

# 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

(Arria V SoC / Cyclone V SoC 編)

Ver.18

# 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

(Arria V SoC / Cyclone V SoC 編)

### 目次

1. はじめに .....	4
2. 事前準備 .....	7
2-1. ボードの設定 .....	7
2-1-1. ボード・レイアウト .....	7
2-1-2. 電源およびケーブルの接続 .....	7
2-1-3. SW10 の設定 .....	7
2-2. FPGA デザイン・ファイル .....	8
2-2-1. FPGA デザイン・ファイルの入手先 .....	8
2-2-2. ハードウェア開発での重要な生成物 (ハンドオフ・ファイル) .....	8
2-2-3. FPGA デザイン・ファイルをターゲット・ボードへダウンロードする方法 .....	9
3. SoC FPGA のブート・フロー .....	13
4. ベアメタル・サンプル・アプリケーションを DS-5 で実行する方法 .....	14
4-1. SoC EDS に付属のサンプル・アプリケーションの紹介 (参考) .....	14
4-2. DS-5 の起動 .....	15
4-2-1. Embedded Command Shell の起動 .....	15
4-2-2. DS-5 の起動 .....	15
4-3. ベアメタル・サンプル・アプリケーションのインポート .....	17
4-4. ベアメタル・サンプル・アプリケーションのビルド .....	19
4-4-1. Makefile の例 (参考) .....	19
4-4-2. プロジェクトのビルド .....	20
4-4-3. ベアメタル・サンプル・アプリケーション・プロジェクトのファイル構成 .....	21
4-5. ベアメタル・サンプル・アプリケーションのデバッグ .....	22
4-5-1. デバッグの実行 .....	22
4-5-2. デバッグの構成 (参考) .....	26
4-5-3. デバッグ・スクリプト・ファイルでの実行内容 (参考) .....	27
5. 新規にベアメタル・アプリケーションを作成して DS-5 で実行する方法 .....	31
5-1. ベアメタル・アプリケーションの新規作成 .....	31
5-2. ベアメタル・アプリケーションのビルド .....	36
5-3. ベアメタル・アプリケーションのデバッグ .....	37
6. FPGA レジスタの確認方法 .....	41

## 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 (Arria V SoC / Cyclone V SoC 編)

7. カスタム・ボードへの対応方法 .....	45
7-1. Arm プロセッサを含むハードウェアの設計を行う.....	45
7-2. Preloader (プリローダ) とは? .....	45
7-3. Preloader の生成手順.....	46
7-3-1. Embedded Command Shell の起動 .....	46
7-3-2. bsp-editor (Preloader Generator) の起動 .....	46
7-3-3. 新規 bsp プロジェクトの作成.....	46
7-3-4. ハンドオフ・ファイルの指定 .....	47
7-3-5. Preloader のユーザ・オプション (Common) の設定 .....	48
7-3-6. Preloader のユーザ・オプション (Advanced spl boot) の設定 .....	49
7-3-7. Preloader のユーザ・オプション (Advanced spl debug) の設定.....	50
7-3-8. bsp プロジェクトの生成 (Generate) .....	51
7-3-9. Preloader のビルド.....	52
7-4. カスタム・ボード向けに生成した Preloader を DS-5 デバッグで使用する方 法 .....	54
7-4-1. Preloader ファイルの差し替え.....	54
7-4-2. Makefile の編集.....	55
8. ベアメタル・アプリケーションを SD カードからスタンドアローン実行する例.....	56
8-1. SD カードの準備 .....	56
8-1-1. <b>SoC EDS v13.1 以前のバージョン</b> で Preloader を SD カードに書き込むには.....	57
8-1-2. <b>SoC EDS v14.0 以降のバージョン</b> で Preloader を SD カードに書き込むには.....	57
8-2. DS-5 プロジェクトへの追加ファイル .....	58
8-3. DS-5 プロジェクトの Makefile の修正.....	59
8-4. SD カード実行のためのソース・ファイルの変更 .....	60
8-5. ベアメタル・アプリケーションのビルド.....	60
8-6. ベアメタル・アプリケーションの実行バイナリの作成と起動 .....	61
8-6-1. アプリケーションを SD カードの「FAT 領域」に配置して “u-boot から起動” する方法.....	61
8-6-2. アプリケーションを SD カードの「u-boot 領域」に格納して “Preloader から起動” する方法 ..	63
改版履歴 .....	65

## 1. はじめに

本資料では Arm® DS-5（以下、DS-5）を利用したインテル® SoC FPGA（以下、SoC FPGA）向けベアメタル・アプリケーションの開発およびデバッグ手法について解説しています。

また、本資料では 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）向けの ベアメタル・サンプル・アプリケーション **Atlas-Blinking-LED-Baremetal-GNU** を例として説明しています。

本資料では以下の内容を説明しています。

- ① ハードウェア開発での重要な生成物（ハンドオフ・ファイル）
- ② SoC FPGA のブート・フロー
- ③ ベアメタル・サンプル・アプリケーションを DS-5 で実行する方法として、
  - ・ インテル® SoC FPGA エンベデッド開発スイート（以下、SoC EDS）に付属のサンプル・アプリケーションの紹介
  - ・ DS-5 の起動
  - ・ ベアメタル・サンプル・アプリケーションのインポート
  - ・ ベアメタル・サンプル・アプリケーションのビルドおよびデバッグ
  - ・ デバッグ・スクリプト・ファイルでの実行内容（参考）
- ④ 新規にベアメタル・アプリケーションを作成して DS-5 で実行する方法として、
  - ・ ベアメタル・アプリケーションの新規作成
  - ・ ベアメタル・アプリケーションのビルドおよびデバッグ
- ⑤ FPGA レジスタの確認方法
- ⑥ カスタム・ボードへの対応方法として、
  - ・ Arm プロセッサを含むハードウェアの設計を行う
  - ・ Preloader（プリローダ）とは？
  - ・ Preloader の生成手順
  - ・ カスタム・ボード向けに生成した Preloader を DS-5 デバッグで使用方法
  - ・ 生成した Preloader を SD カードに書き込む方法
- ⑦ ベアメタル・アプリケーションを SD カードからスタンドアロン実行する例
  - ・ SD カードの準備
  - ・ アプリケーションを SD カードの「FAT 領域」に配置して“u-boot から起動”する方法
  - ・ アプリケーションを SD カードの「u-boot 領域」に格納して“Preloader から起動”する方法

