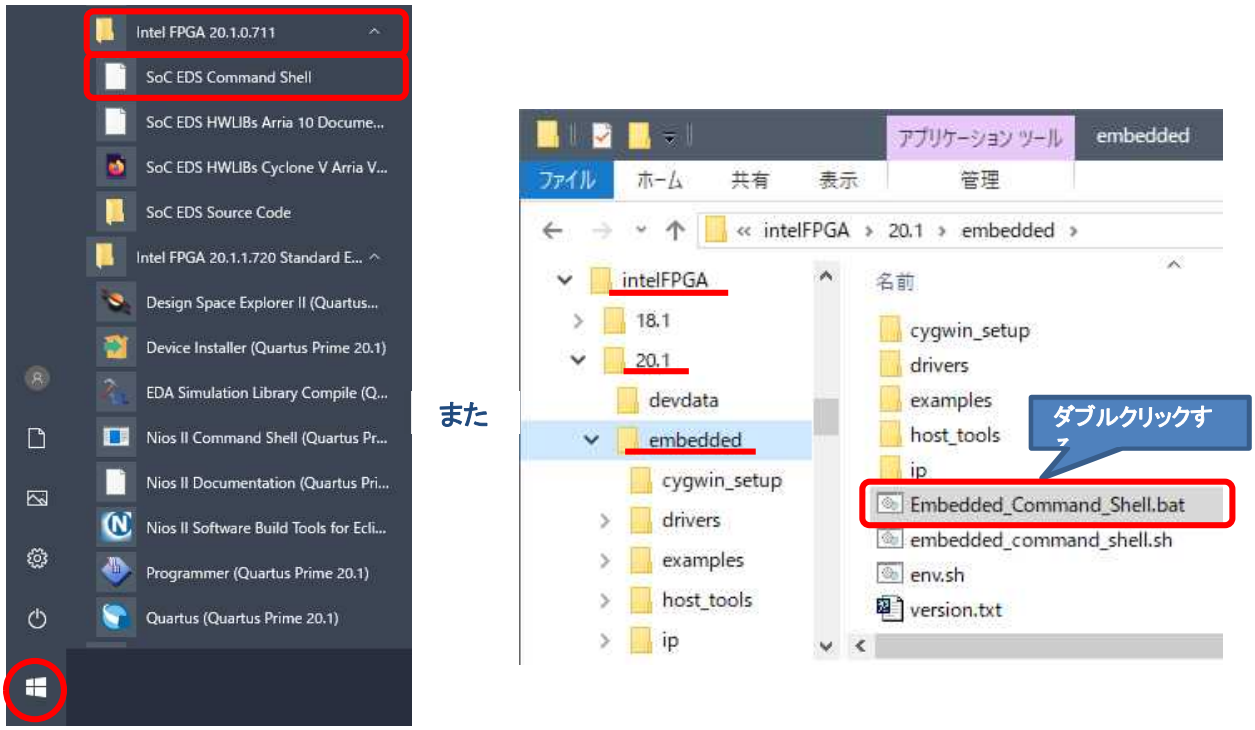


図 5-6. Embedded Command Shell の起動

____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/Program¥ Files/Arm/Development¥ Studio¥ 2020.1/bin/cmdsuite.exe ↵
> bash ↵
$ armds_ide & ↵
```

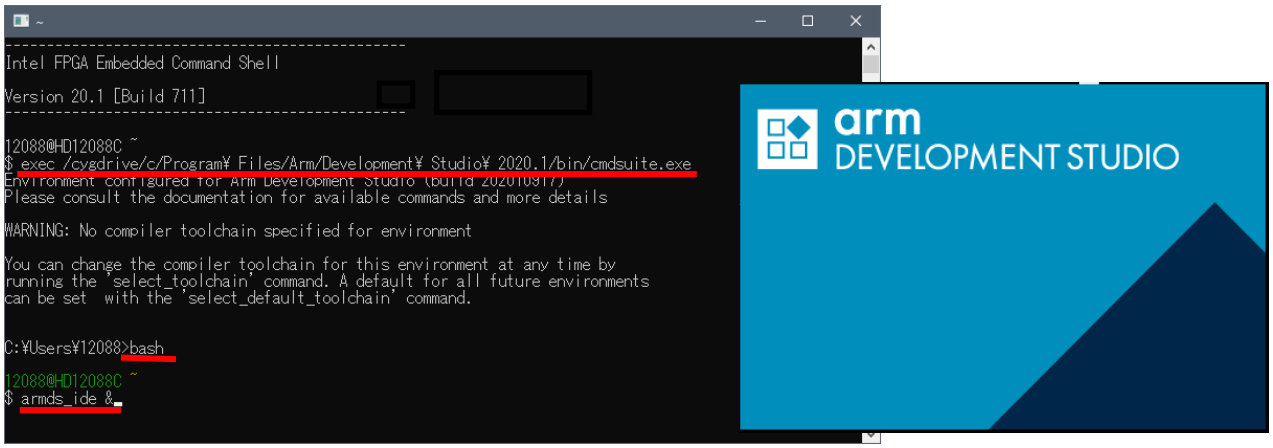


図 5-7. Arm® DS の起動と起動画面

____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
```

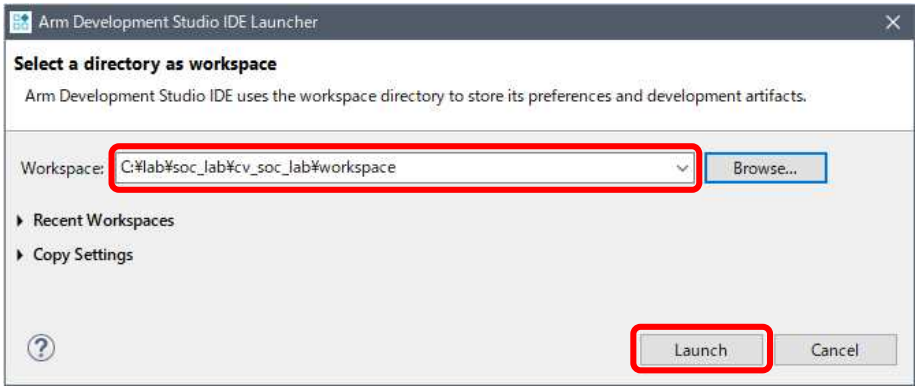


図 5-8. Workspace の作成

____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。

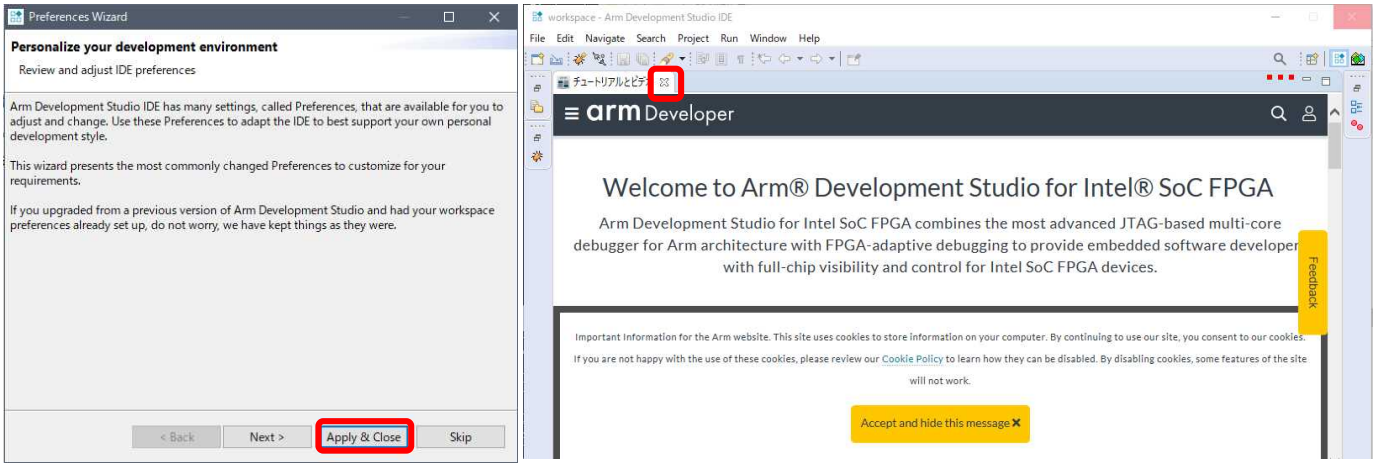


図 5-9. Preference Wizard および Welcome 画面

> ⓘ **Note:**
>
> 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た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。