本資料の説明で使用している主な開発環境を以下に示します。

【表 1-1】この資料の説明で使用している主な環境

項番	項目	内容
1	ホスト PC	Microsoft® Windows® 7 Professional SP1 (64 bit) 搭載の 64 bit マシン 本資料では、Windows® 7 Professional を使用して動作の確認を行っております。
2	インテル® Quartus® Prime スタンダード・エディション開発ソフトウェア	SoC FPGA のハードウェアを開発するためのツールです。 ソフトウェア開発に必要なハンドオフ・ファイルの生成も行います。 本資料では、Quartus Prime スタンダード・エディション開発ソフトウェア v18.0 を使用しています。 ■ Quartus Prime スタンダード・エディション v18.0 (以降、Quartus Prime) <a href="http://fpgasoftware.intel.com/18.0/?edition=standard&amp;download_manager=dlm3&amp;platform=windows">http://fpgasoftware.intel.com/18.0/?edition=standard&amp;download_manager=dlm3&amp;platform=windows</a> <b>⚠ 注記:</b> 本資料の説明では Cyclone® V を使用していますので、Device データとして Cyclone® V をインストールしておく必要があります。 Quartus Prime の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。 <a href="https://service.macnica.co.jp/library/118817">https://service.macnica.co.jp/library/118817</a>
3	インテル® SoC FPGA エンベデッド開発スイート (SoC EDS) スタンダード・エディション (DS-5 Intel® SoC FPGA Edition)	SoC FPGA のソフトウェアを開発するためのツールです。 ハンドオフ・ファイルを使用して、ターゲット・ボード固有の Preloader (プリローダ) を作成します。また SoC EDS に含まれる DS-5 Intel® SoC FPGA Edition を使用して、アプリケーション・ソフトウェアをコンパイルしデバッグすることができます。 本資料では、SoC EDS スタンダード・エディション v18.0 を使用しています。 ■ SoC EDS スタンダード・エディション v18.0 <a href="http://fpgasoftware.intel.com/soceds/18.0/?edition=standard&amp;platform=windows&amp;download_manager=dlm3">http://fpgasoftware.intel.com/soceds/18.0/?edition=standard&amp;platform=windows&amp;download_manager=dlm3</a> <b>⚠ 注記:</b> インテル® FPGA ダウンロード・ケーブル II (USB-Blaster™ II) を使用したベアメタル・アプリケーションのデバッグには、DS-5 Intel® SoC FPGA Edition (有償版) が必要になります。 SoC EDS の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。 <a href="https://service.macnica.co.jp/library/118873">https://service.macnica.co.jp/library/118873</a>
4	Atlas-SoC ボード または DE10-Nano ボード	本資料の説明でターゲット・ボードとして使用する、Cyclone® V SoC を搭載した Terasic 社 Atlas-SoC / DE10-Nano ボードです。 ■ Atlas-SoC ボード <a href="https://rocketboards.org/foswiki/view/Documentation/AtlasSoCDevelopmentPlatform">https://rocketboards.org/foswiki/view/Documentation/AtlasSoCDevelopmentPlatform</a> <a href="http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&amp;CategoryNo=205&amp;No=941&amp;PartNo=4#Alt">http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&amp;CategoryNo=205&amp;No=941&amp;PartNo=4#Alt</a> ■ DE10-Nano ボード <a href="https://rocketboards.org/foswiki/view/Documentation/DE10NanoDevelopmentBoard">https://rocketboards.org/foswiki/view/Documentation/DE10NanoDevelopmentBoard</a> <a href="https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&amp;CategoryNo=205&amp;No=1046&amp;PartNo=4">https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&amp;CategoryNo=205&amp;No=1046&amp;PartNo=4</a>
5	ベアメタル・サンプル・アプリケーション	本資料の「4. ベアメタル・サンプル・アプリケーションを DS-5 で実行する方法」の説明で使用する、Atlas-SoC / DE10-Nano ボード上で動作する、LED 点滅ベアメタル・サンプル・アプリケーションです。 Arm プロセッサから FPGA ファブリック側に実装された PIO ペリフェラルにアクセスし LED の点灯、消灯を制御します。 実際に動作確認を行う場合は、本資料と併せて以下のファイルを取得してください。 <a href="#">Atlas-Blinking-LED-Baremetal-GNU.zip</a> このファイルを解凍すると、Atlas-Blinking-LED-Baremetal-GNU.tar.gz ファイルが含まれています。 本資料の説明では、Atlas-Blinking-LED-Baremetal-GNU.tar.gz を C:¥Temp に格納したものと説明しています。 <b>⚠ 注記:</b> 本サンプルを実行する前に、ターゲット・ボード (Atlas-SoC / DE10-Nano) に .sof ファイルをダウンロードしてください。

6	Cyclone® V 向け GCC リンカ・スクリプト・ファイル	<p>本資料の「5. 新規にベアメタル・アプリケーションを作成して DS-5 で実行する方法」の説明で使用する、Cyclone® V 向け GCC リンカ・スクリプト・ファイルです。</p> <p>このリンカ・スクリプトでは、64KB の内部 RAM (On-Chip RAM) にプログラムのダウンロードを指定するとともに、DS-5 のセミホスティング機能の利用をリンカへ指示します。</p> <p>実際に動作確認を行う場合は、本資料と併せて以下のファイルを取得してください。  <a href="#">CycloneV-dk-oc-ram-hosted_id.zip</a></p> <p>このファイルを解凍すると、cycloneV-dk-oc-ram-hosted.id ファイルが含まれています。</p> <p>本資料の説明では、cycloneV-dk-oc-ram-hosted.id を <code>C:\Temp</code> に格納したものと説明しています。</p>
7	スタンドアローン 実行用オプション・ ファイル	<p>本資料の「8. ベアメタル・アプリケーションを SD カードからスタンドアローン実行する例」の説明で使用する、スタンドアローン実行用オプション・ファイルです。</p> <p>実際に動作確認を行う場合は、本資料と併せて以下のファイルを取得してください。  <a href="#">Atlas-Blinking-LED-Baremetal-GNU Additional files for stand-alone.zip</a></p> <p>このファイルを解凍すると、  <b>altera-socfpga-unhosted.id</b>  <b>Makefile</b>  <b>startup.s</b>          ファイルが含まれています。</p>


**参考:**

- SoC FPGA のベアメタルに関する基本操作については、以下のユーザ・ガイドが参考になります。
  - 『[Bare Metal User Guide UG-01165](#)』 (英語版)
  - 『[ベアメタルのユーザ・ガイド UG-01165](#)』 (日本語版)
  - 『[UG-01165: Bare Metal User Guide -> Errata - Intel](#)』 (英文ページ)
- SoC FPGA のベアメタル開発者向け情報については、以下のページを参照ください。
  - 『[Intel SoC FPGA Bare-metal Developer Center](#)』 (英文ページ)
- SoC FPGA のベアメタル・プログラミングとハードウェア・ライブラリに関する無償オンライン・トレーニングは、以下のページを参照ください。
  - 『[SoC Bare-metal Programming and Hardware Libraries - Intel](#)』 (英語、28 分)

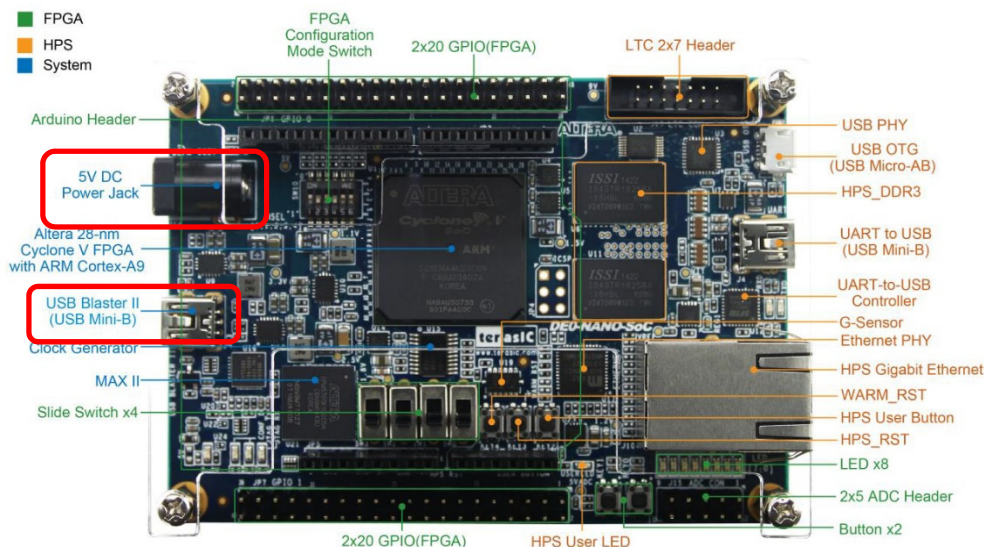
## 2. 事前準備

本資料では ターゲット・ボードとして Atlas-SoC ボード または DE10-Nano ボード を例として説明しています。ここでは、上記ボードを使用する際に必要なボード設定および FPGA デザイン・ファイルについて説明します。

### 2-1. ボードの設定

#### 2-1-1. ボード・レイアウト

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

#### 2-1-2. 電源およびケーブルの接続

AC アダプタ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)を DC 入力 (J14) に接続します。
- Mini USB ケーブルでホスト PC とオン・ボード USB-Blaster™ II コネクタ (J13) を接続します。

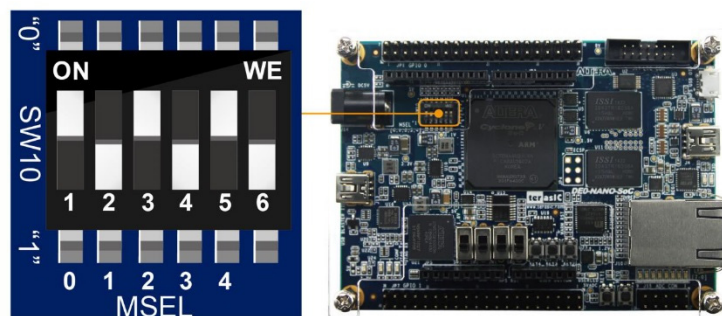
#### 2-1-3. SW10 の設定

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】 ジャンパの設定

## 2-2. FPGA デザイン・ファイル

本資料の説明で使用しているベアメタル・サンプル・アプリケーション Atlas-Blinking-LED-Baremetal-GNU を実行する前に、ターゲット・ボードに FPGA デザイン・ファイル (.sof) をダウンロードしておく必要があります。

### 2-2-1. FPGA デザイン・ファイルの入手先

Terasic 社の以下のページから CD-ROM イメージをダウンロードします。

① Atlas-SoC ボード

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=941&PartNo=4#Alt>

CD-ROM イメージ: DE0-Nano-SoC CD-ROM (rev.D0 Board)