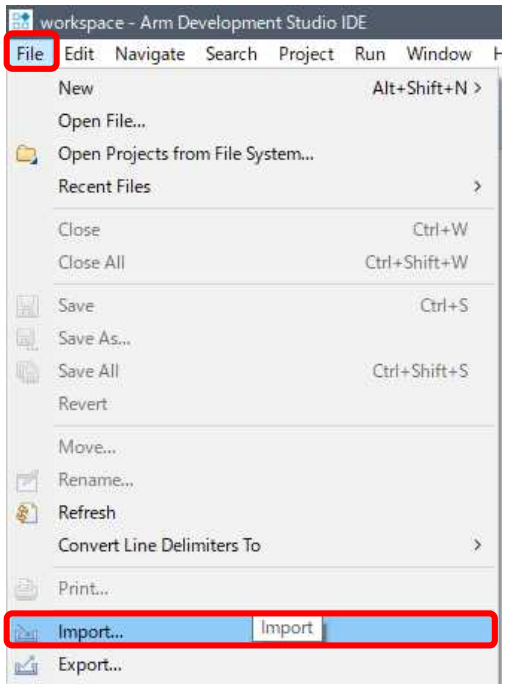


図 5-10.「インポート」メニュー

____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。

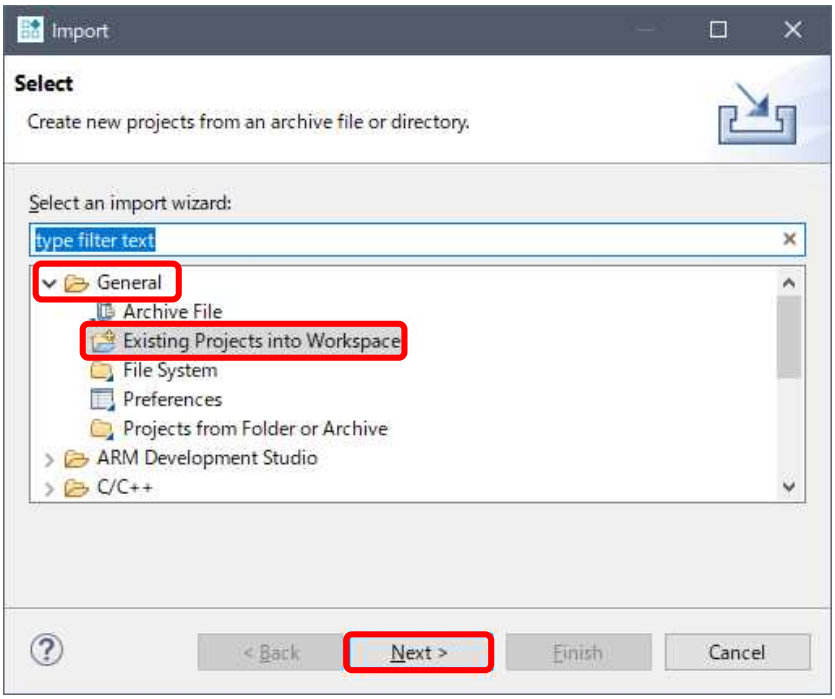


図 5-11. 既存プロジェクトのインポート

____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。

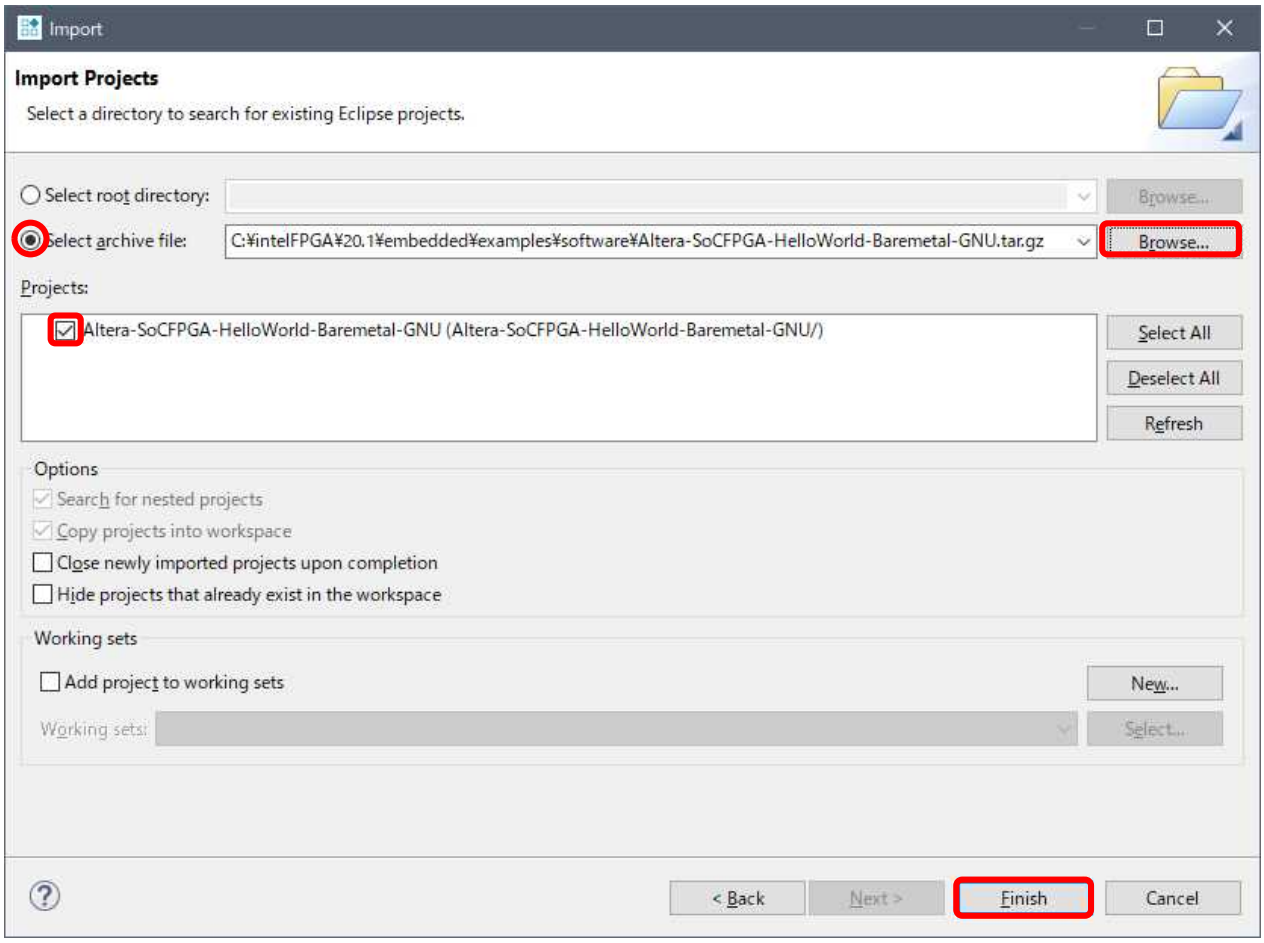


図 5-12. Hello World サンプル・アプリケーションの選択

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。

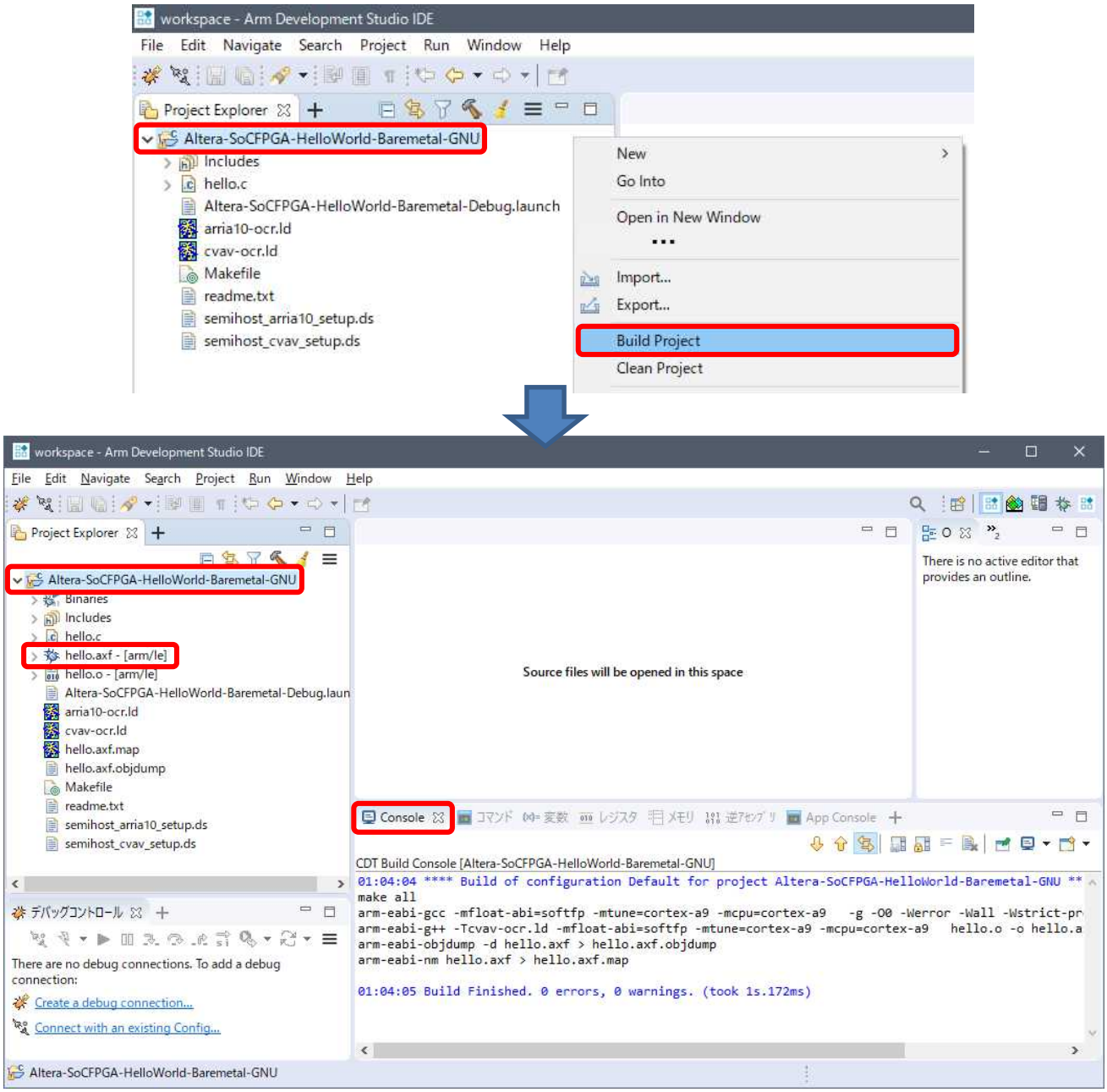


図 5-13. Hello World プロジェクトのビルド

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。

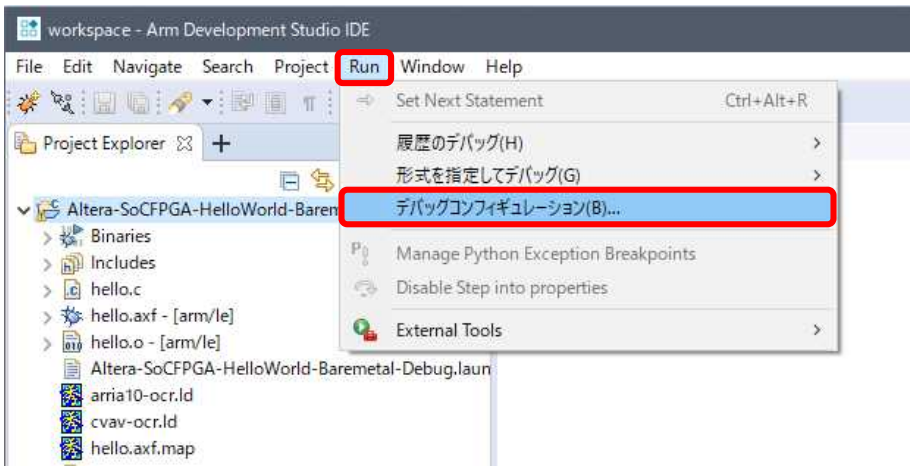


図 5-14. デバッグコンフィギュレーションの選択

____ 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。

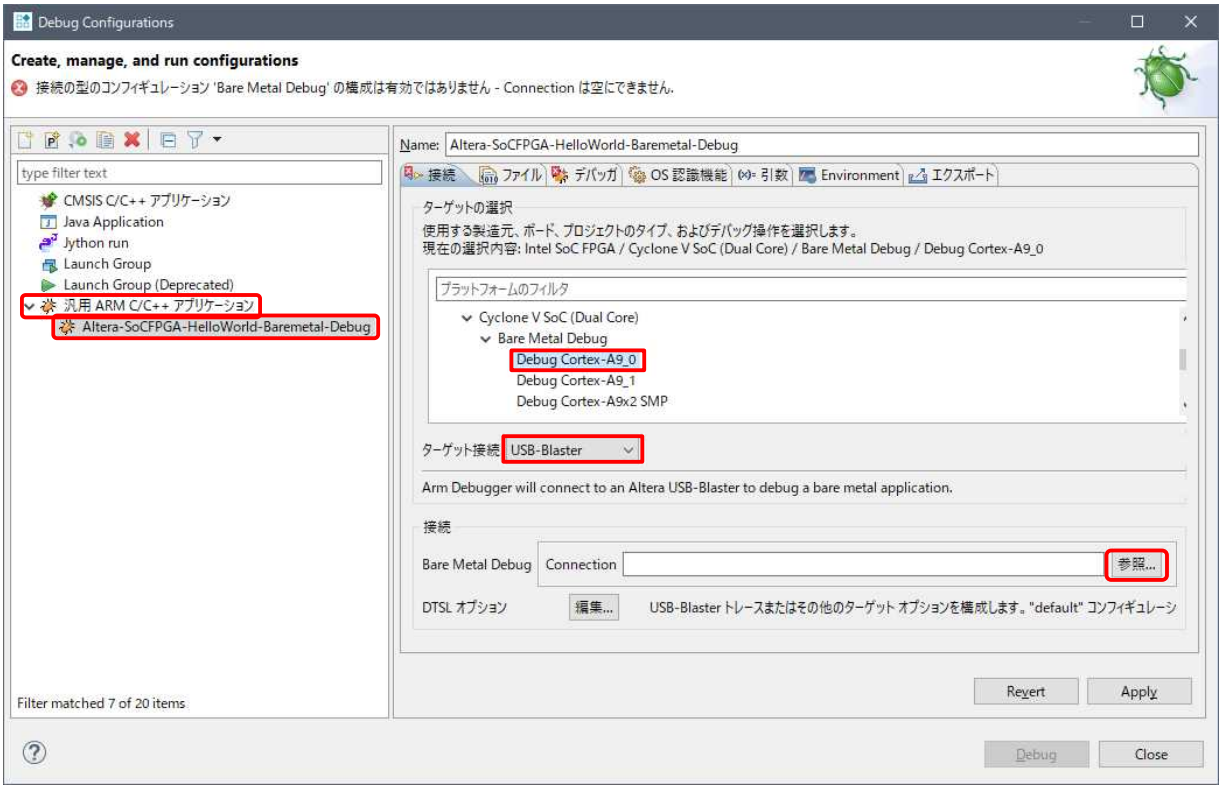


図 5-15. デバッグコンフィギュレーション（接続タブ）

____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ガスクリプト（.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。

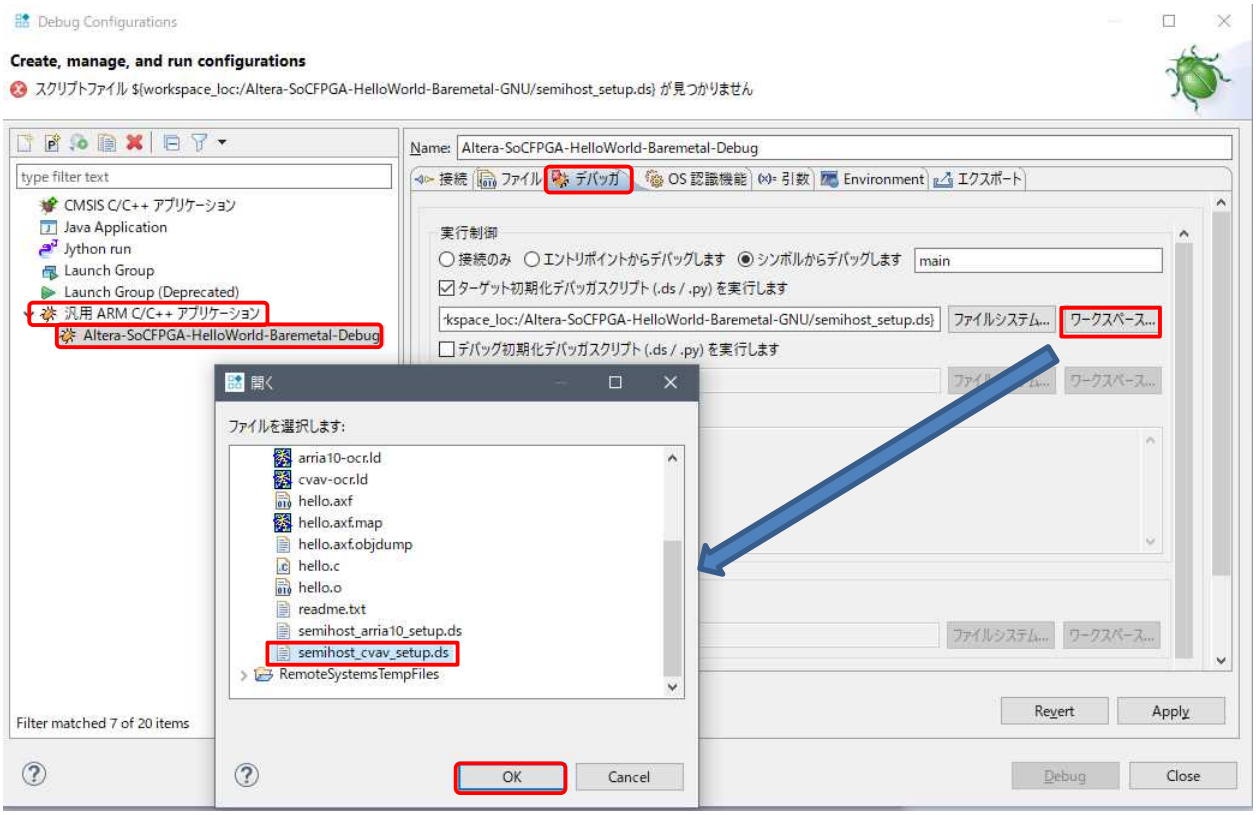


図 5-17. デバッグコンフィギュレーション（デバッガタブ）

____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。

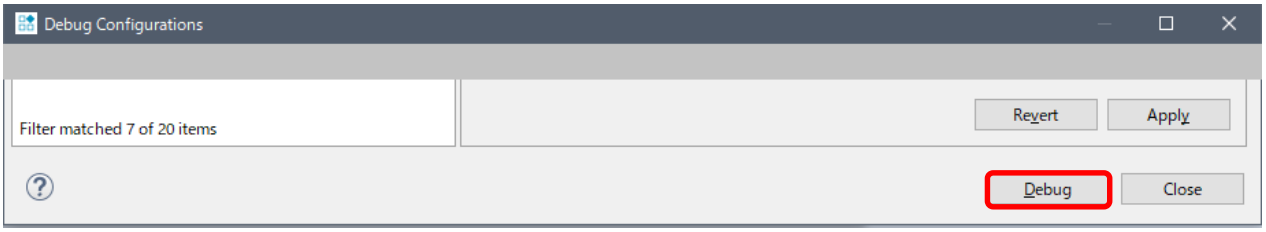


図 5-18. デバッグの実行

____ 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。

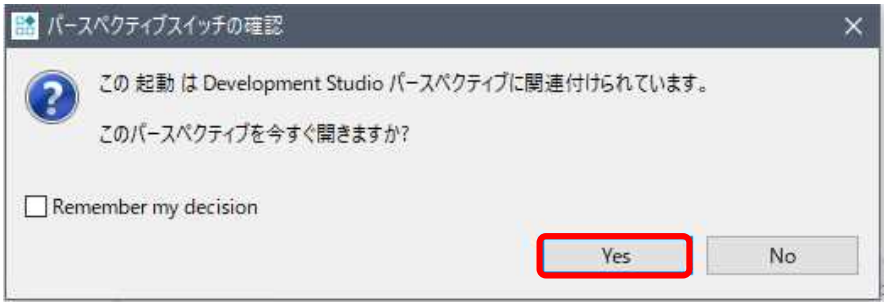


図 5-19. パースペクティブスイッチの確認

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。

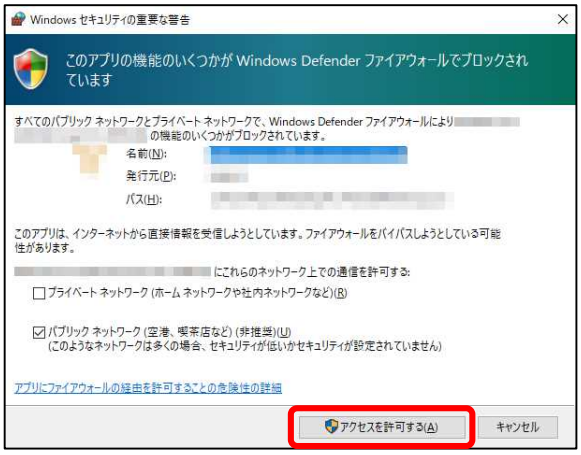


図 5-20. セキュリティの警告

> ⓘ **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ーが 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

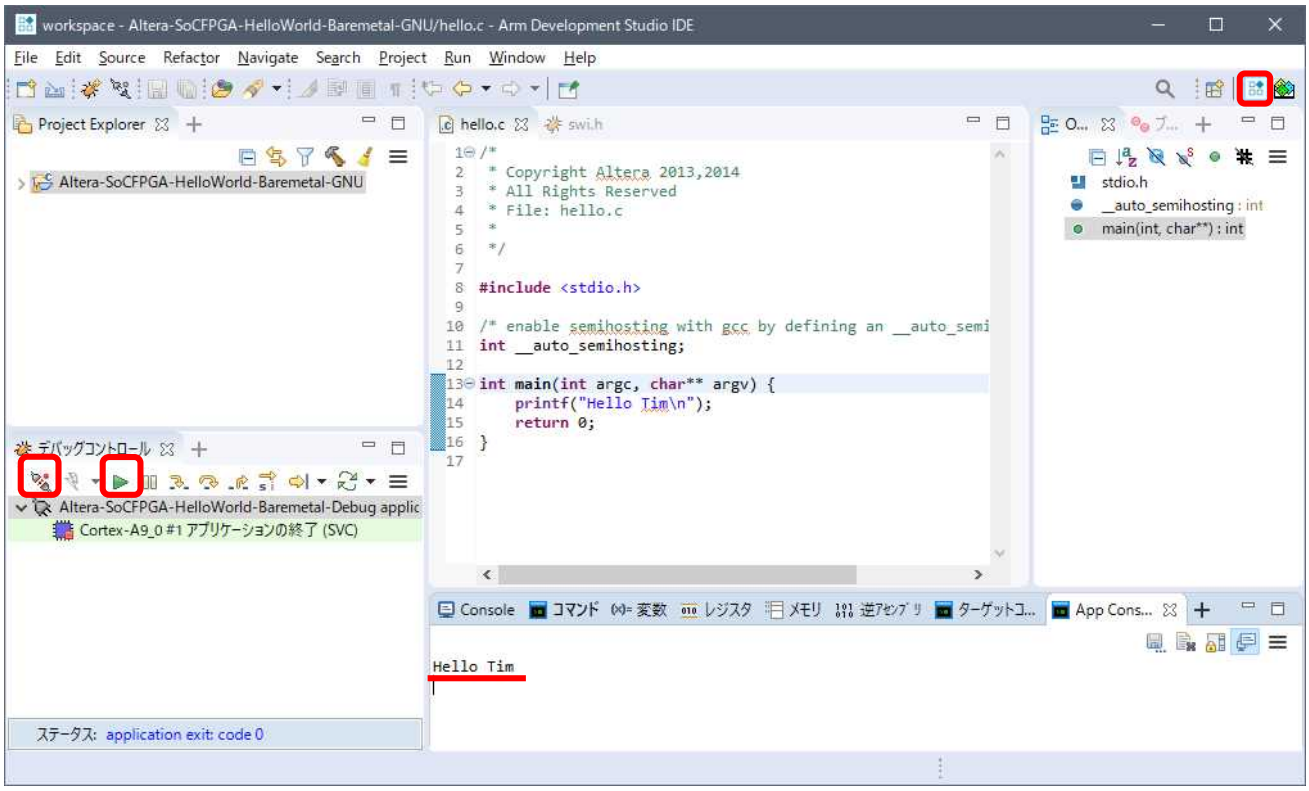
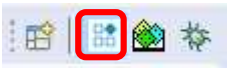
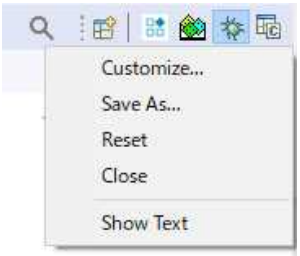


図 5-21. Hello Tim の表示

____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。

> ⓘ **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。

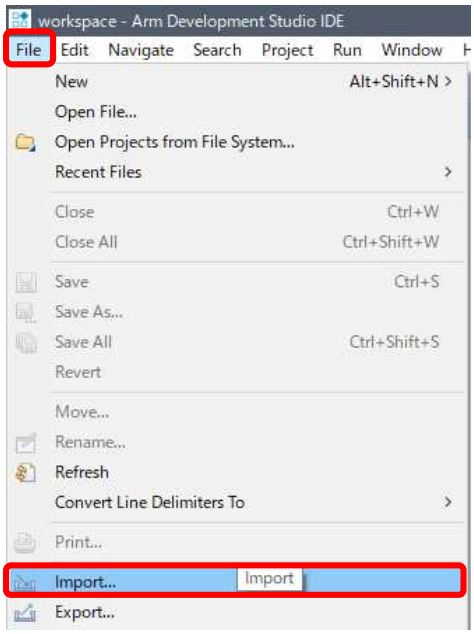


図 5-22.「インポート」メニュー

____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。

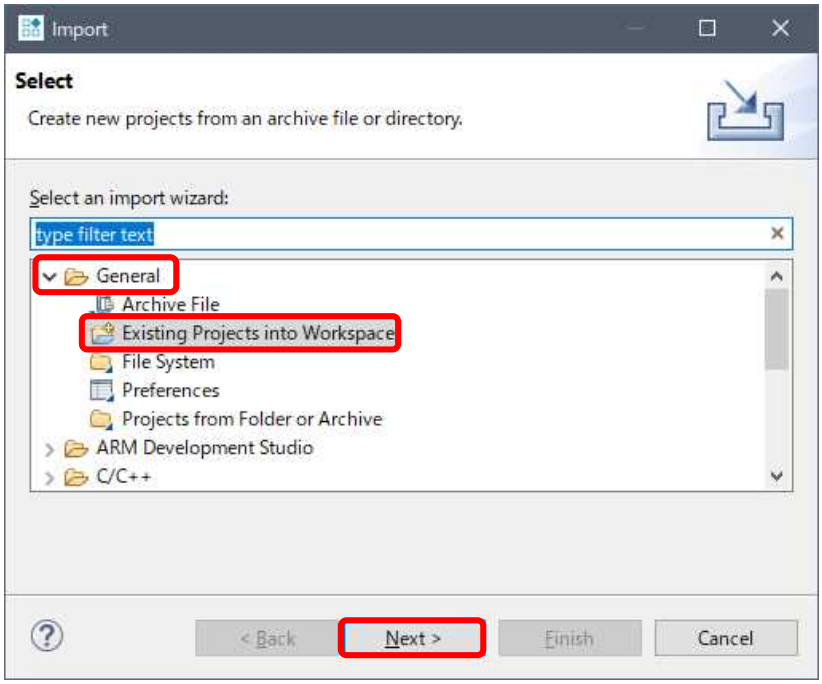


図 5-23. 既存プロジェクトのインポート

____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。

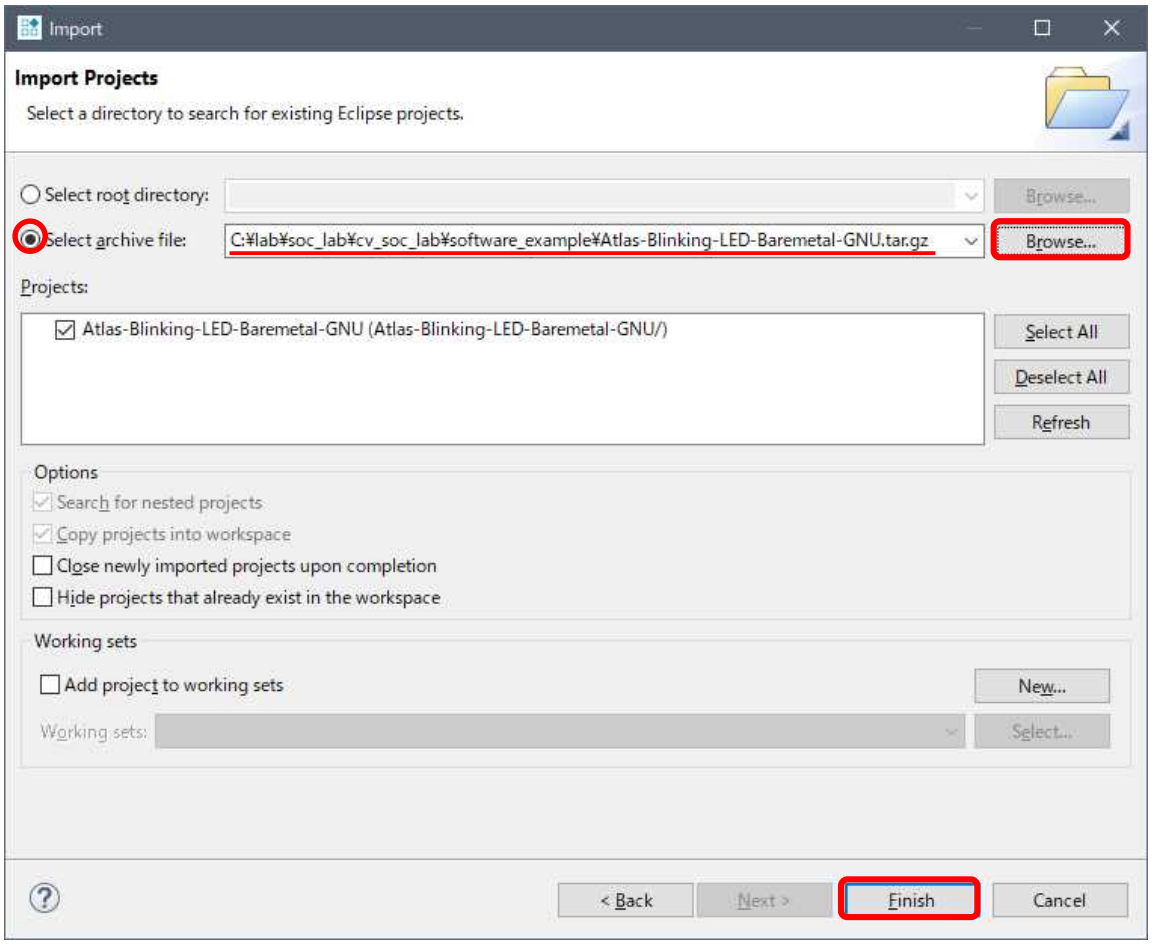


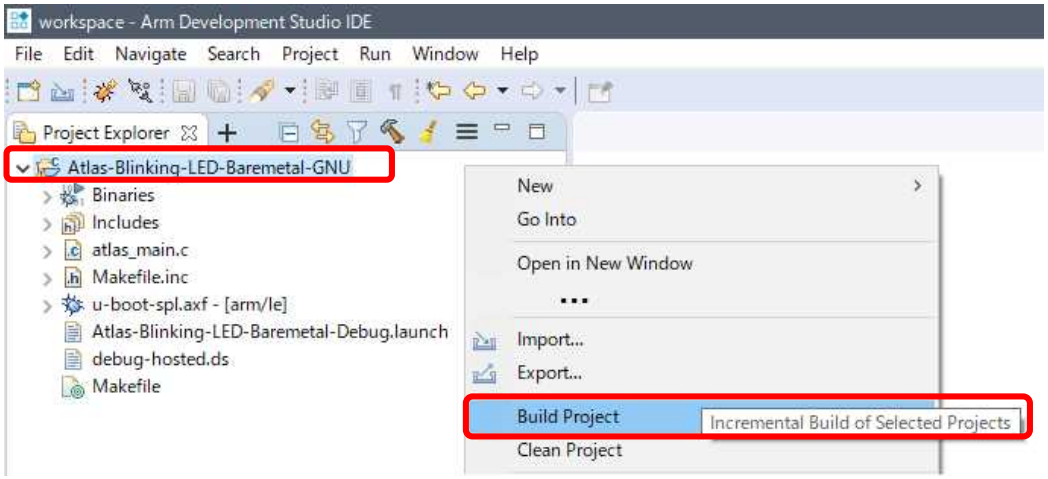
図 5-24. LED Blink サンプル・アプリケーションの選択

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。



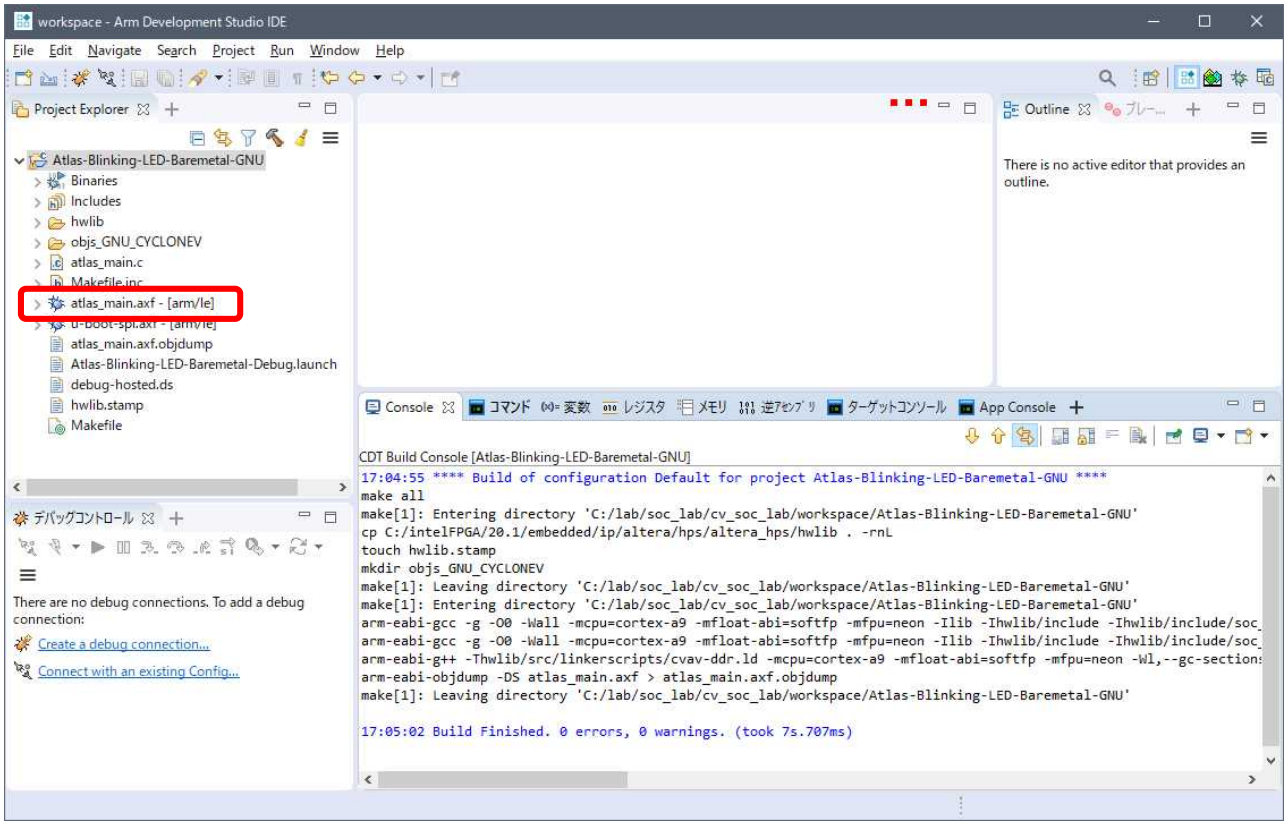


図 5-25. LED Blink サンプル・アプリケーションのビルド

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。

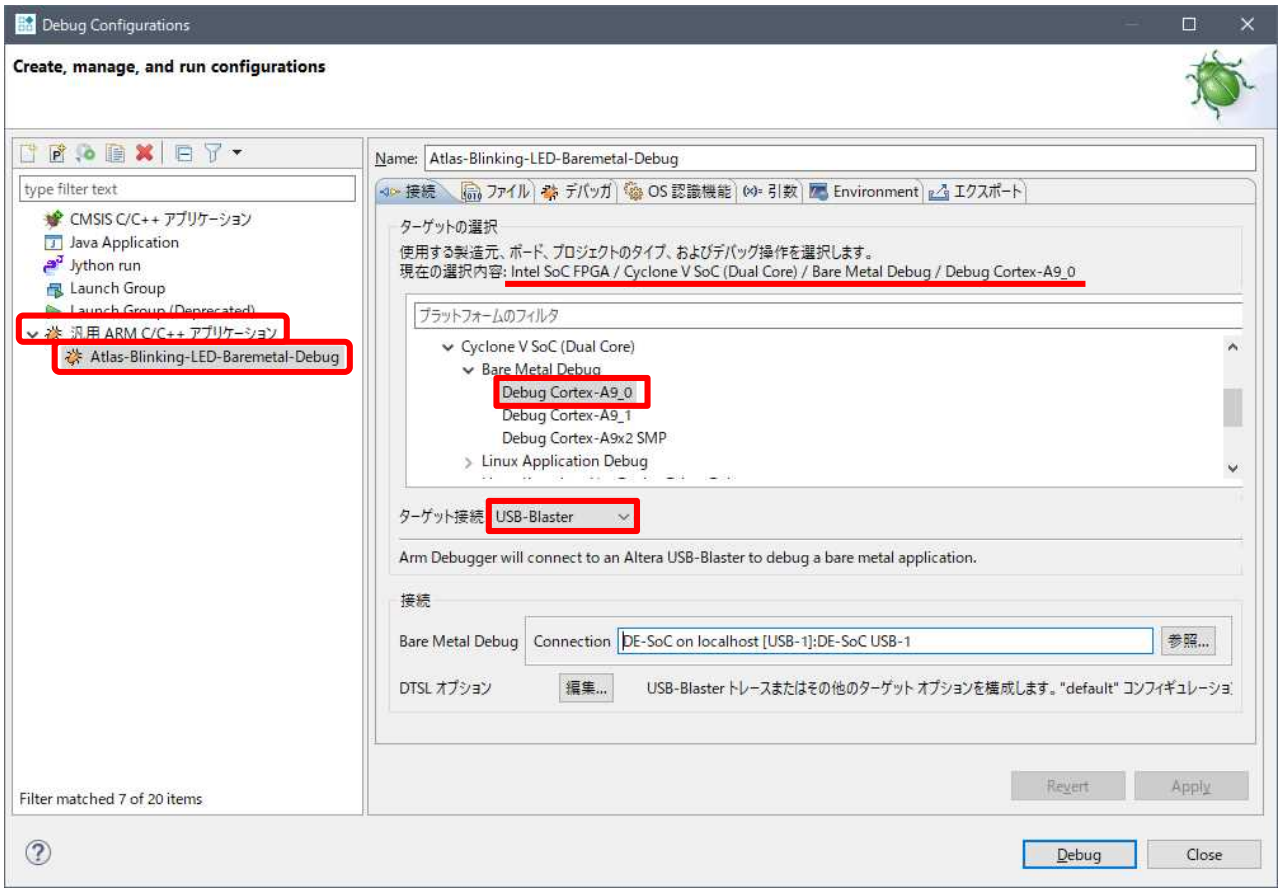


図 5-26. LED Blink サンプル・アプリケーションのデバッグ構成

____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。

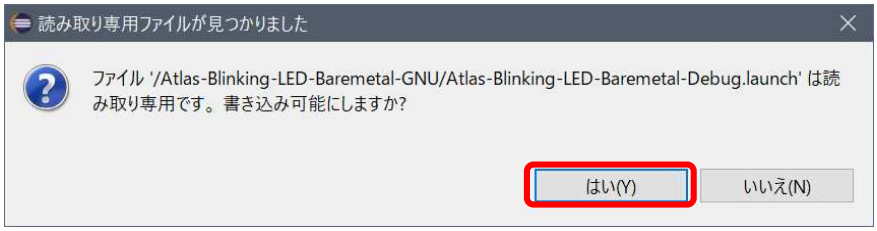


図 5-27. 確認ポップアップ

____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。

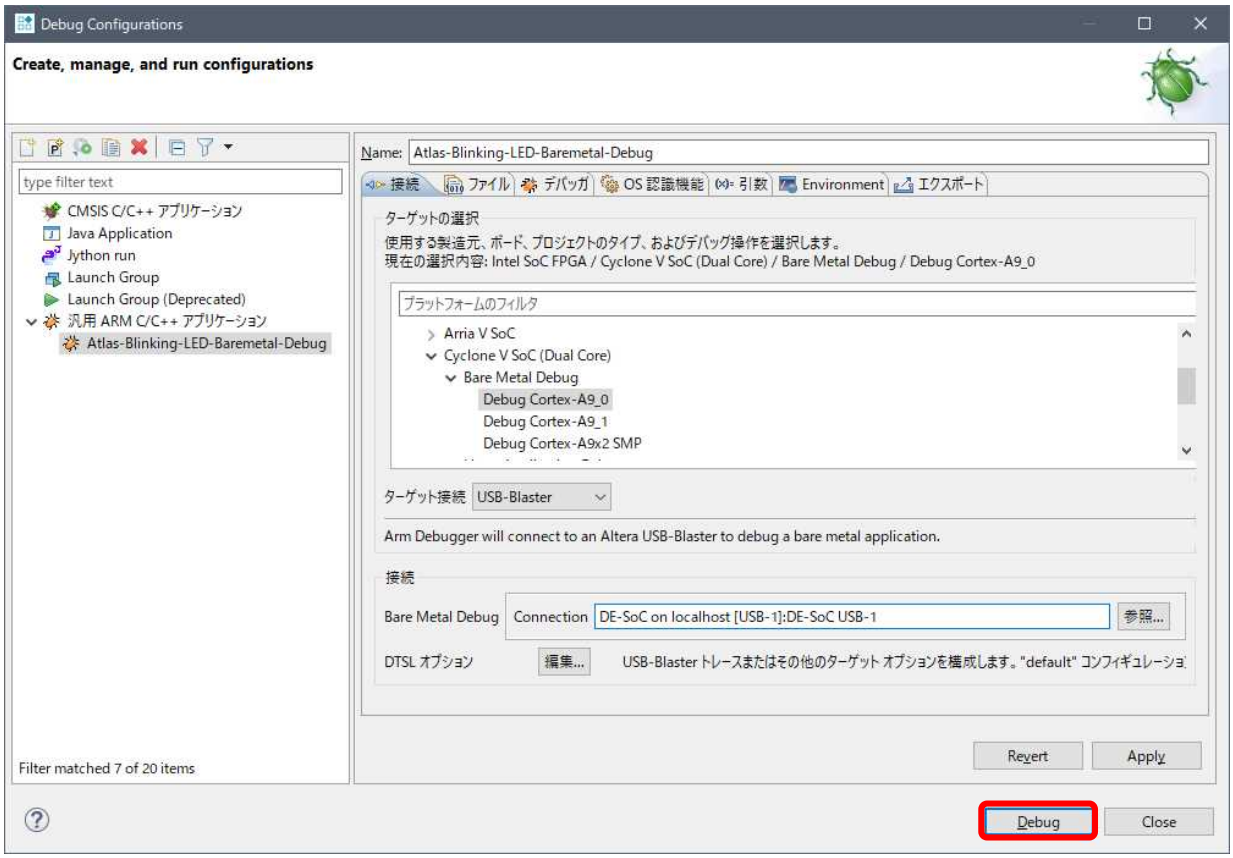


図 5-29. LED Blink サンプル・アプリケーションのデバッグ

____ 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。

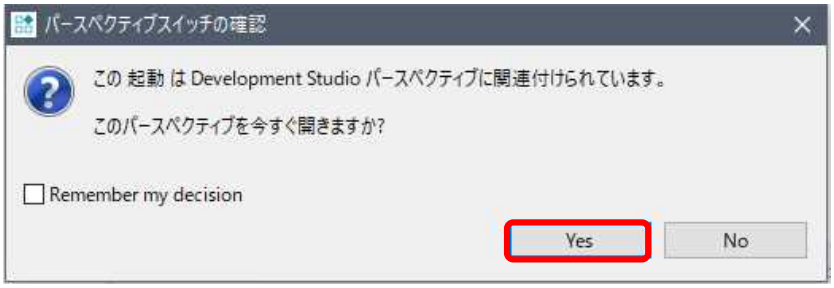


図 5-30. パースペクティブスイッチの確認

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。

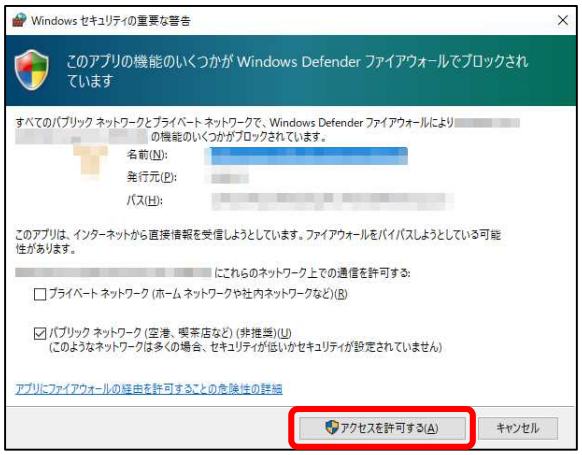


図 5-31. セキュリティの警告

> ⓘ **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。

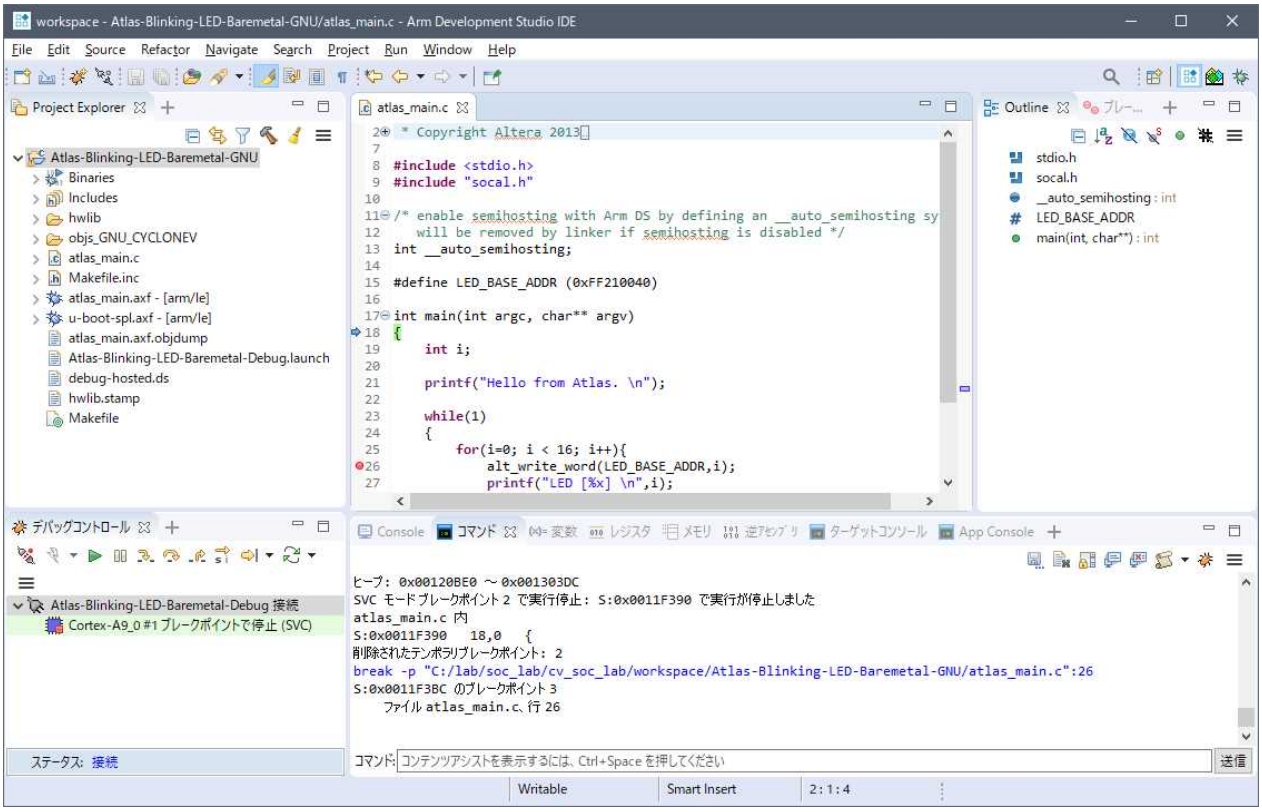


図 5-32. ブレークポイントの設定

____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、「4. 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。

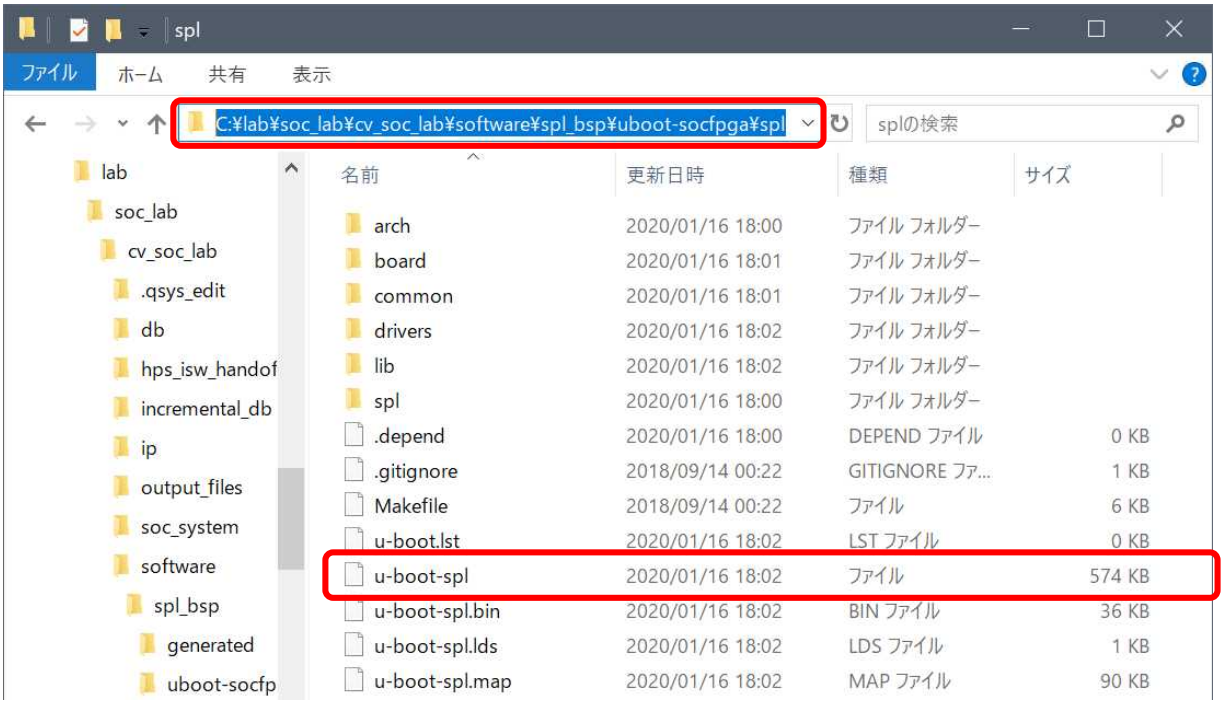