② DE10-Nano ボード

<https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=205&No=1046&PartNo=4>

CD-ROM イメージ: DE10-Nano CD-ROM (rev. B2 Hardware)

❗ **Note:**

本資料の説明では、入手したハードウェア・デザインを C:\Work に格納したものと説明しています。

### 2-2-2. ハードウェア開発での重要な生成物 (ハンドオフ・ファイル)

ベアメタル・アプリケーションの開発およびデバッグでは、ハードウェアの開発において最終的に生成されたフォルダとファイルを使用します。

これらのフォルダとファイルを「ハンドオフ・ファイル」と呼びます。ベアメタル開発において、特に重要な「ハンドオフ・ファイル」は次の 3 つです。

(1) <Quartus プロジェクト>\output\_files フォルダ

正しく生成されていれば、通常は output\_files フォルダの中に .sof という拡張子のファイルが出力されているはずですが (Quartus プロジェクトの設定や構成に依存します)。このファイルを SoC FPGA にプログラムして FPGA をコンフィグレーションします。

⚠ **注記:**

この説明では、Terasic 社のページからダウンロードした CD-ROM イメージを使用して、ハンドオフ・ファイルを生成しています。

Atlas-SoC ボードの場合は、以下の例のように output\_files フォルダがなく、DE0\_NANO\_SOC\_GHRD フォルダの直下に .sof ファイルが生成されることに注意してください。

Atlas-SoC ボードの場合は、

C:\Work\DE0-Nano-SoC\_v.1.3.0\_HWrevD0\_SystemCD\Demonstrations\SoC\_FPGA\DE0\_NANO\_SOC\_GHRD\soc\_system.sof

DE10-Nano ボードの場合は、

C:\Work\DE10-Nano\_v.1.2.2\_HWrevB2\_SystemCD\Demonstrations\SoC\_FPGA\DE10\_NANO\_SoC\_GHRD\output\_files\DE10\_NANO\_SoC\_GHRD.sof

(2) <Quartus プロジェクト>\hps\_isw\_handoff\soc\_system\_hps\_0 フォルダ

正しく生成されていれば、hps\_isw\_handoff\soc\_system\_hps\_0 フォルダの中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、「7-3. Preloader の生成手順」に利用します。Preloader 生成のために使用する bsp-editor (Preloader Generator) ツールで、この hps\_isw\_handoff\soc\_system\_hps\_0 フォルダのパスを指定するので覚えておいてください。



## (3) &lt;Quartus プロジェクト&gt;¥&lt;Platform Designer プロジェクト&gt;¥synthesis フォルダ

正しく生成されていれば、synthesis フォルダの中に .svd という拡張子のファイルが出力されているはず。この .svd ペリフェラル記述ファイルは、「6. FPGA レジスタの確認方法」に利用します。

DS-5 Intel® SoC FPGA Edition で、この <Platform Designer プロジェクト>¥synthesis フォルダのパスを指定するので覚えておいてください。


**注記:**

この説明では、Terasic 社のページからダウンロードした CD-ROM イメージを使用して、ハンドオフ・ファイルを生成しています。 .svd ファイルは下記の場所に生成されます。

**Atlas-SoC ボードの場合、**

C:\Work¥DE0-Nano-SoC\_v.1.3.0\_HWrevD0\_SystemCD¥Demonstrations¥SoC\_FPGA¥DE0\_NANO\_SoC\_GHRD¥soc\_system¥synthesis¥soc\_system\_hps\_0\_hps.svd

**DE10-Nano ボードの場合、**

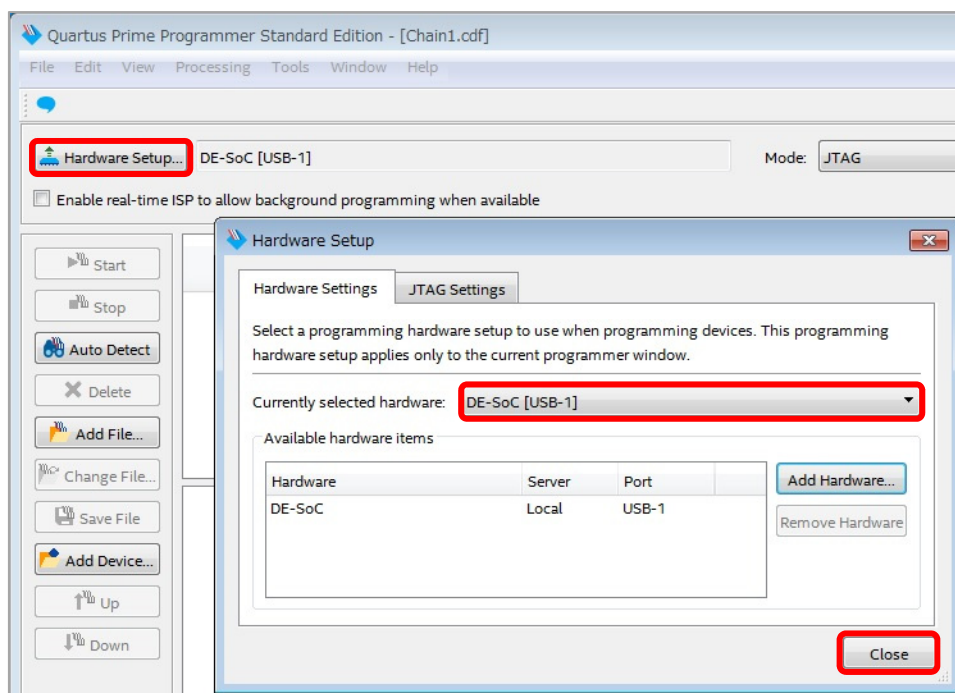
C:\Work¥DE10-Nano\_v.1.2.2\_HWrevB2\_SystemCD¥Demonstrations¥SoC\_FPGA¥DE10\_NANO\_SoC\_GHRD¥soc\_system¥synthesis¥soc\_system\_hps\_0\_hps.svd

## 2-2-3. FPGA デザイン・ファイルをターゲット・ボードへダウンロードする方法

上記で入手したハードウェア・デザイン (sof ファイル) を FPGA にダウンロードします。

「2-1. 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 アダプタを接続してください。

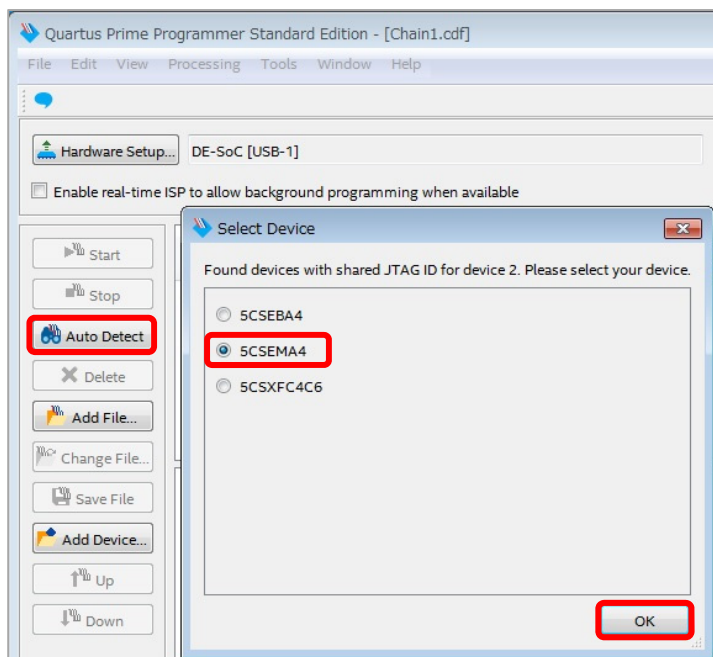
- \_\_\_ 1. Quartus Prime メニューの「Tools」⇒「Programmer」、または Programmer アイコン  をクリックし、Programmer を起動します。
- \_\_\_ 2. Programmer 内にある [Hardware Setup] ボタンをクリックし、Hardware Setup ウィンドウ内の **Currently selected hardware** のプルダウンリストから DE-SoC を選択し、ウィンドウを Close します。