図 5-33. U-boot-spl ファイルの確認

> ⓘ **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” 等にリネームします。

____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。

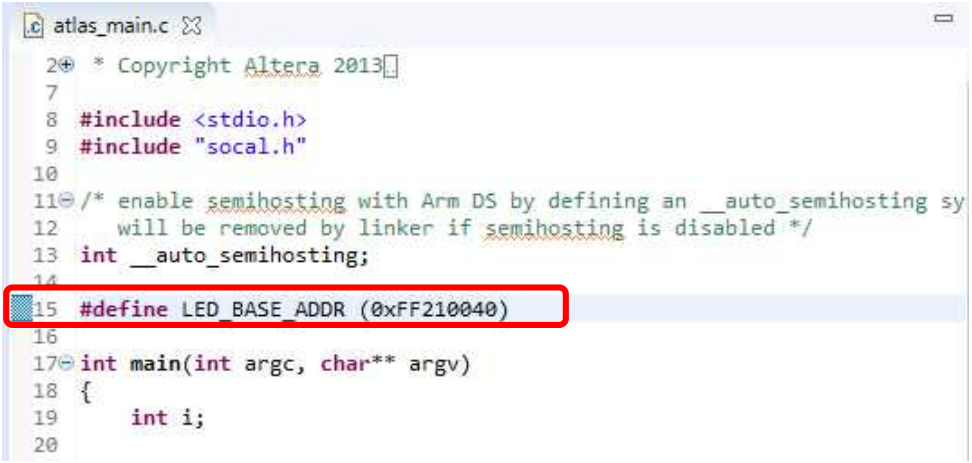


図 5-34. 今までのアドレス指定方法

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”
```

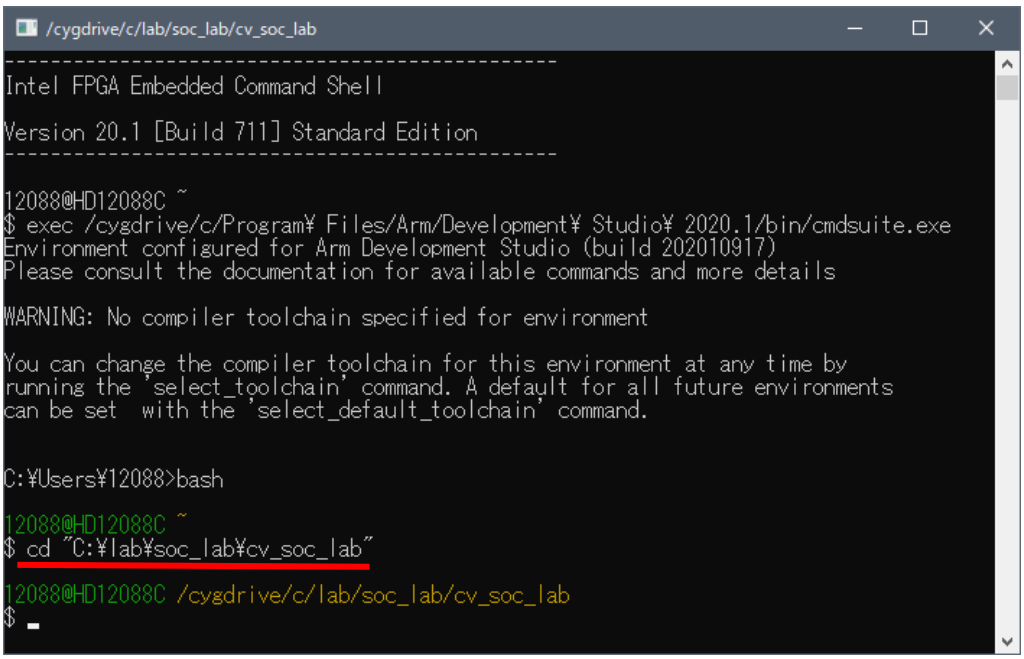


図 5-35. ディレクトリーの移動

____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。

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

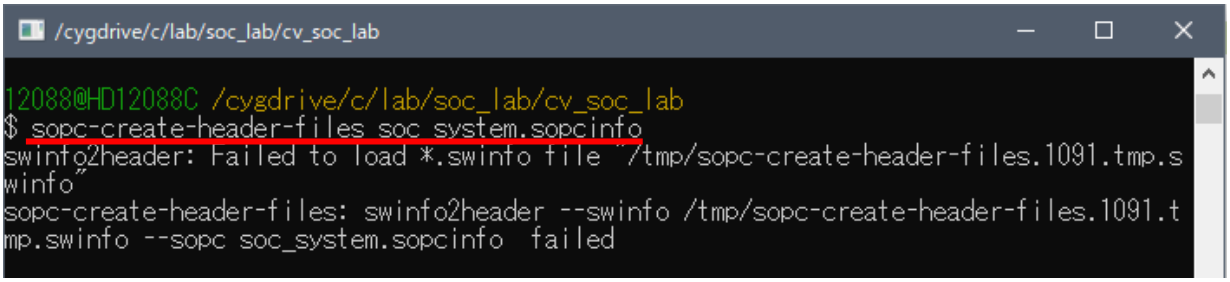


図 5-36. システム・ヘッダーファイル生成コマンドの実行（エラー）

> ⚠ **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 sopc-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 sopc-create-header-files が実行エラーになるトラブルの回避策」

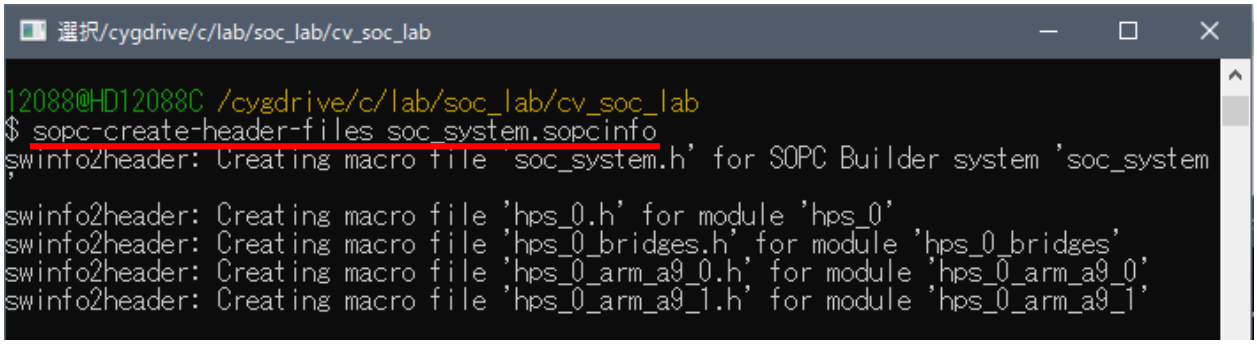


図 5-37. システム・ヘッダーファイル生成コマンドの実行（成功）

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。

atlas_main.c

```
 2  * Copyright Altera 2013
 7
 8 #include <stdio.h>
 9 #include "socal.h"
10 #include "hps_0_arm_a9_0.h"
11
12 /* enable semihosting with Arm DS by defining a
13    will be removed by linker if semihosting is
14 int __auto_semihosting;
15
16 /* #define LED_BASE_ADDR (0xFF210040) */
17 #define LED_BASE_ADDR (LED_PIO_BASE)
18
19 int main(int argc, char** argv)
20 {
21     int i;
22
23     printf("Hello from Atlas. \n");
24
25     while(1)
26     {
27         for(i=0; i < 16; i++){
28             alt_write_word(LED_BASE_ADDR,i);
29             printf("LED [%x] \n",i);
30         }
31     }
```

*atlas_main.c　hps_0_arm_a9_0.h