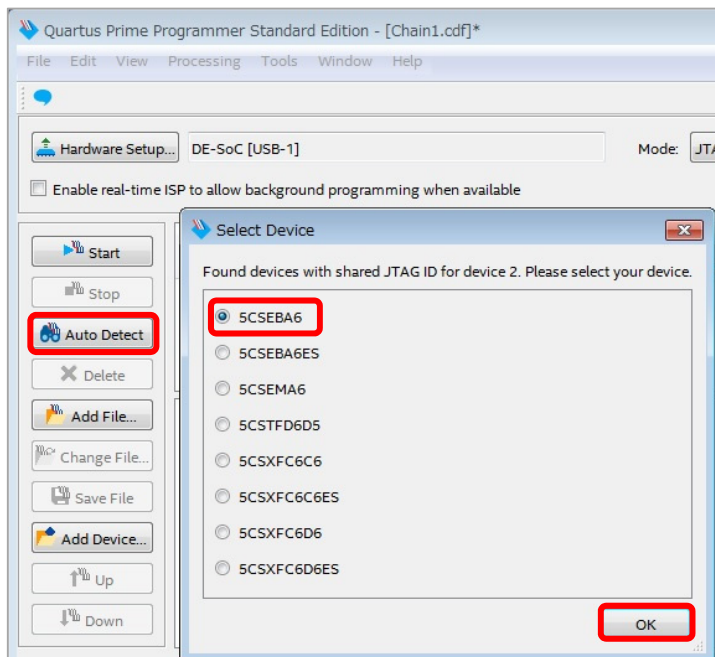
【図 2-3】 Hardware Setup

- \_\_\_ 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  
DE10-Nano ボードの場合は 5CSEBA6  
 を選択し、**[OK]** をクリックします。

#### Atlas-SoC ボードの場合

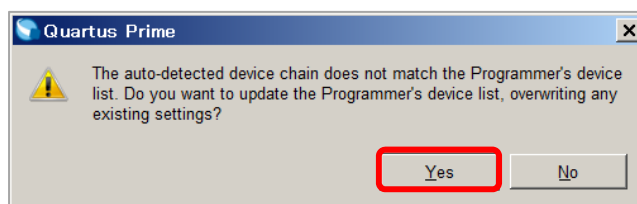


#### DE10-Nano ボードの場合



【図 2-4】 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 2-5】 ダイアログ・ボックス

これにより、JTAG チェイン 上に SOCVHPS と 5CSMA4 / 5CSEBA6 が表示されます。SOCVHPS は Hard Processor System (以降 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・ボックスにおいて、

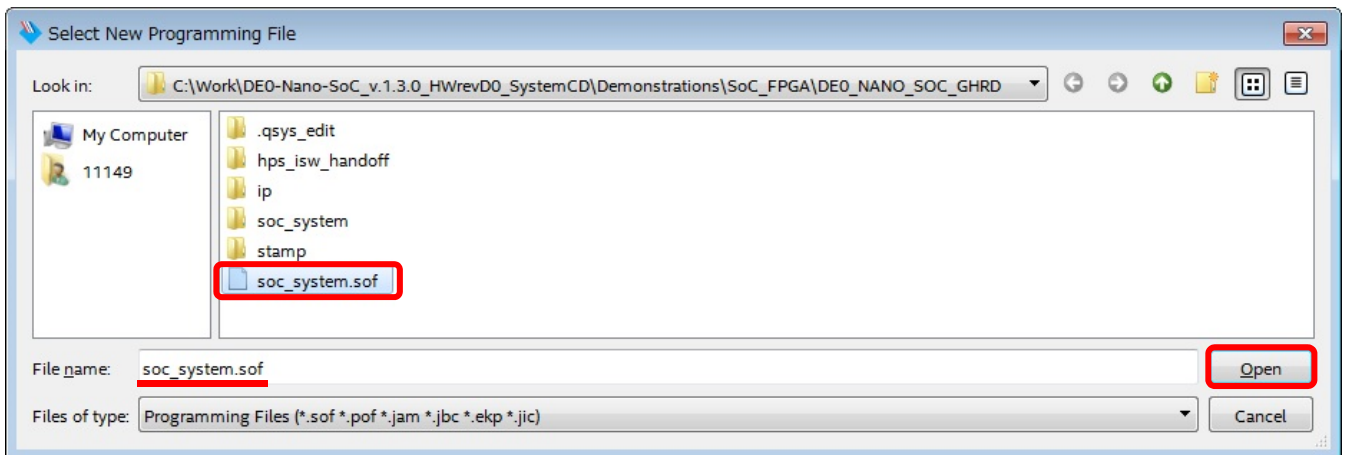
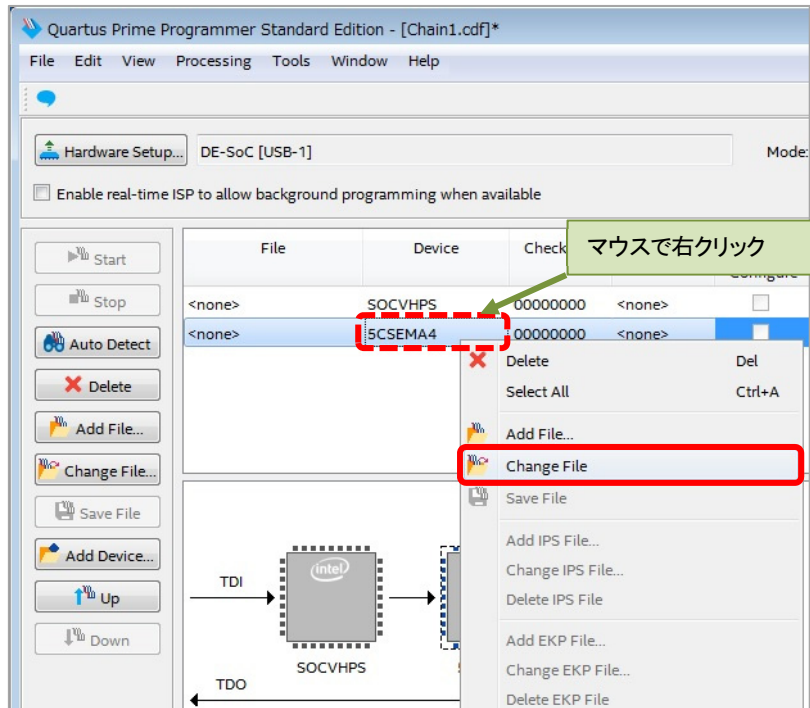
Atlas-SoC ボードの場合は、

C:\Work\DE0-Nano-SoC\_v.1.3.0\_HWrevD0\_SystemCD\Demonstrations\SoC\_FPGA\DE0\_NANO\_Soc\_GHRD\soc\_system.sof