```
72 /*
73  * Macros for device 'led_pio', class 'altera_avalon_pio'
74  * The macros are prefixed with 'LED_PIO_'.
75  * The prefix is the slave descriptor.
76  */
77 #define LED_PIO_COMPONENT_TYPE altera_avalon_pio
78 #define LED_PIO_COMPONENT_NAME led_pio
79 #define LED_PIO_BASE 0xff210040
80 #define LED_PIO_SPAN 32
81 #define LED_PIO_END 0xff21005f
82 #define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
83 #define LED_PIO_BIT_MODIFYING_OUTPUT_REGISTER 1
84 #define LED_PIO_CAPTURE 0
85 #define LED_PIO_DATA_WIDTH 8
86 #define LED_PIO_DO_TEST_BENCH_WIRING 0
87 #define LED_PIO_DRIVEN_SIM_VALUE 0
88 #define LED_PIO_EDGE_TYPE NONE
89 #define LED_PIO_FREQ 50000000
90 #define LED_PIO_HAS_IN 1
91 #define LED_PIO_HAS_OUT 1
92 #define LED_PIO_HAS_TRI 0
93 #define LED_PIO_IRQ_TYPE NONE
94 #define LED_PIO_RESET_VALUE 0
95
```

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。

> ⓘ **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。

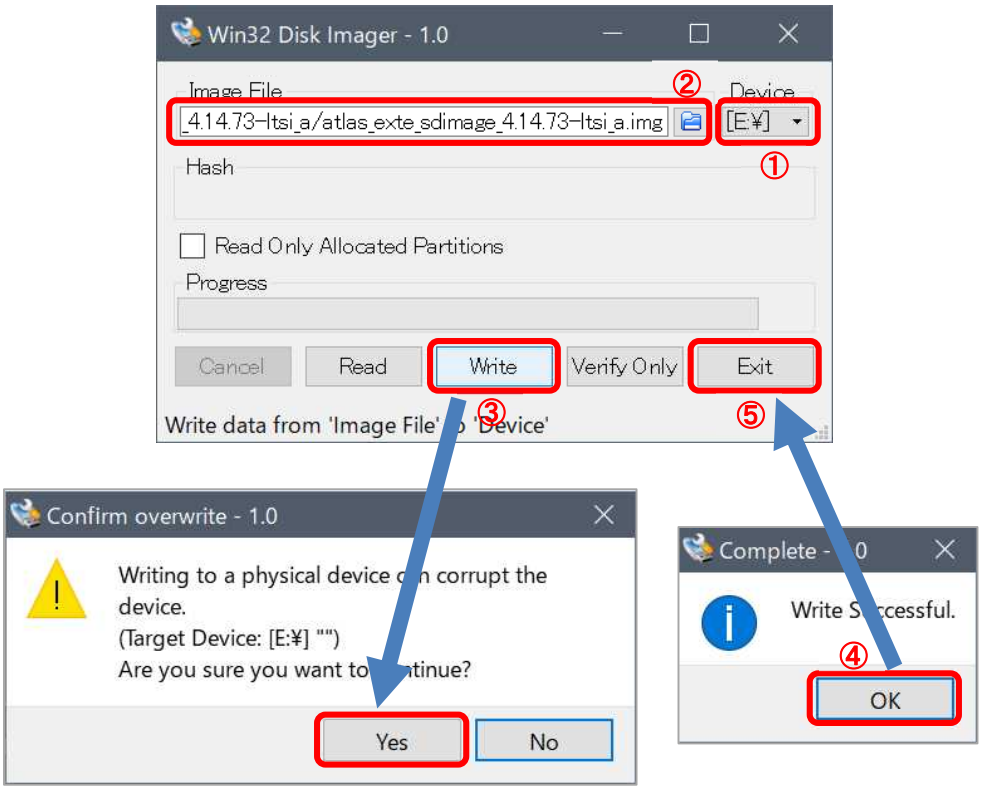


図 6-2. Win32DiskImager

____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。

下図は Atlas-SoC ボードの例ですが、DE10-Nano ボードも基本的には同じです。

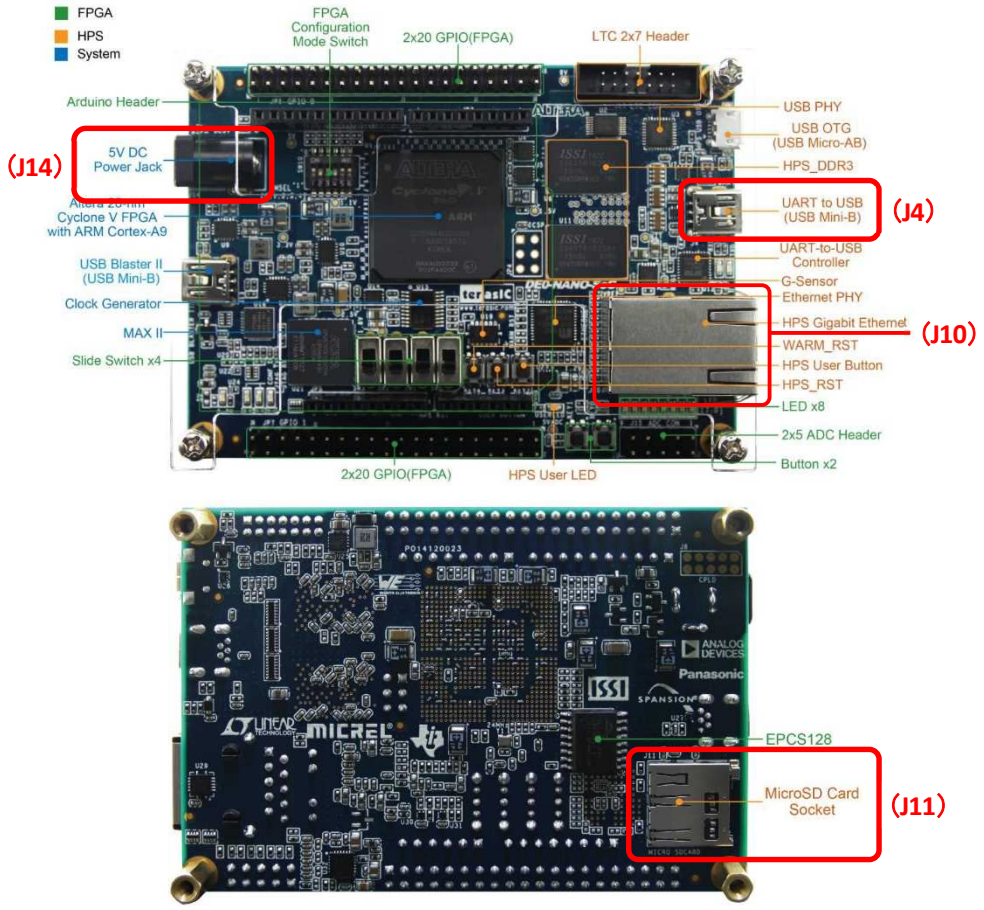


図 6-3. 本演習で使用するインターフェイス（Atlas-SoC ボード）

____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、イーサーネット・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 のイーサーネット・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。

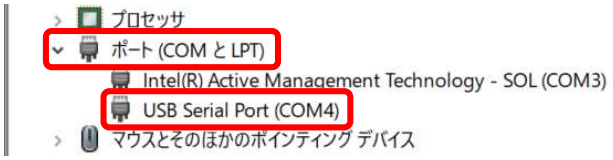


図 6-4. COM ポートの確認

____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）。

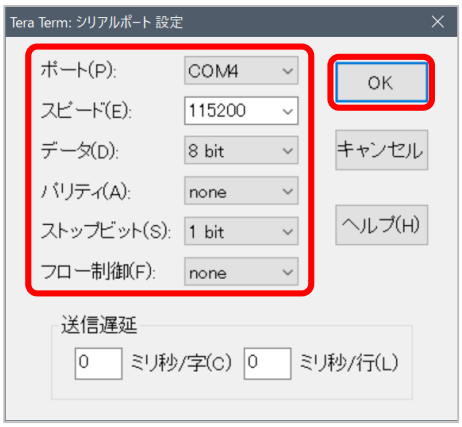


図 6-5. シリアルポートの設定

____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。

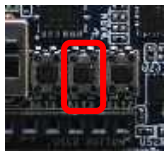


図 6-6. WARM リセットボタン（KEY3）

____9. Linux カーネルが起動したら、 **root** ↵ でログインします。

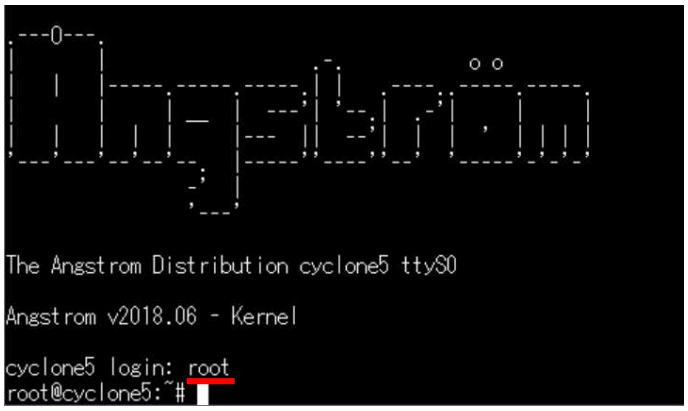


図 6-7. Root でログイン

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。

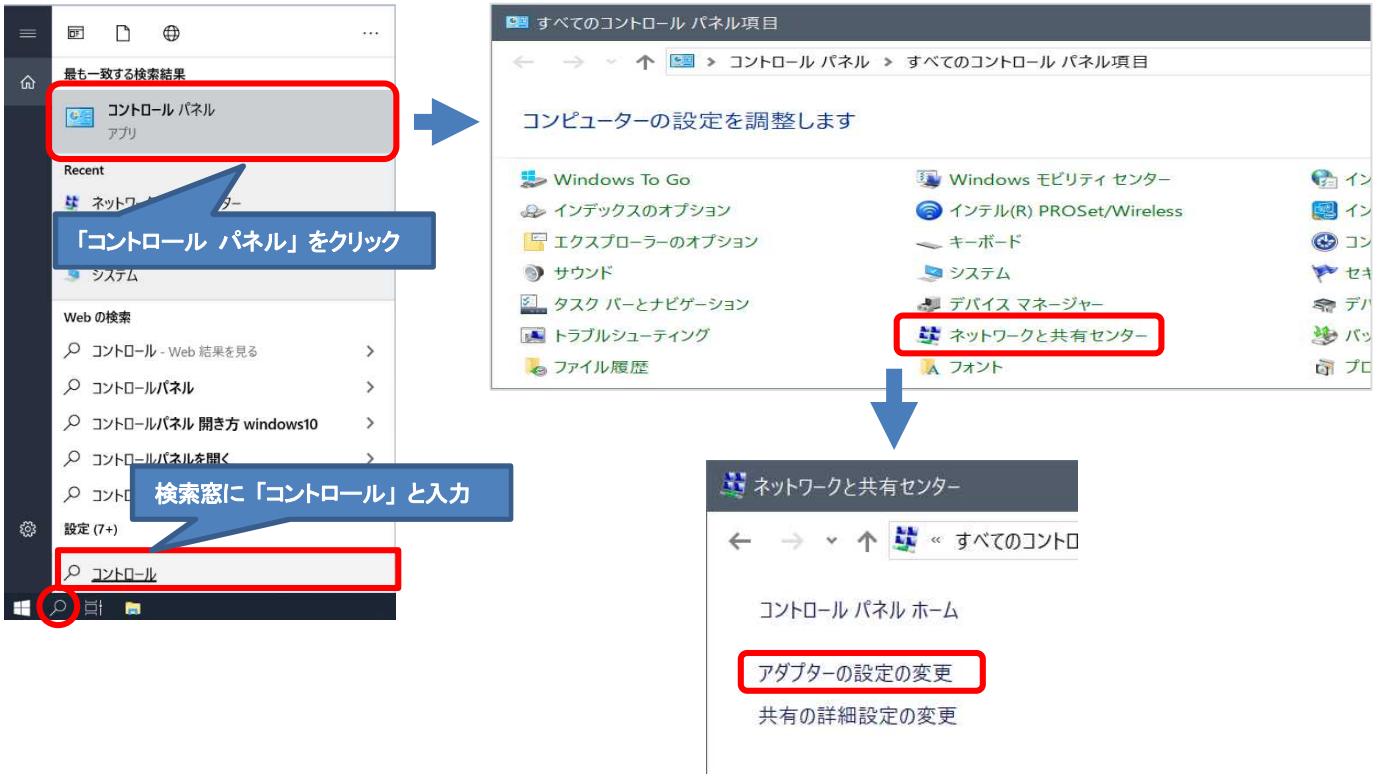


図 6-9. アダプターの設定の変更

____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。

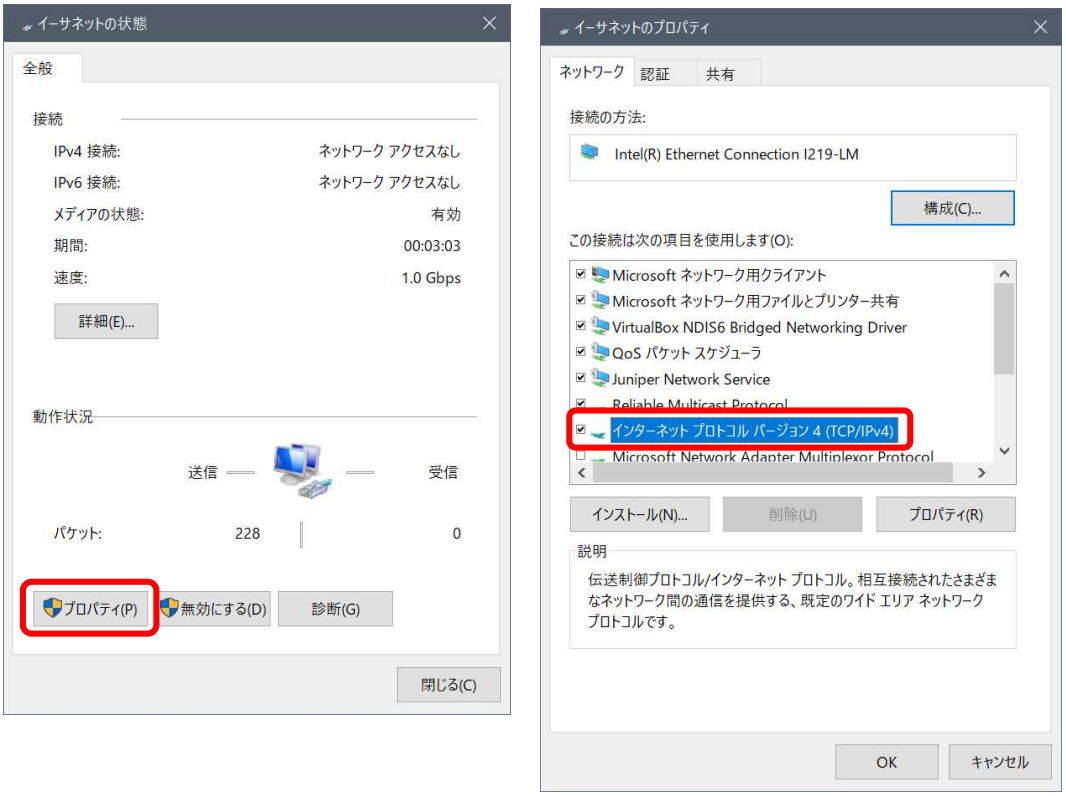


図 6-11. ローカル エリア接続のプロパティ

____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。

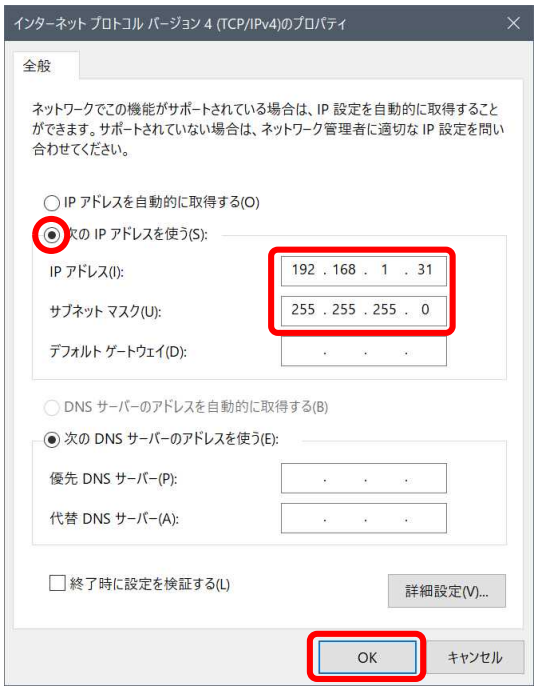


図 6-12. 「IP アドレス」と「サブネット マスク」の設定

____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。

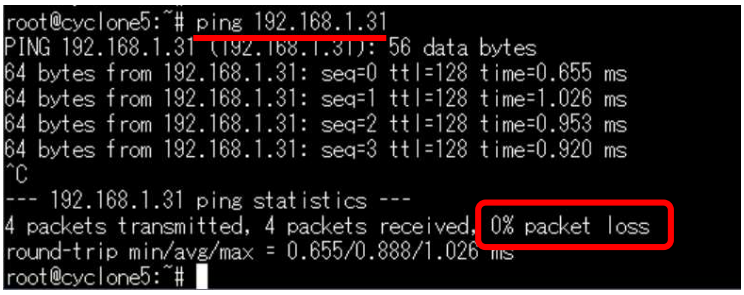


図 6-13. PC に対して ping を実行して接続を確認

____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。

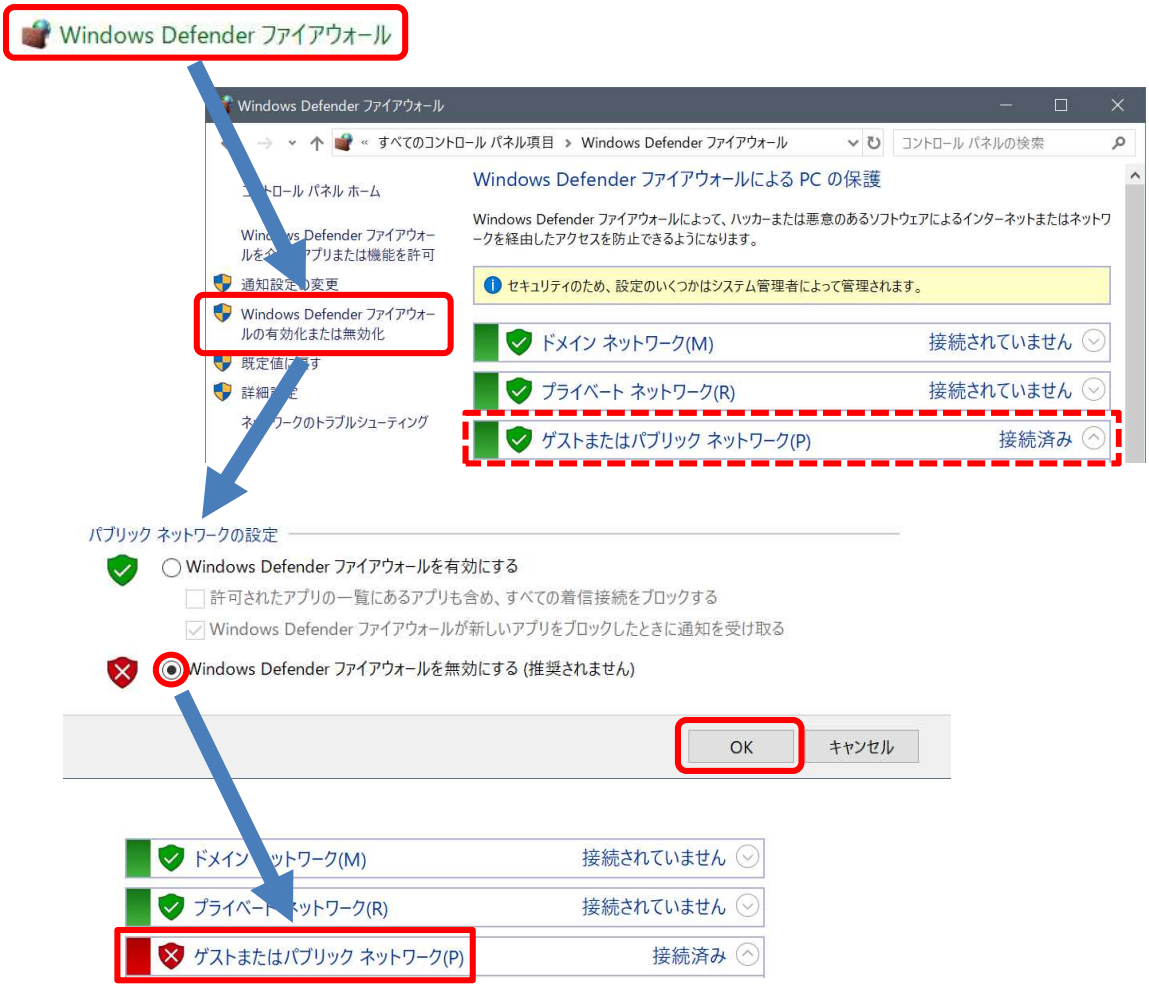


図 6-14. Windows ファイアウォール設定

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。

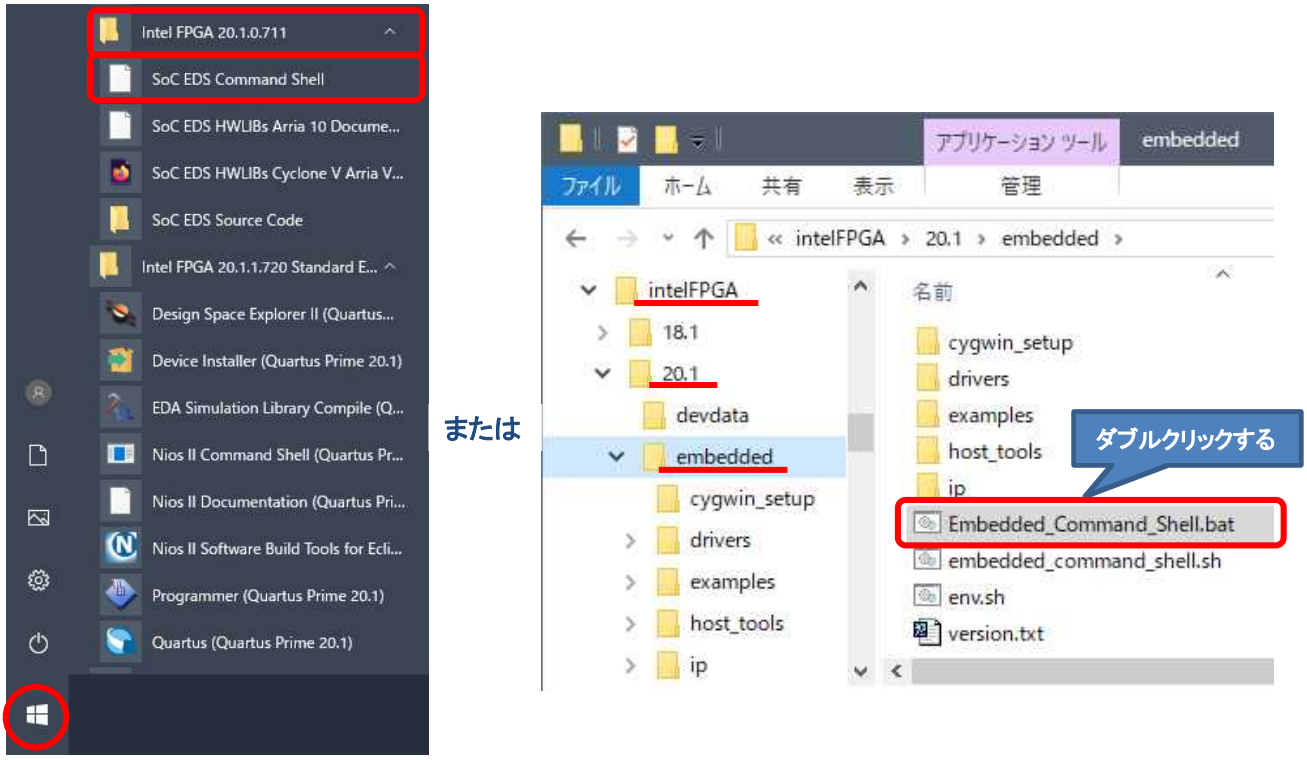


図 6-15. Embedded Command Shell を起動

____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 & ↵
```

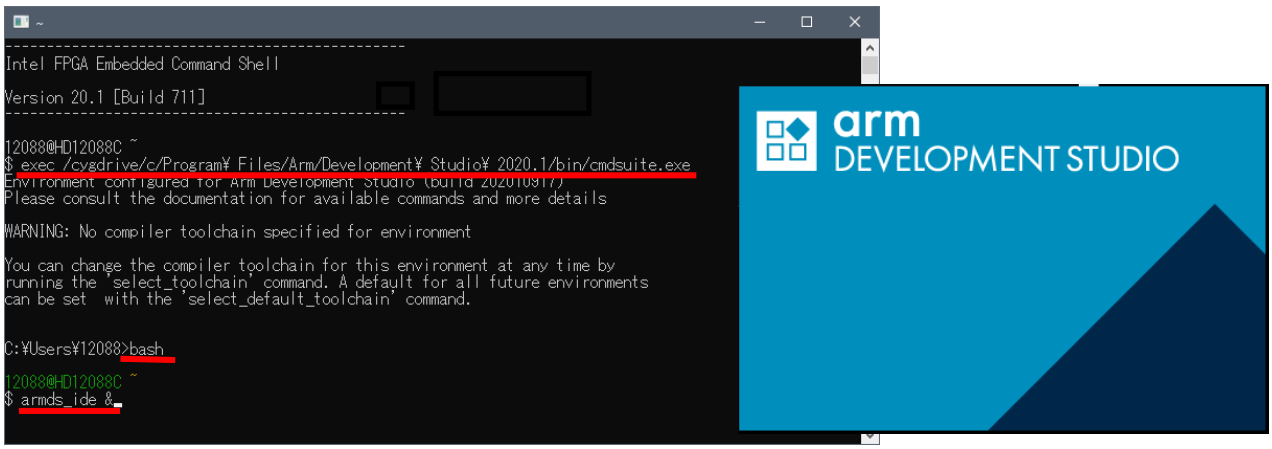


図 6-16. Embedded Command Shell から Arm® DS を起動

____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
```

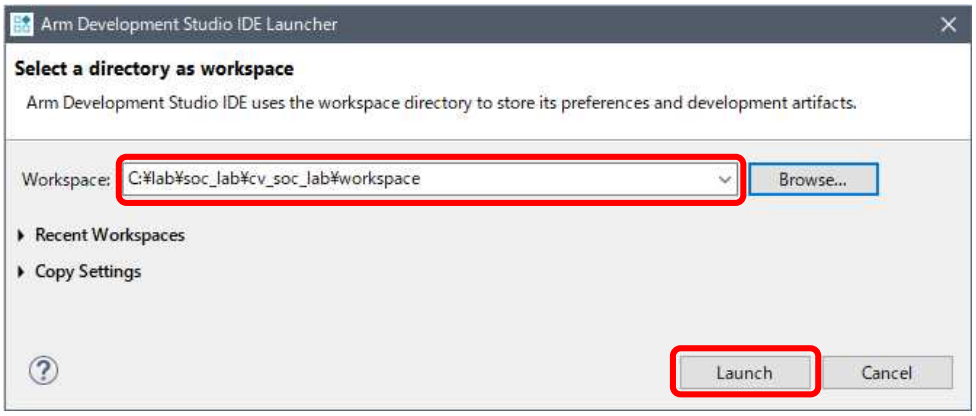


図 6-17. Workspace の作成

____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。

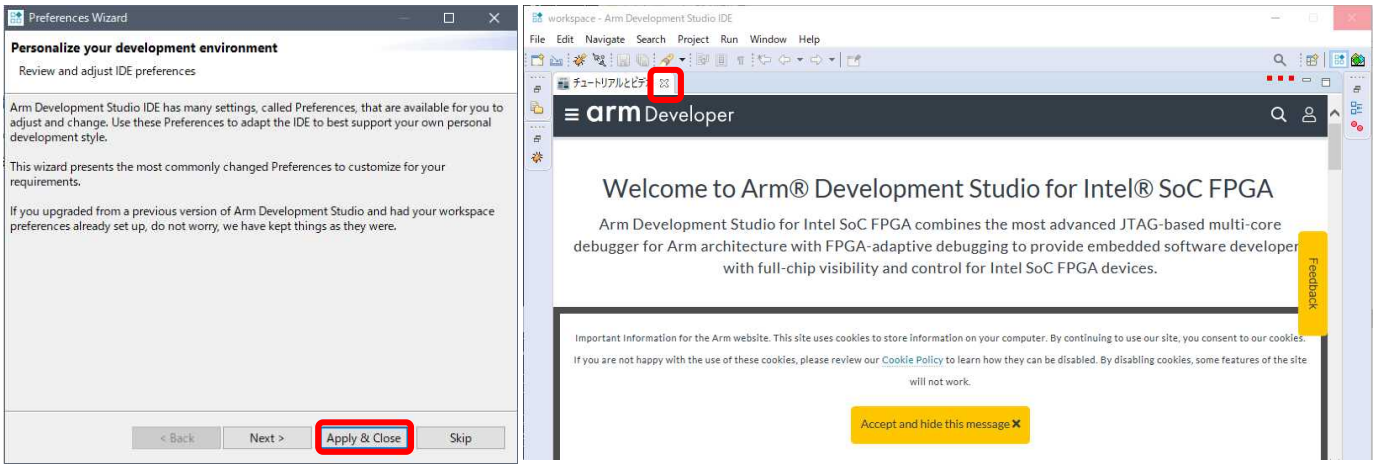


図 6-18. Preference Wizard および Welcome 画面

> ⓘ **Note:**
>
> Arm® DS（バージョン 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た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。

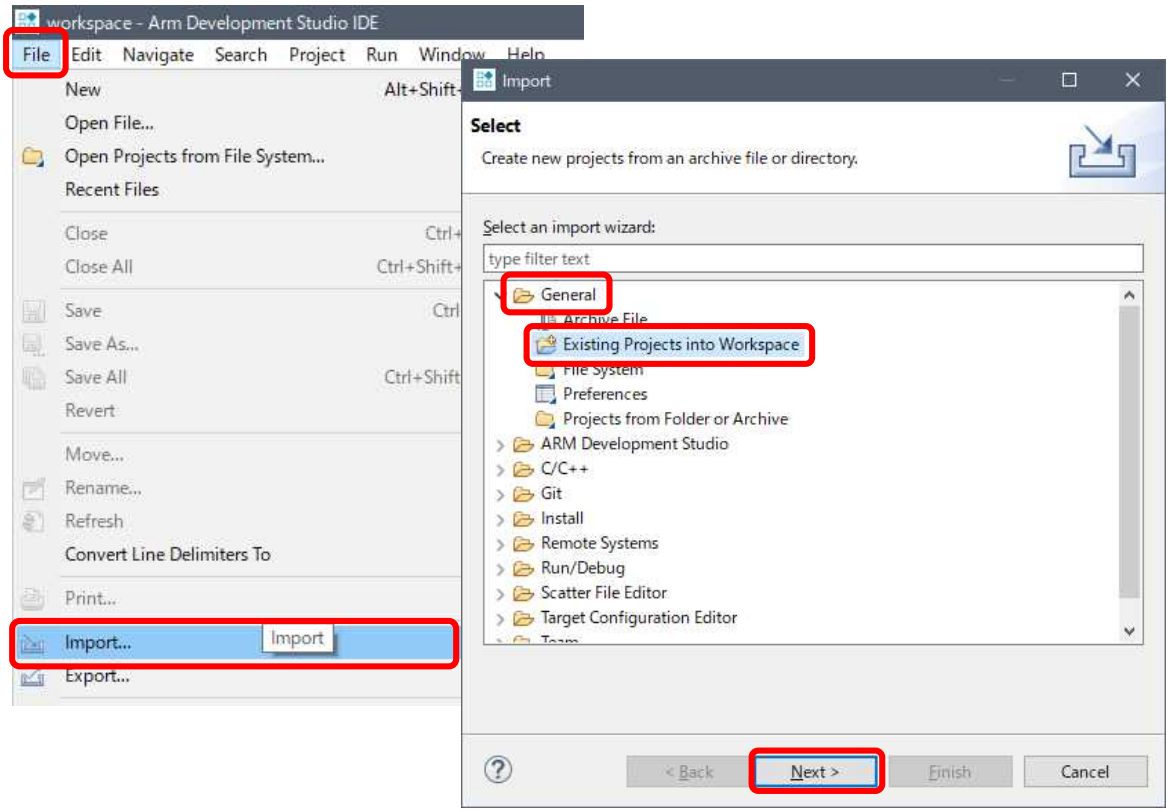


図 6-19.「ファイル」⇒「インポート」

____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。

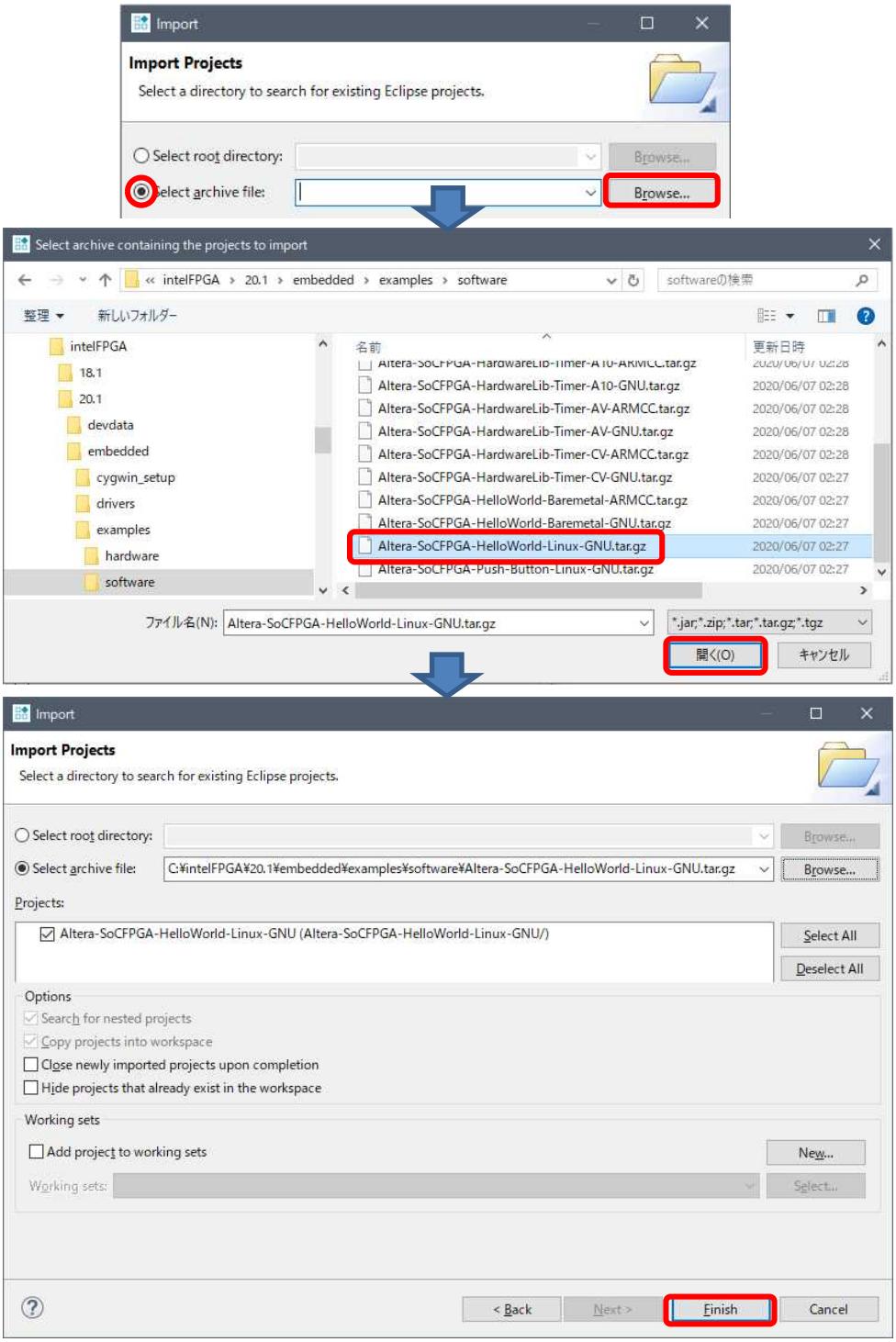


図 6-20. サンプル・プロジェクトのインポート

____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。

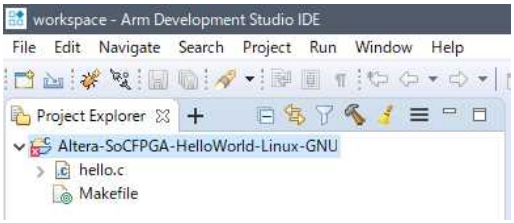


図 6-21. 追加された Altera-SoCFPGA-HelloWorld-Linux-GNU プロジェクト（ツールチェーン未検出）

____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ーを表示します。「***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。

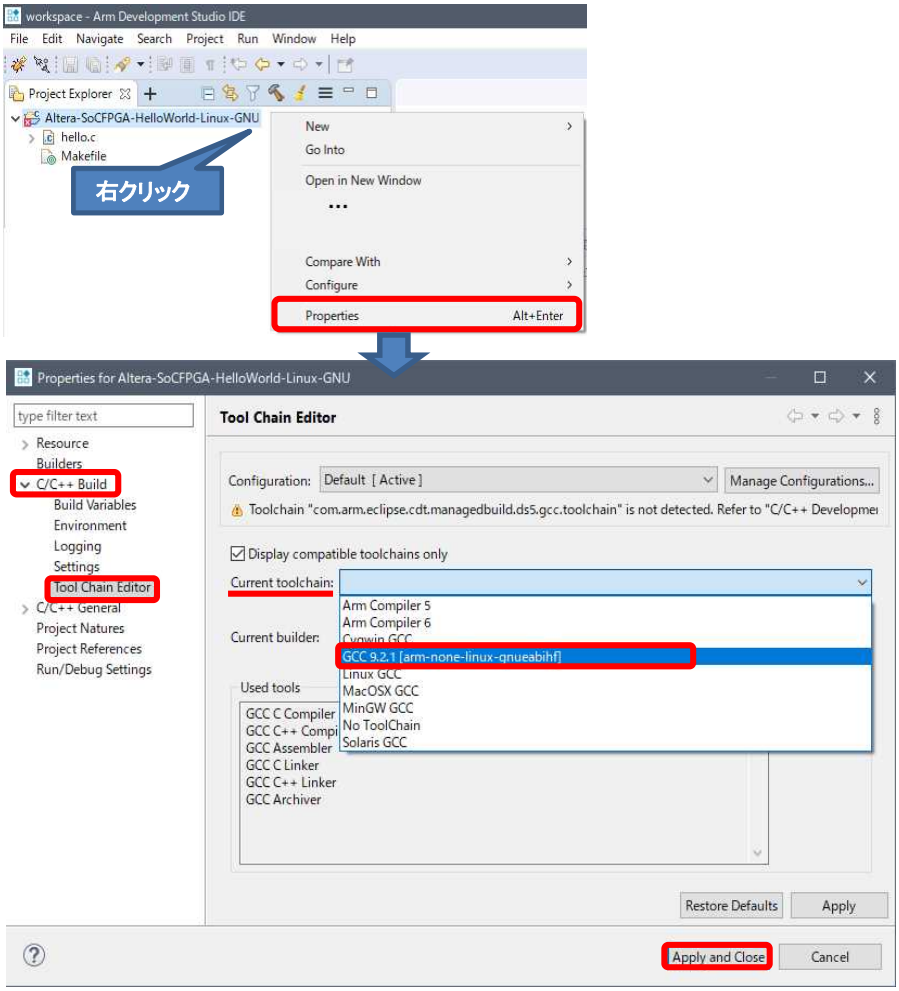


図 6-22. Linux GCC ツールチェーンの選択

____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。

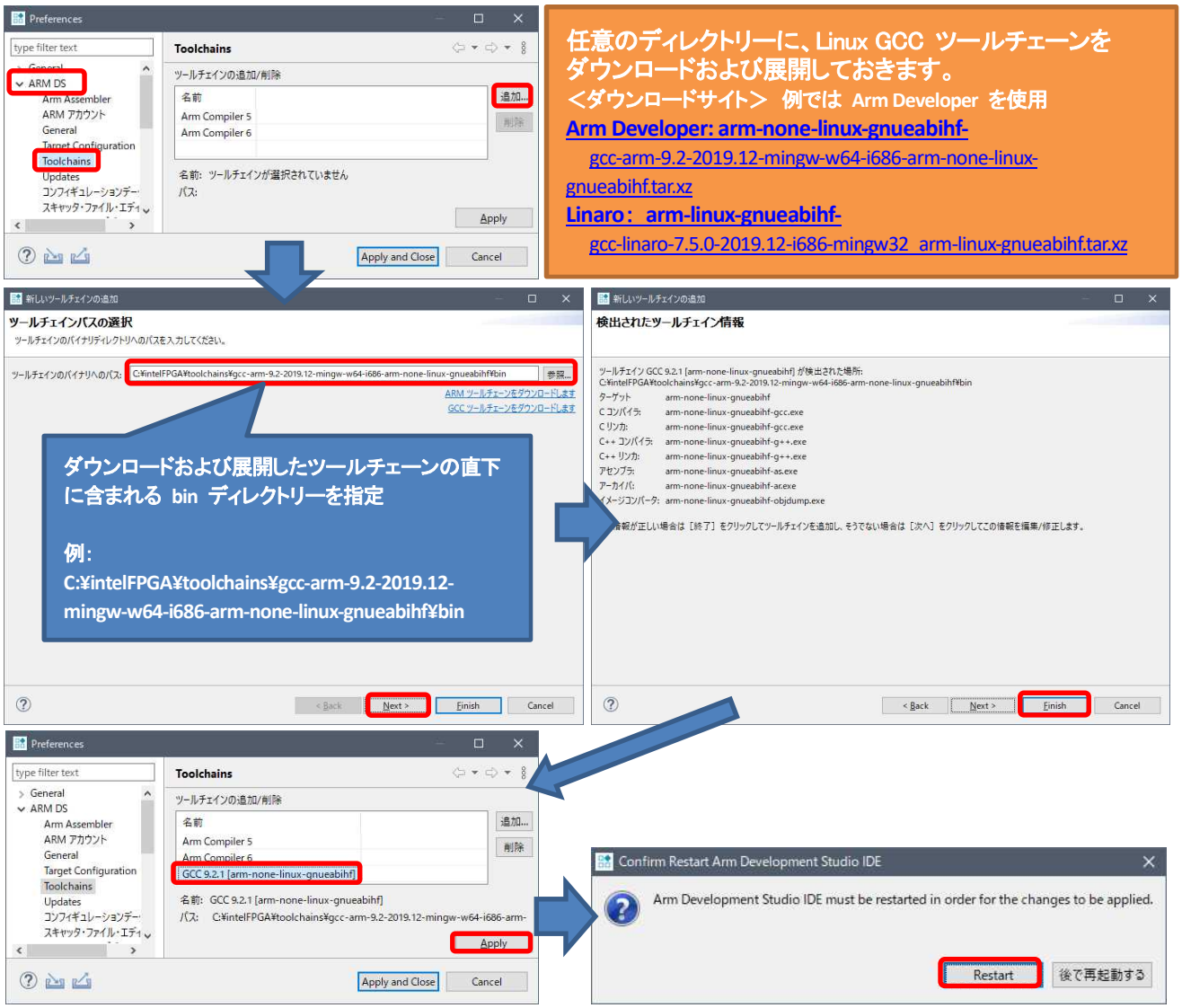


図 6-23. Linux GCC ツールチェーンの追加

____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。

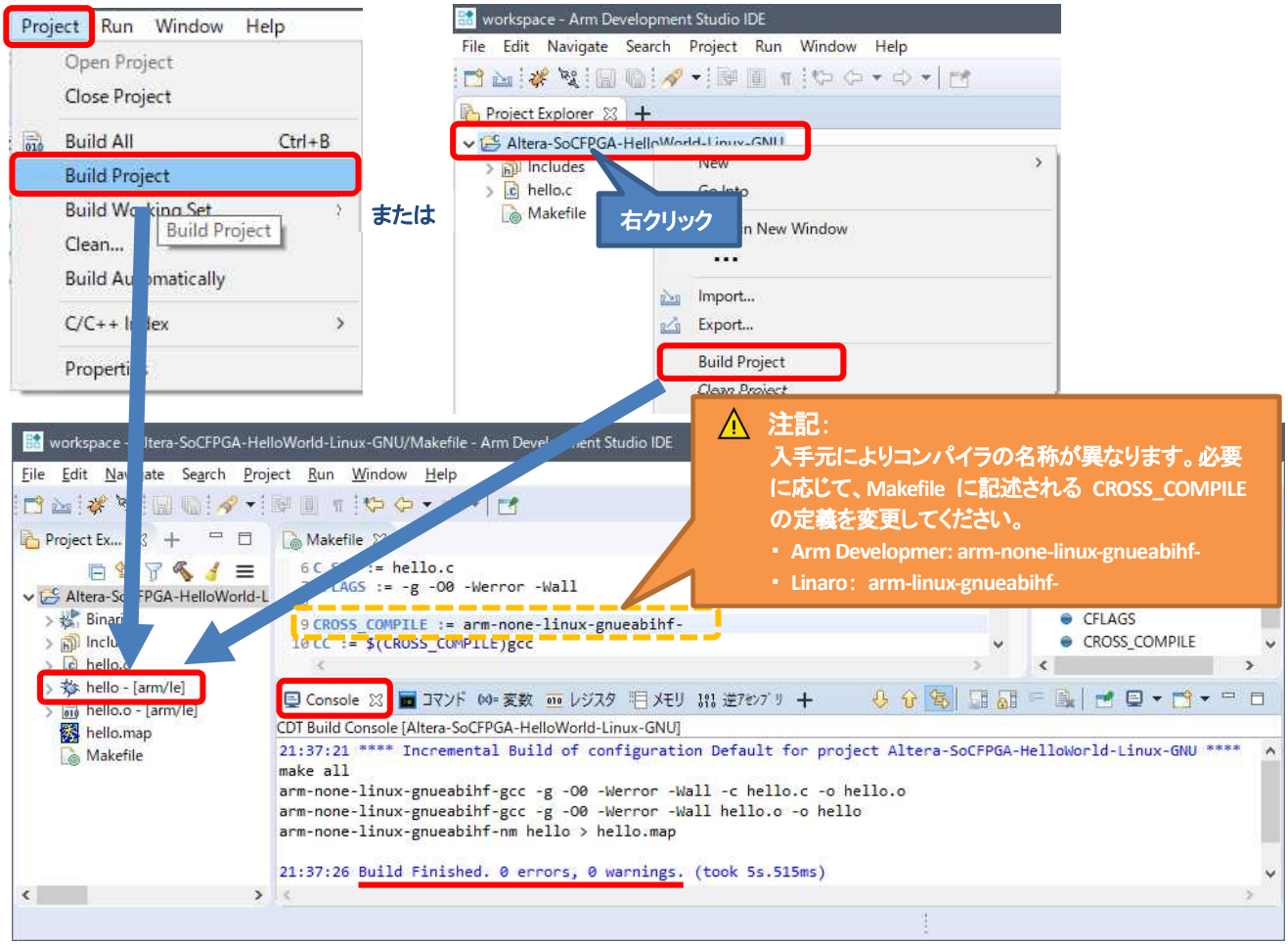


図 6-25. プロジェクトのビルド

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。

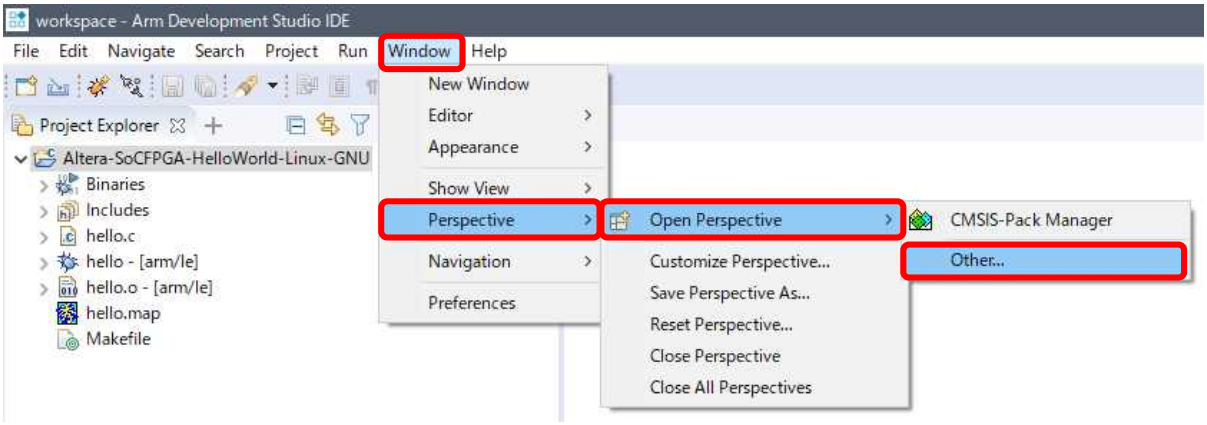


図 6-26. 「パースペクティブを開く」⇒「その他」を選択

____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。

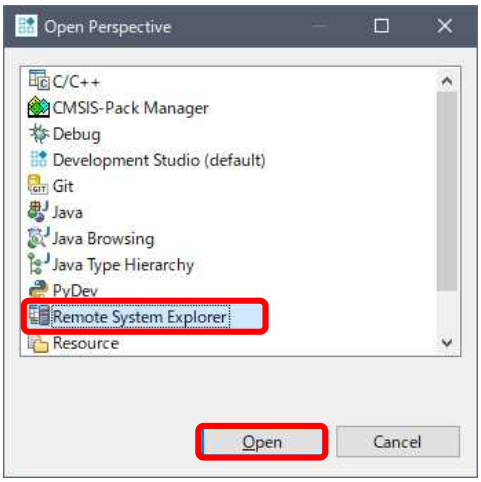


図 6-27. 「リモート・システム・エクスプローラー」の選択

____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。

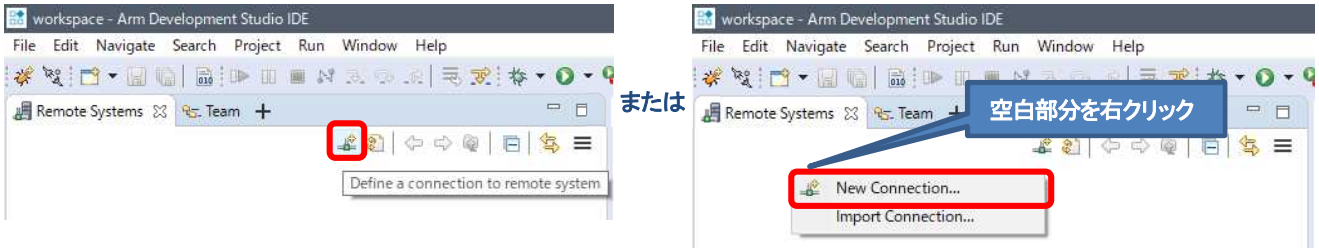