DE10-Nano ボードの場合は、

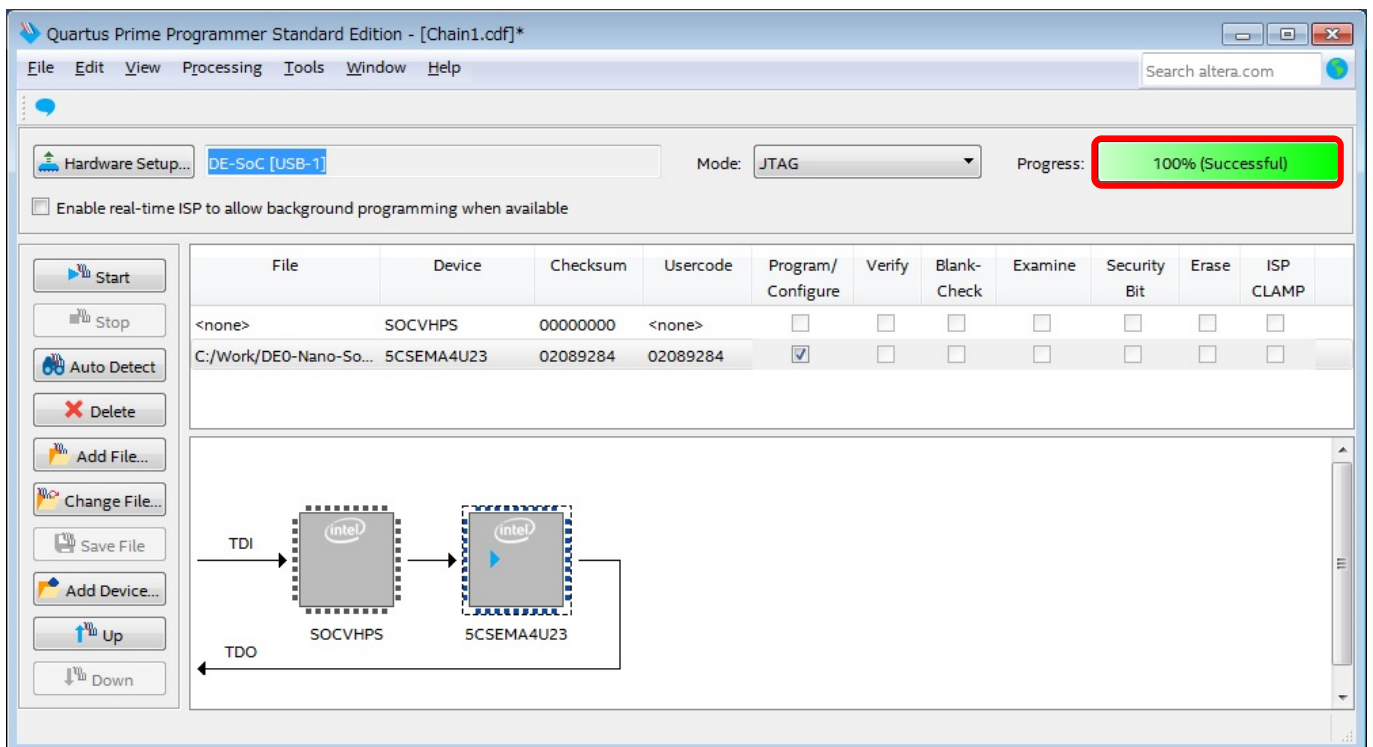
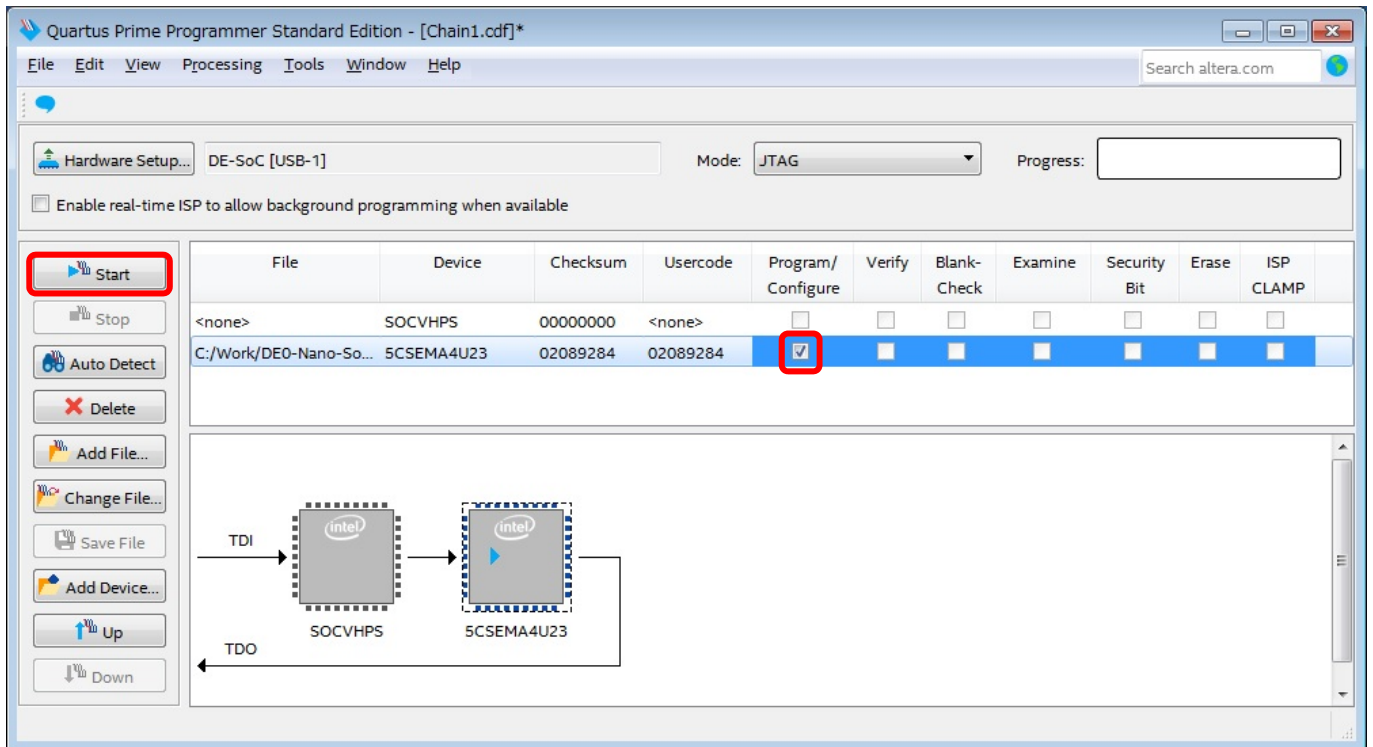
C:\Work\DE10-Nano\_v.1.2.2\_HWrevB2\_SystemCD\Demonstrations\SoC\_FPGA\DE10\_NANO\_Soc\_GHRD\output\_files\DE10\_NANO\_Soc\_GHRD.sof

を選択します。



【図 2-6】 sof ファイルの選択 (Atlas-SoC ボードの場合の例)

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。この動作により FPGA 側に動作イメージが書き込まれた状態となります。



【図 2-7】 sof のダウンロード (Atlas-SoC ボードの場合の例)

### 3. SoC FPGA のブート・フロー

まずはじめに SoC FPGA のブート・フローについて説明します。

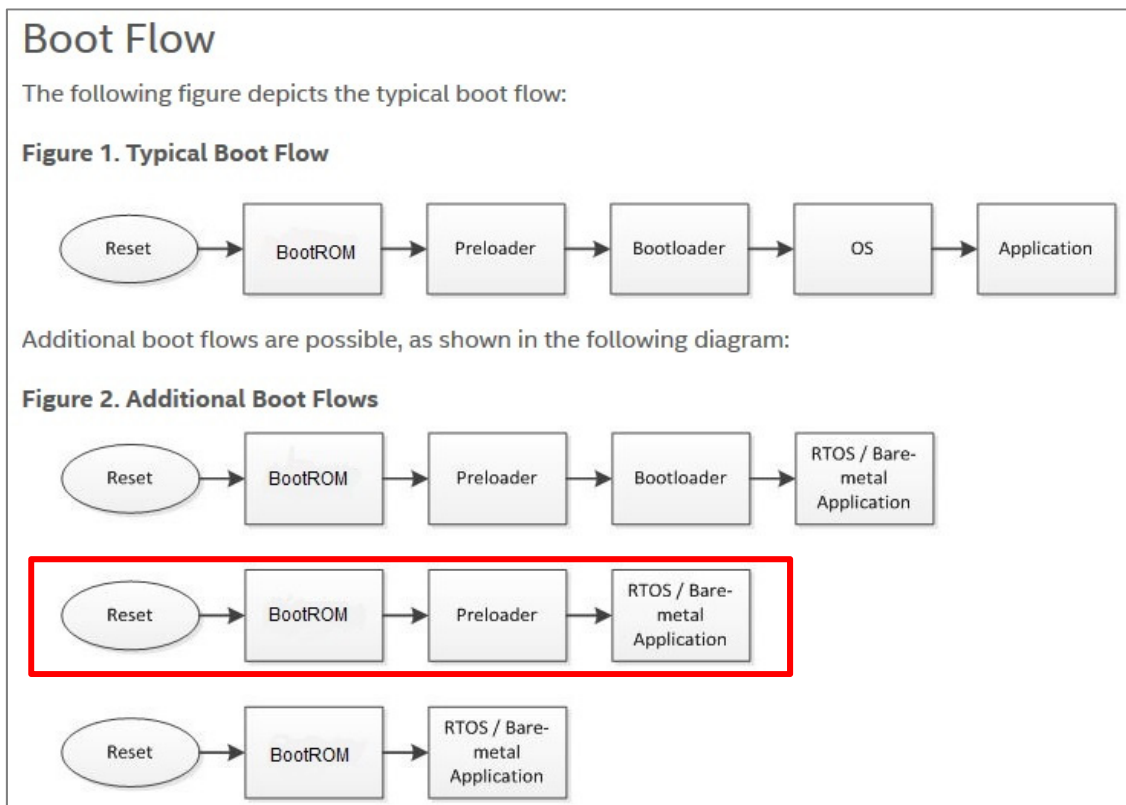
以下の図の通り、Arm のブート・フローには複数のステージが存在します。

ブート・フローに関する詳細は、アプリケーション・ノート 709 (AN 709) をご確認ください。

<https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/an/an709.pdf>

ベアメタル・アプリケーションの場合の多くは、以下赤枠で示した Preloader から直接ベアメタル・アプリケーションを起動する方法が用いられます。

本資料でもこのブート・フローを実現するための仕組みについて解説しています。



【図 3-1】 SoC FPGA のブート・フロー

- **BootROM**

SoC FPGA の内蔵オンチップ ROM に焼き込まれているブート・コードです (ユーザ変更不可)。

- **Preloader**

ハンドオフ・ファイルの情報を元に、HPS IO や SDRAM コントローラの初期化など動作するために必要な処理を実行します。

- **Bootloader**

ユーザ・ブートローダ・コード (U-Boot など) です。アプリケーションや OS に依存します。

- **Baremetal Application**

インテル® のハードウェア・ライブラリ (HWLib) を使用した、直接ハードウェアを読み書きするアプリケーションです。

## 4. ベアメタル・サンプル・アプリケーションを DS-5 で実行する方法

この章では、既存のベアメタル・サンプル・アプリケーション・プロジェクトを DS-5 にインポートしてデバッグ／実行する方法について説明します。

### 4-1. SoC EDS に付属のサンプル・アプリケーションの紹介（参考）

SoC EDS をインストールすると、SoC EDS に付属しているサンプル・アプリケーションが下記のフォルダに格納されます。

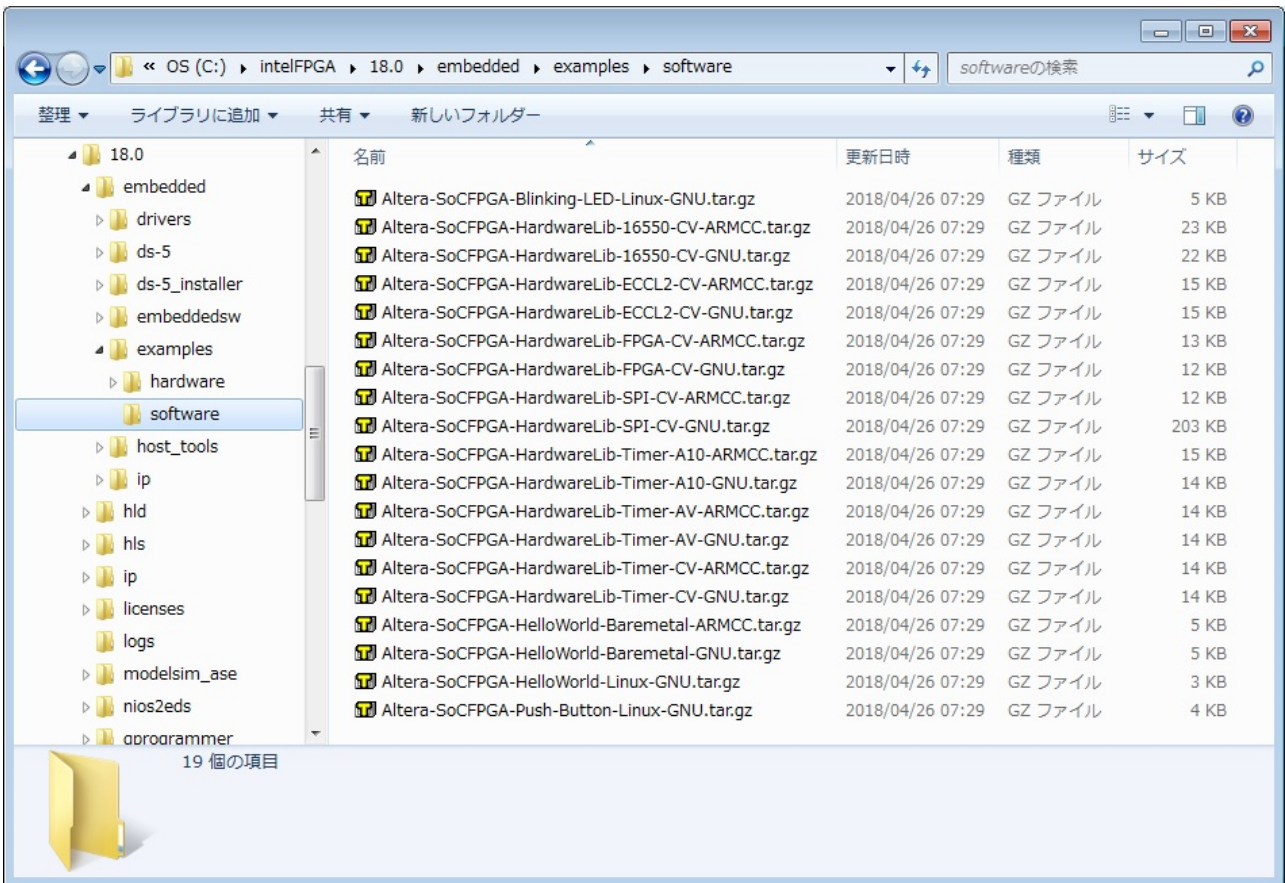
<intel FPGA installation directory>\¥embedded¥examples¥software

例) C:\¥intelFPGA¥18.0¥embedded¥examples¥software

これらのサンプル・アプリケーションは、DS-5 Intel® SoC FPGA Edition からインポートしてビルドおよびデバッグすることが可能です。

これらのファイルの中で、名前が “Altera-SoCFPGA-HardwareLib-” や “Altera-SoCFPGA-HelloWorld-Baremetal-” となっているものが、ベアメタル対応アプリケーションです。

また、名前に “GNU” と付いているものは GNU コンパイラ (GCC) 版、“ARMCC” と付いているものは Arm コンパイラ (ARMCC) 版となります。



【図 4-1】 SoC EDS 付属のサンプル・アプリケーション

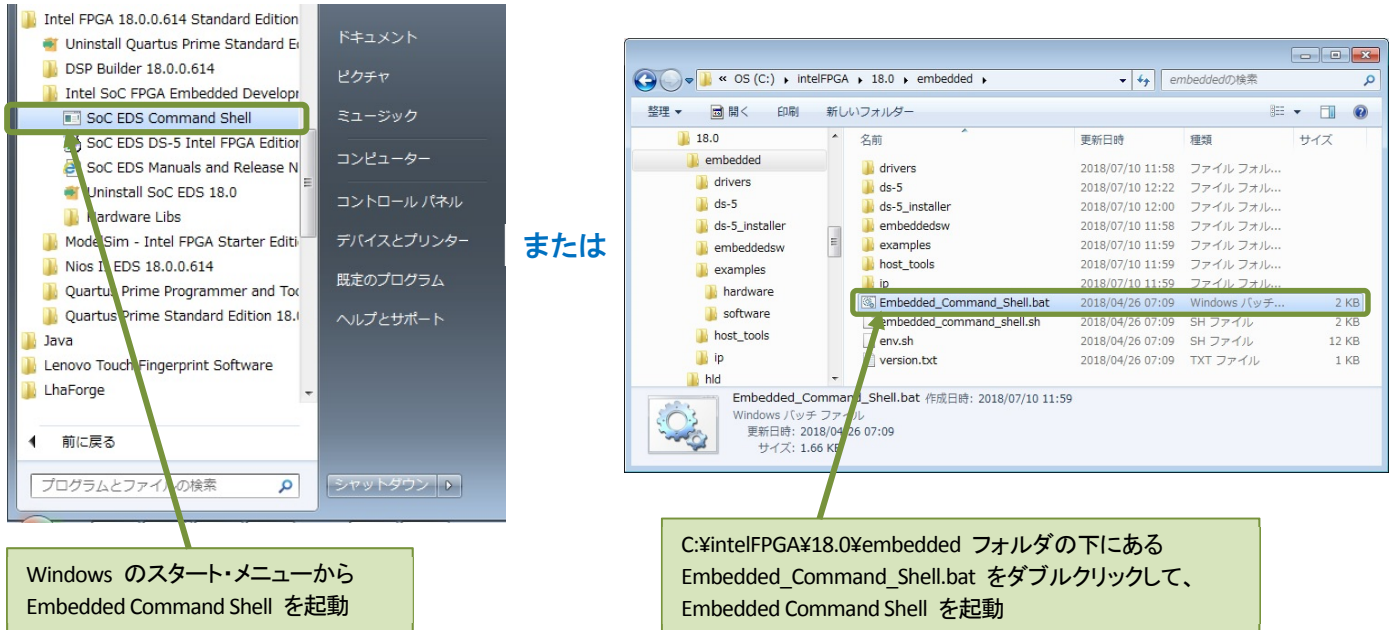
## 4-2. DS-5 の起動

SoC EDS に含まれている DS-5 Intel® SoC FPGA Edition を起動します。

SoC EDS に対する各種環境設定を自動的に実施するために、DS-5 は次の Embedded Command Shell から起動してください。

### 4-2-1. Embedded Command Shell の起動

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（embedded フォルダ）下に格納されている起動用スクリプトを実行し、Embedded Command Shell を起動します。