図 6-28. 「リモート・システム・エクスプローラー」での新規接続

____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。

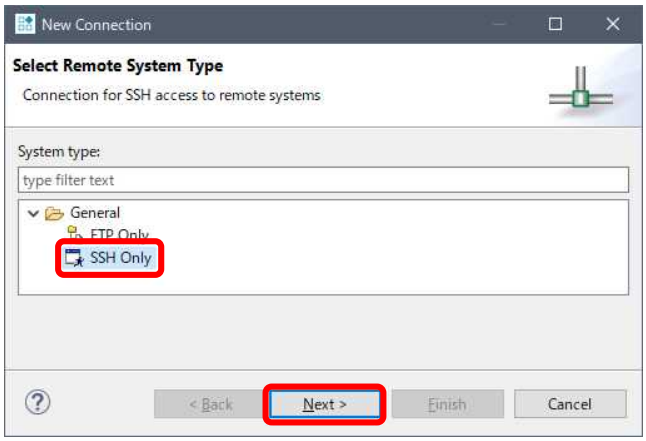


図 6-29.「SSH のみ」を選択

____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。

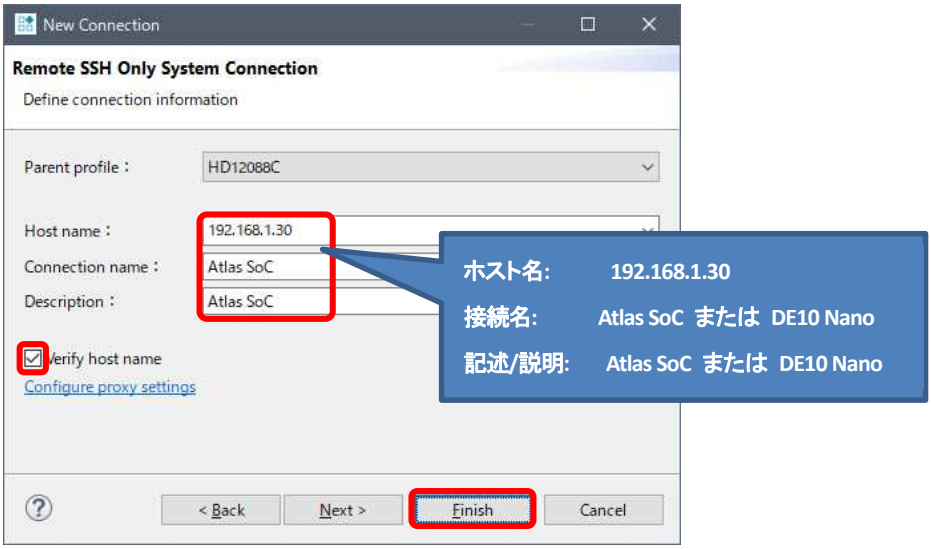


図 6-30. 接続設定

____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。

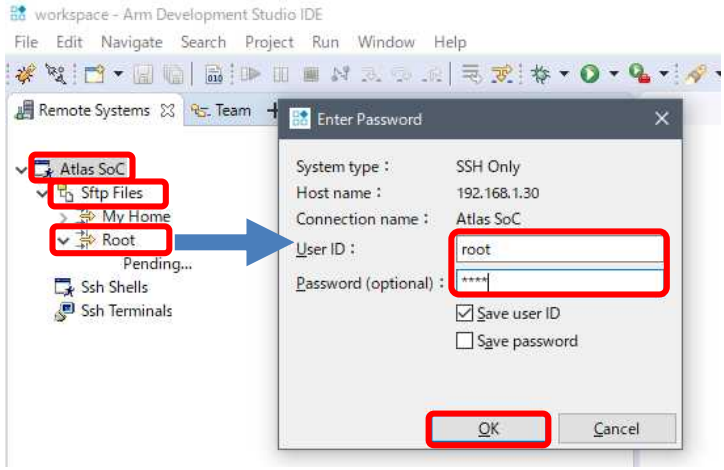


図 6-31. ユーザー ID とパスワードを入力

____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。

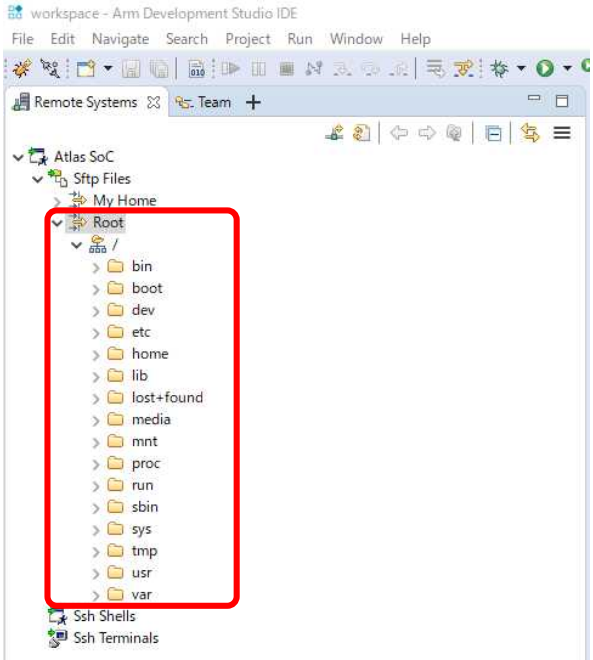


図 6-33. リモート・システム・エクスプローラーに現在のボード上のファイル群が表示

____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。

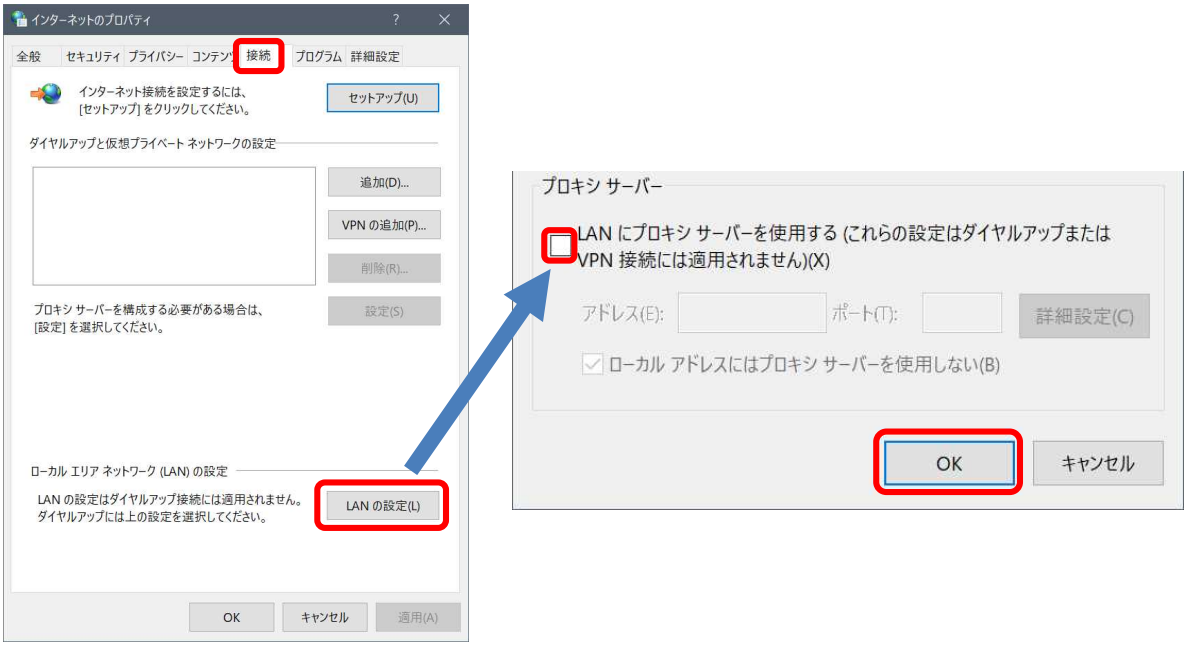


図 6-34. プロキシサーバーの設定

____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-SoCFPGA-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。

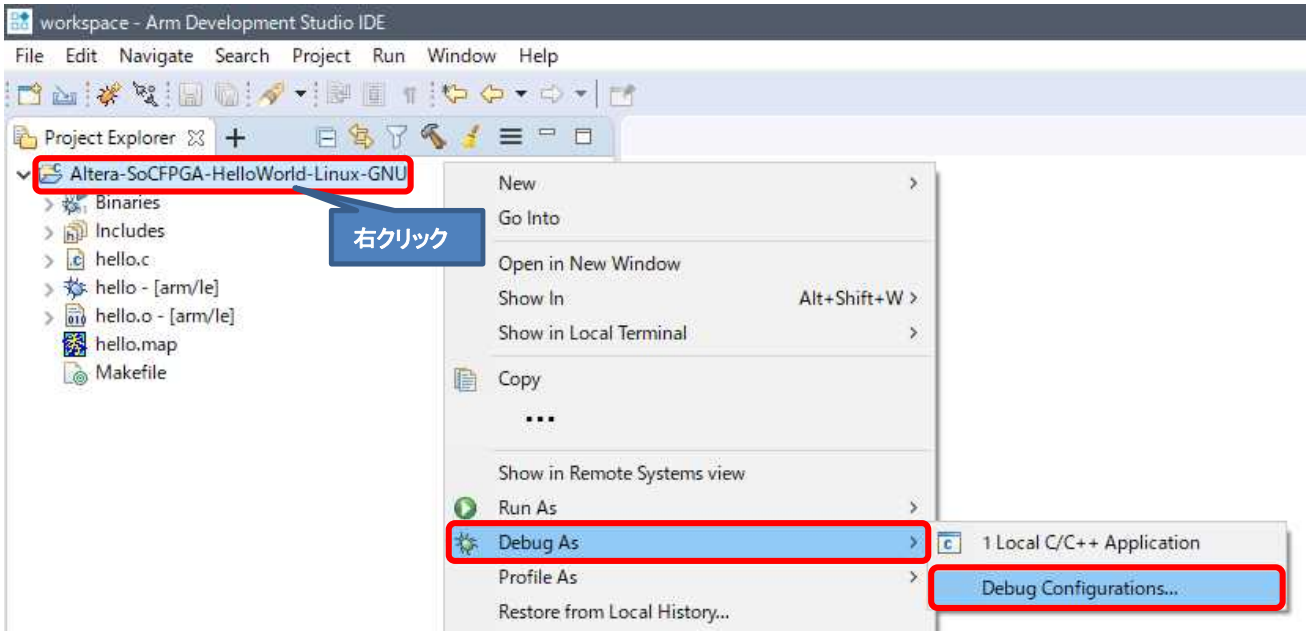


図 6-35. 「デバッグ」⇒「デバッグの構成」を選択

____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。

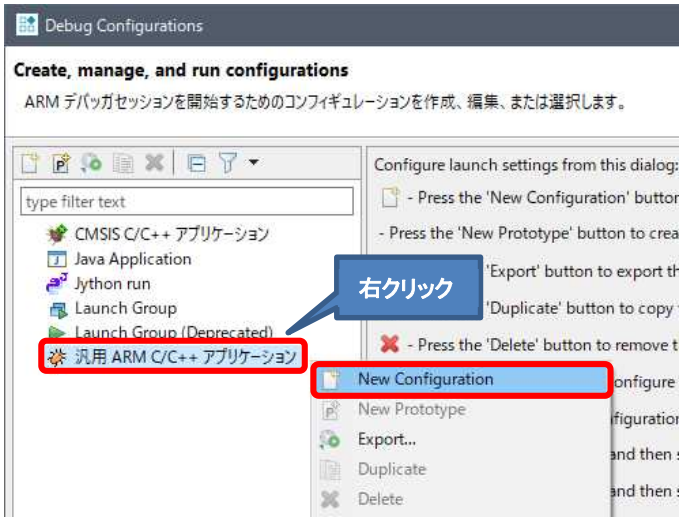


図 6-36. 新しいデバッグ・コンフィグレーションを作成

____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。

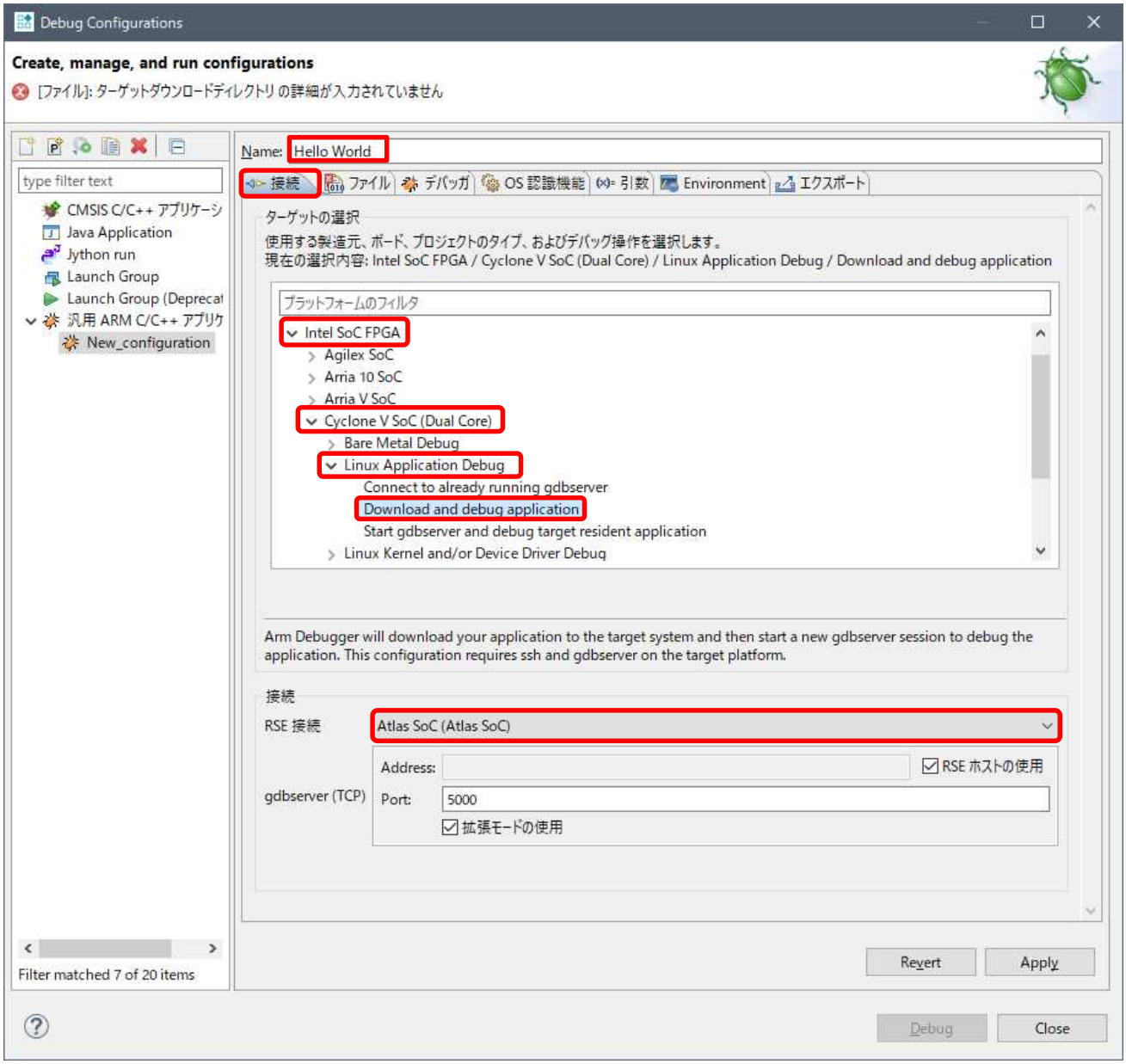


図 6-37. デバッグ構成の設定（1）

____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。

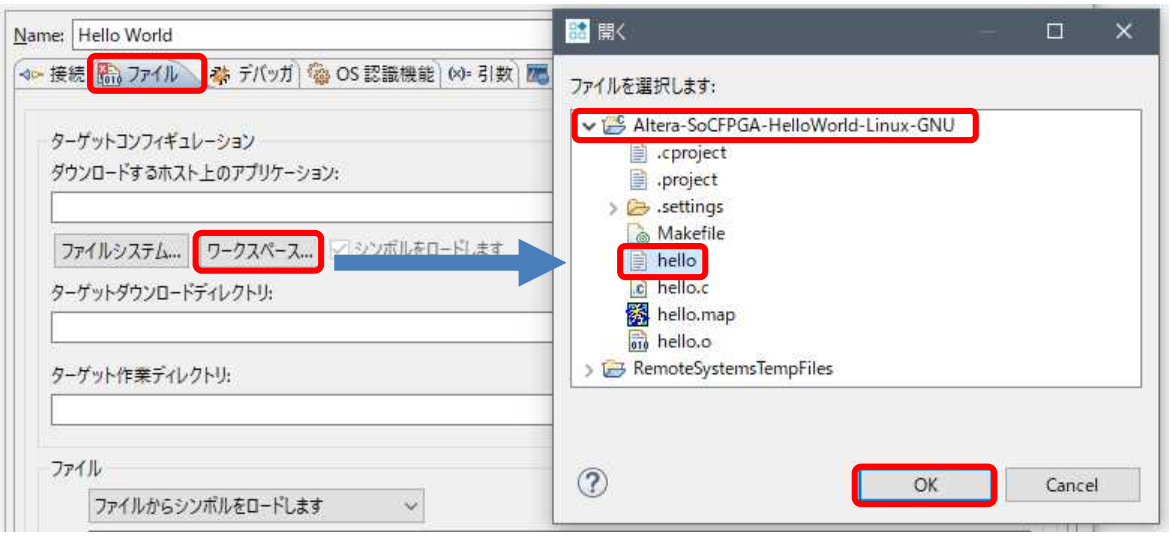


図 6-38. デバッグ構成の設定（2）

____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。

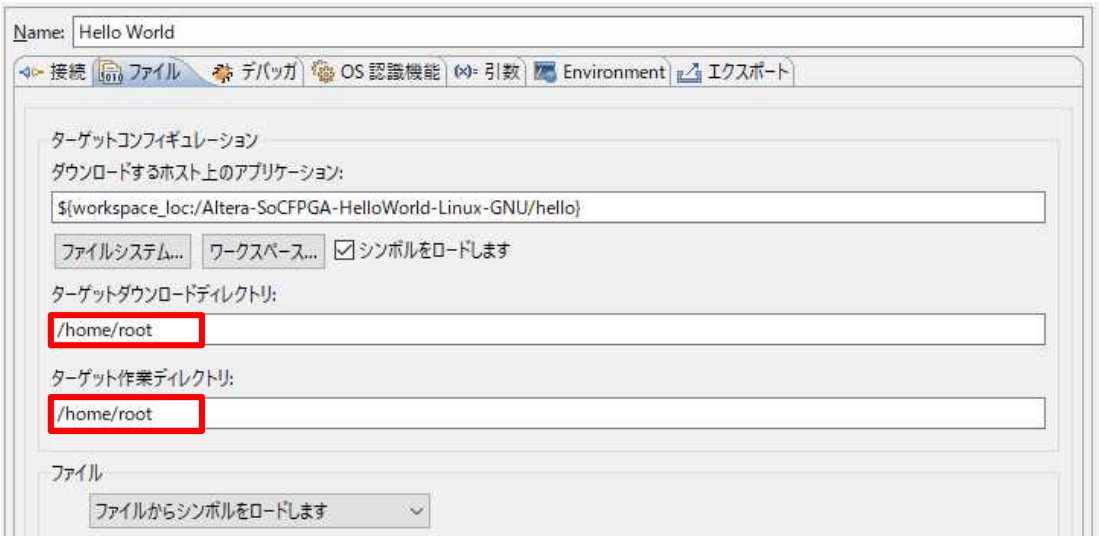


図 6-39. デバッグ構成の設定（3）

____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。

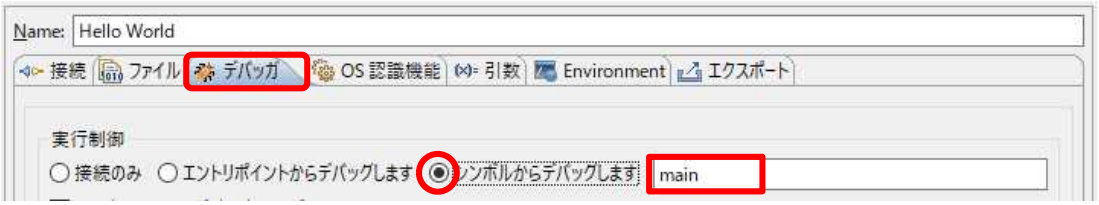


図 6-40. デバッグ構成の設定（4）

____10. ［***デバッグ***］ボタン Debug 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。

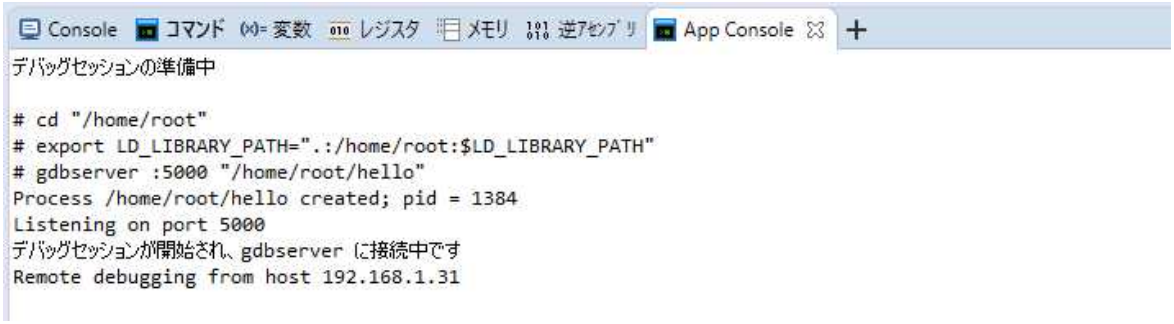


図 6-41. デバッグセッション開始時の App Console

____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。

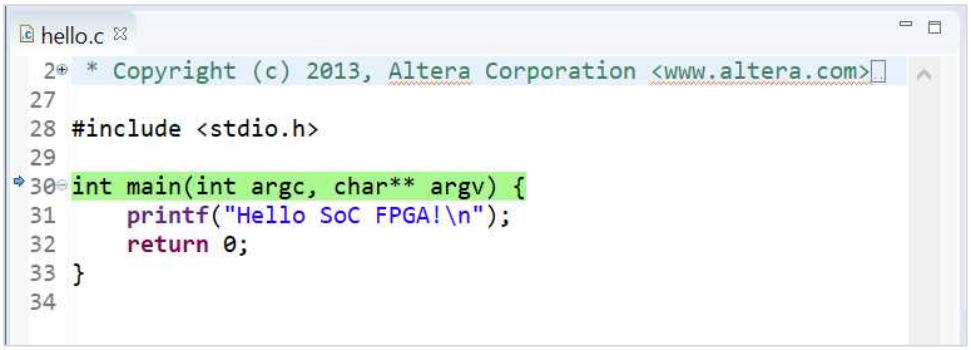


図 6-42. Main 関数でブレーク

____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。

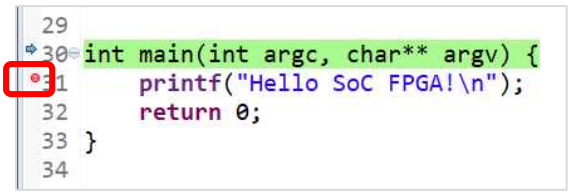


図 6-43. ブレークポイントの設定

____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。

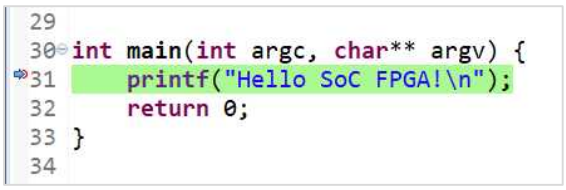


図 6-44. ブレークポイントで停止

____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 デバッガ***」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。

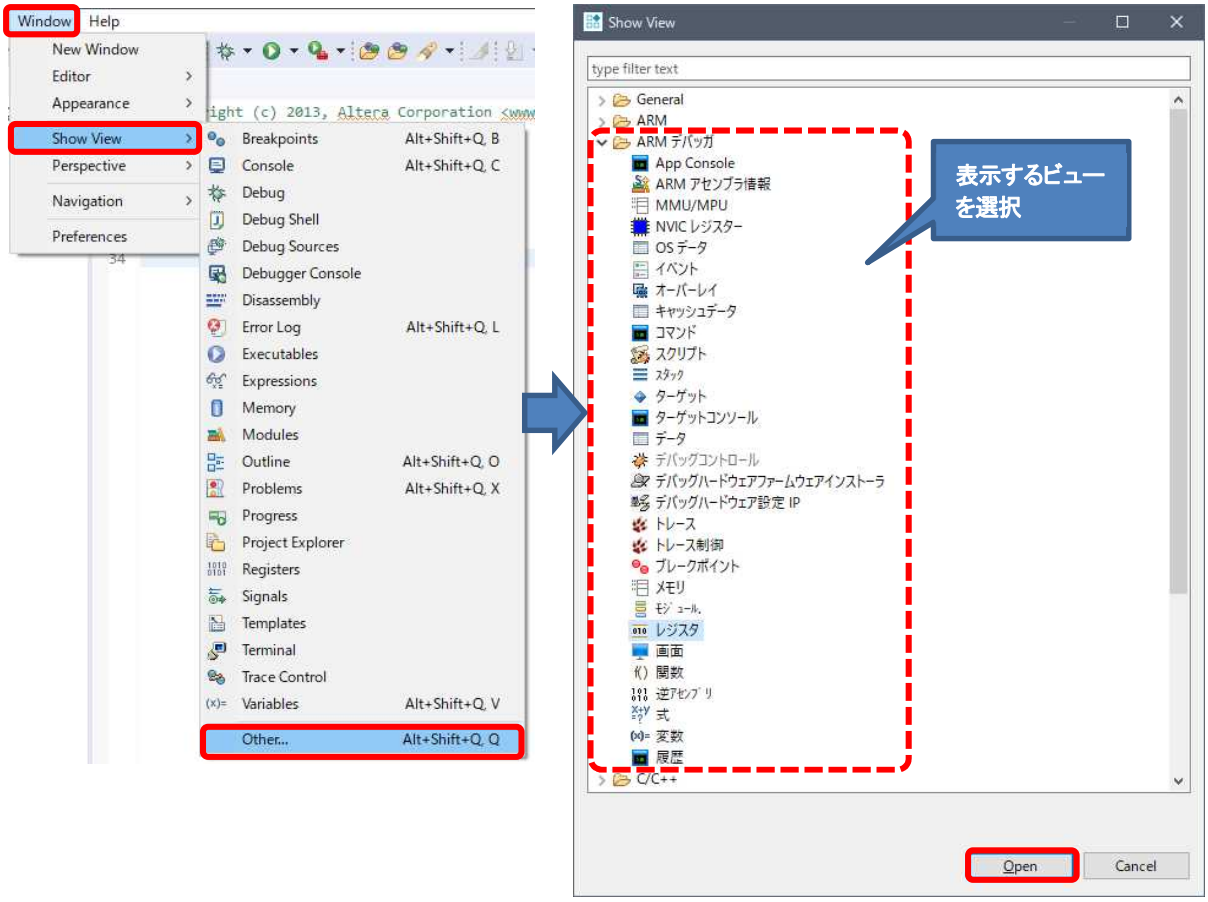


図 6-45. ビューの追加

____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。

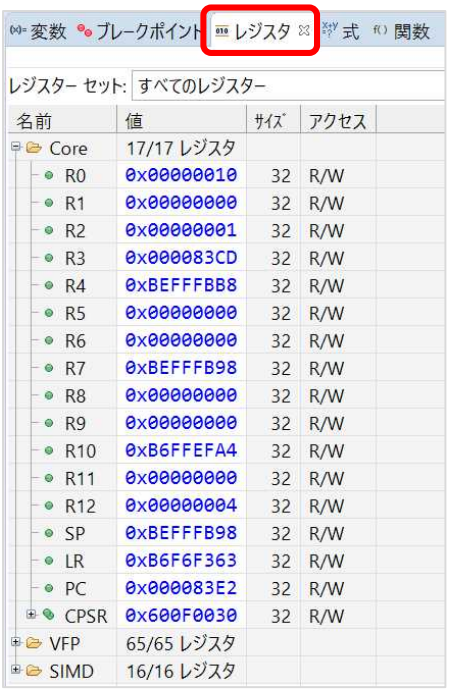


図 6-46. 「レジスタ」ビュー

____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。

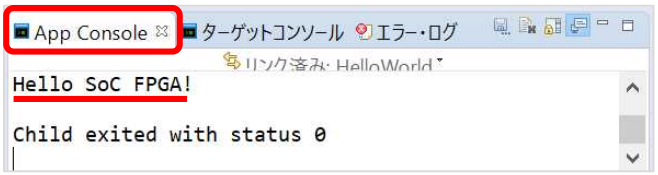


図 6-48. 「App Console」（アプリケーション・コンソール）ビュー

____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。

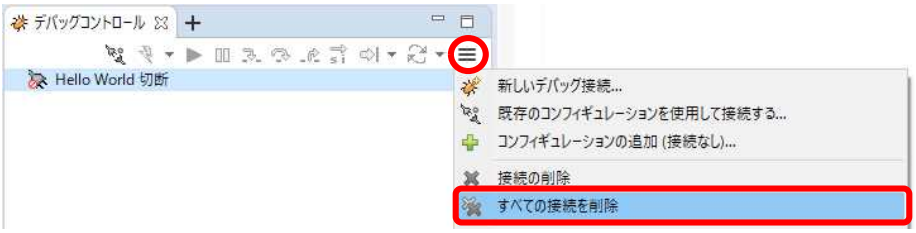


図 6-50. デバッグコントロール「すべての接続を削除」

____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

本演習ではアルテラ® SoC FPGA の開発環境である 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
  - Altera® SoC FPGA まとめページ
  - 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
  - Macnica Altera FPGA Insights
  - Macnica Altera FPGA Insights の技術コンテンツページ
  - Macnica Altera FPGA Insights の FAQ ページ
  - マクニカ半導体事業 : SoC FPGA 関連の記事や資料
  - マクニカ半導体事業 : SoC FPGA 関連の FAQ
  - マクニカ半導体事業 : Altera® FPGA 関連の FAQ
- デバイスやツールについての説明
  - Altera® FPGAs and Programmable Devices - FPGA Documentation
- 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
  - Altera Opensource

改版履歴

Revision	年月	概要
v20.1 r3	2020 年 12 月	SoC EDS v20.1 向けの内容にアップデート
v20.1 r4	2025 年 2 月	書式変更、記載内容の見直し、リンク URL 修正