【図 4-2】 Embedded Command Shell の起動

### 4-2-2. DS-5 の起動

1. 下図のように Embedded Command Shell のウィンドウが開いたら `eclipse &` とコマンド入力して DS-5 Intel® SoC FPGA Edition を起動します。



【図 4-3】 DS-5 の起動

2. ワークスペース・フォルダの入力を求められます。ソフトウェア・プロジェクトのために固有のワークスペースを選択または作成します。

パスを指定して [OK] をクリックします（この例では、ワークスペースに C:¥Work¥DS-5 Workspace を指定しています。フォルダが存在しない場合は自動的に作成されます）。



【図 4-4】 DS-5 のワークスペースの指定

3. DS-5 ウェルカム画面が表示される場合は、[閉じる] (× マーク) をクリックします。

DS-5 ウェルカム画面は、ドキュメント、チュートリアルやビデオにアクセスするために使用することができます。



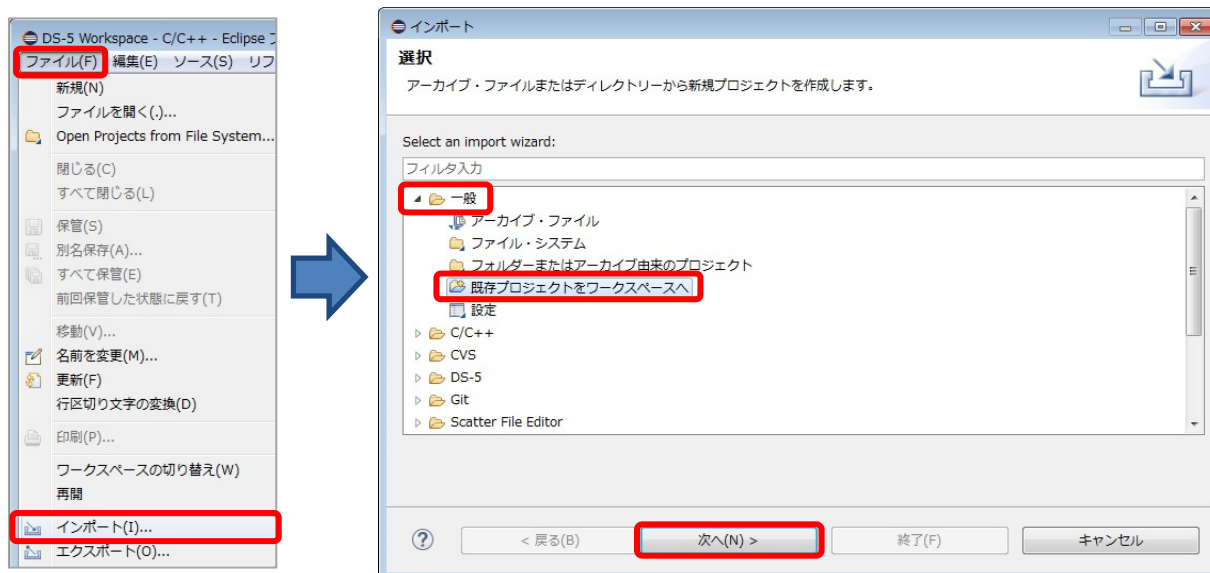
【図 4-5】 DS-5 ウェルカム画面



### 4-3. ベアメタル・サンプル・アプリケーションのインポート

この例では、事前に用意された Atlas-SoC / DE10-Nano ボード向け LED 点滅ベアメタル・サンプル・アプリケーション Atlas-Blinking-LED-Baremetal-GNU を DS-5 にインポートします。

- \_\_\_ 1. DS-5 のメニューから「ファイル(F)」 「インポート(I)...」を選択します。
- \_\_\_ 2. 「一般」 「既存プロジェクトをワークスペースへ」を選択し、[次へ(N)] をクリックします。

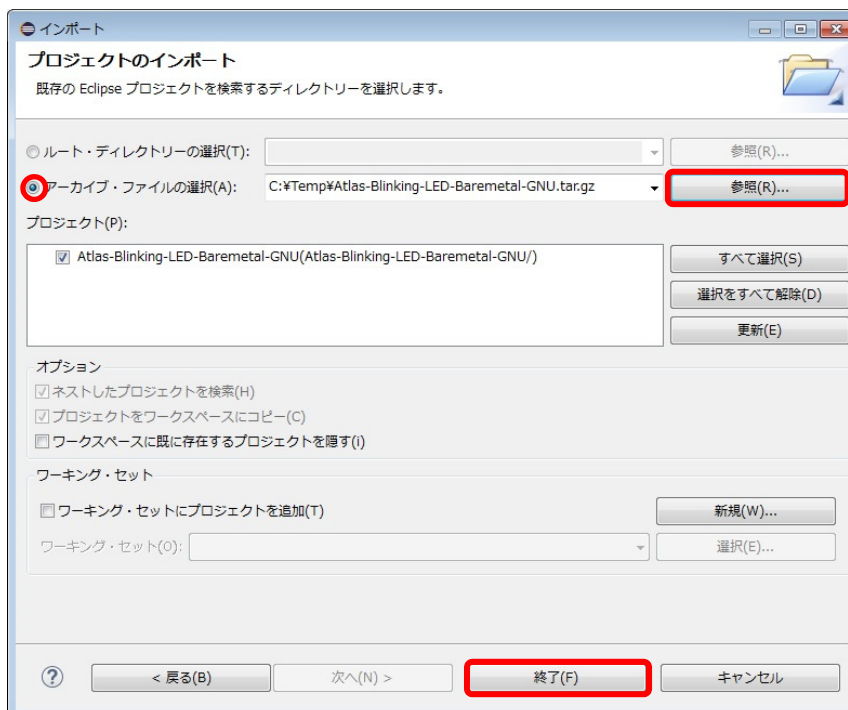


【図 4-6】 既存プロジェクトのインポート

- \_\_\_ 3. 「アーカイブ・ファイルの選択(A):」 オプションを選択します。[参照(R)] ボタンより、以下のサンプル・プロジェクトを指定します。Atlas-Blinking-LED-Baremetal-GNU.tar.gz 選択後、[終了(F)] ボタンを押します。

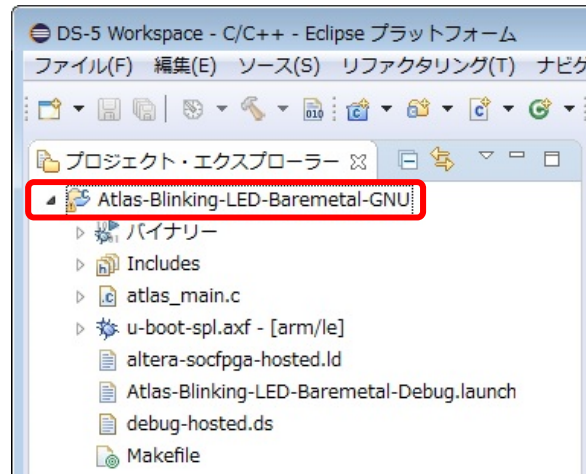
❗ **Note:**

本資料の説明では、Atlas-Blinking-LED-Baremetal-GNU.tar.gz を C:¥Temp に格納したものと説明しています。



【図 4-7】 サンプル・アプリケーションの選択

4. DS-5 画面左側のプロジェクト・エクスプローラーパネルにインポートしたベアメタル・サンプル・アプリケーション・プロジェクト **Atlas-Blinking-LED-Baremetal-GNU** が追加され、Atlas-Blinking-LED-Baremetal-GNU を展開すると、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 4-8】 インポートにより追加されたプロジェクト

#### 4-4. ベアメタル・サンプル・アプリケーションのビルド

次にインポートしたベアメタル・サンプル・アプリケーション・プロジェクトをビルドして実行できるようにします。

##### 4-4-1. Makefile の例（参考）

サンプル・アプリケーションの Makefile の例を下図に示します。

### ❗ Note:

ご利用のサンプル・アプリケーションにより、Makefile の記述内容が異なる場合があります。



```

36 ALT_DEVICE_FAMILY ?= soc_cv_av
37
38 SOCEDS_ROOT ?= $(SOCEDS_DEST_ROOT)
39 HWLIBS_ROOT = $(SOCEDS_ROOT)/ip/altera/hps/altera_hps/hwlib
40
41 HWLIBS_SRC := alt_clock_manager.c alt_generalpurpose_io.c alt_globaltmr.c alt_interrupt.c alt_timers.c alt_watchdog.c
42 EXAMPLE_SRC := hwlib.c
43 C_SRC := $(EXAMPLE_SRC) $(HWLIBS_SRC)
44
45 LINKER_SCRIPT := cycloneV-dk-ram-hosted.ld
46
47 MULTILIBFLAGS := -mcpu=cortex-a9 -mfloat-abi=softfp -mcpu=neon
48 CFLAGS := -g -O0 -Wall -Werror -std=c99 $(MULTILIBFLAGS) -I$(HWLIBS_ROOT)/include -I$(HWLIBS_ROOT)/include/$(ALT_DEVICE_FAMILY) -D$(ALT_DEVICE_FAMILY)
49 LDFLAGS := -T$(LINKER_SCRIPT) $(MULTILIBFLAGS)
50
51 CROSS_COMPILE := arm-altera-eabi-
52 CC := $(CROSS_COMPILE)gcc
53 LD := $(CROSS_COMPILE)g++
54 NM := $(CROSS_COMPILE)nm
55 OD := $(CROSS_COMPILE)objdump
56 OC := $(CROSS_COMPILE)objcopy
57
58 RM := rm -rf
59 CP := cp -f
60
61 ELF ?= $(basename $(firstword $(C_SRC))).axf
62 SPL := u-boot-spl.axf
63 OBJ := $(patsubst %.c,%.o,$(C_SRC))
64
65 .PHONY: all
67 all: $(ELF) $(SPL)
68
69 .PHONY: clean
70 clean:
71 $(RM) $(ELF) $(SPL) $(OBJ) *.objdump *.map $(HWLIBS_SRC)
72
73 define SET_HWLIBS_DEPENDENCIES
74 $(1): $(2)
75 $(CP) $(2) $(1)
76 endef
77
78 ALL_HWLIBS_SRC = $(wildcard $(HWLIBS_ROOT)/src/hwmgr/*.c) $(wildcard $(HWLIBS_ROOT)/src/hwmgr/$(ALT_DEVICE_FAMILY)/*.c)
79
80 $(foreach file,$(ALL_HWLIBS_SRC),$(eval $(call SET_HWLIBS_DEPENDENCIES,$(notdir $(file)),$(file))))
81
82 $(OBJ): %.o: %.c Makefile
83 $(CC) $(CFLAGS) -c $< -o $@
84
85 $(ELF): $(OBJ)
86 $(LD) $(LDFLAGS) $(OBJ) -o $@
87 $(OD) -d $@ > $@.objdump
88 $(NM) $@ > $@.map
89
90 $(SPL): $(SOCEDS_ROOT)/examples/hardware/cv_soc_devkit_ghrd/software/preloader/u-boot-socfpga/spl/u-boot-spl
91 $(CP) $< $@
92 $(OD) -d $@ > $@.objdump
  
```

【図 4-9】 Makefile の例

Makefile の中では主に次のものが定義されています。

- SoC FPGA デバイス・ファミリの指定
- SoC EDS のパス設定
- ビルド対象のソース・ファイルの指定
- リンカ・ファイルの指定
- クロス・コンパイラの指定
- HWLib ソース・ファイルのコピー
- Preloader 実行可能バイナリ (u-boot-spl.axf) のコピー

### ⚠ 注記:

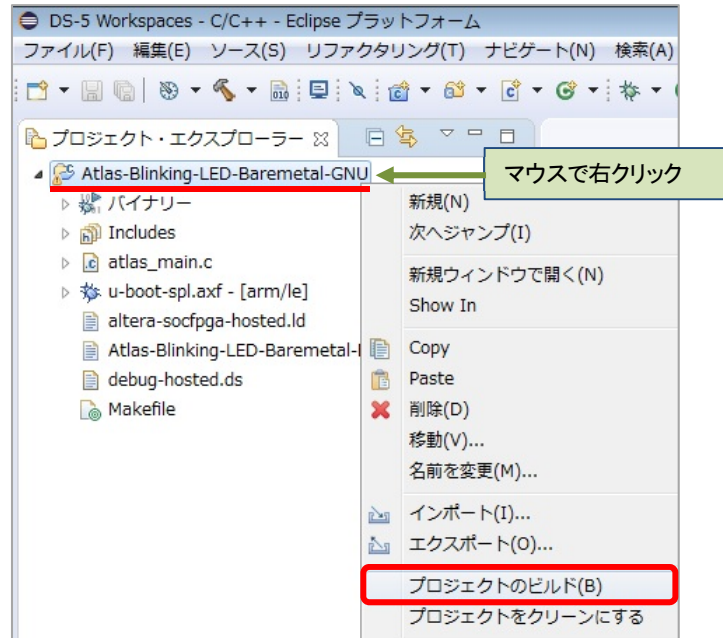
クロス・コンパイラの指定は、ご使用の SoC EDS のバージョンにより以下のように異なりますのでご注意ください。

- ・ v13.1 までは、CROSS\_COMPILE := arm-none-eabi-
- ・ v14.0 からは、CROSS\_COMPILE := arm-altera-eabi-

#### 4-4-2. プロジェクトのビルド

DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクト）をハイライトし、右クリックして「プロジェクトのビルド(B)」を実行します。

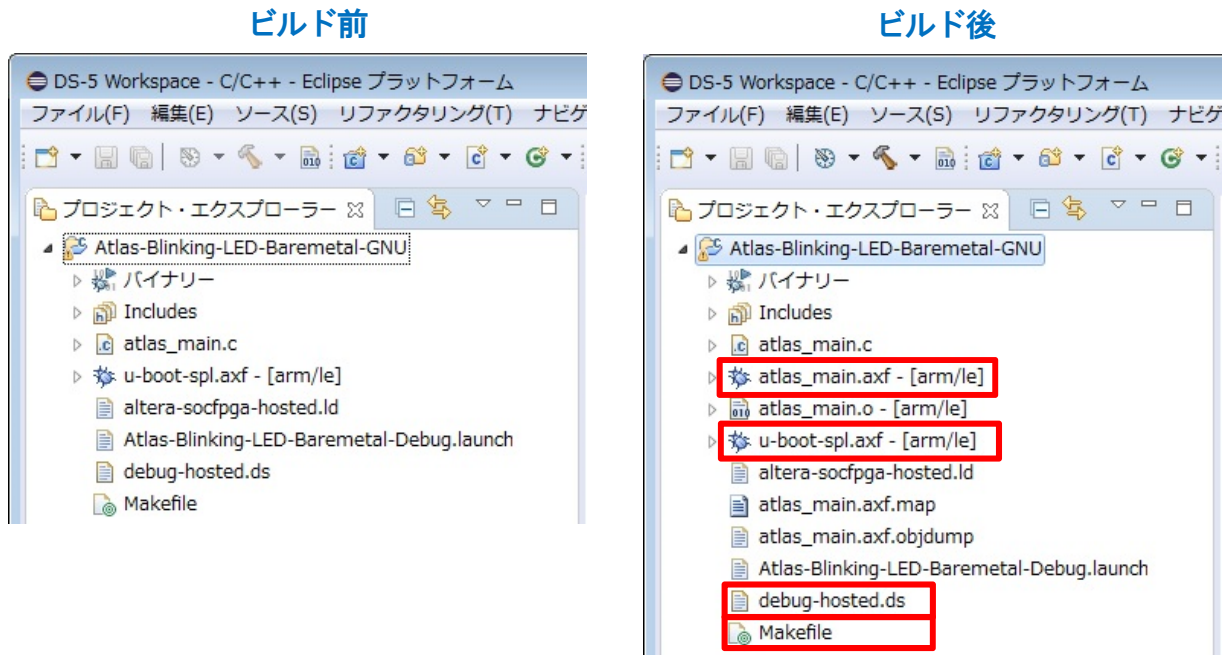
ビルドが完了すると、ベアメタル・アプリケーションの **.axf ファイル**（この例では、atlas\_main.axf）が生成されます。



【図 4-10】 プロジェクトのビルド

#### 4-4-3. ベアメタル・サンプル・アプリケーション・プロジェクトのファイル構成

サンプル・アプリケーションのファイル構成例を下図に示します。



【図 4-11】 サンプル・アプリケーションのファイル構成例

この中で特に重要なファイルとしては次のものがあります。

- **Makefile** : プロジェクトをビルドする際の指示書
- **debug-hosted.ds** : デバッグ時に実行するスクリプト・ファイル
- **<アプリケーション名>.axf** : サンプル・アプリケーションの実行可能バイナリ
- **u-boot-spl.axf** : Preloader の実行可能バイナリ

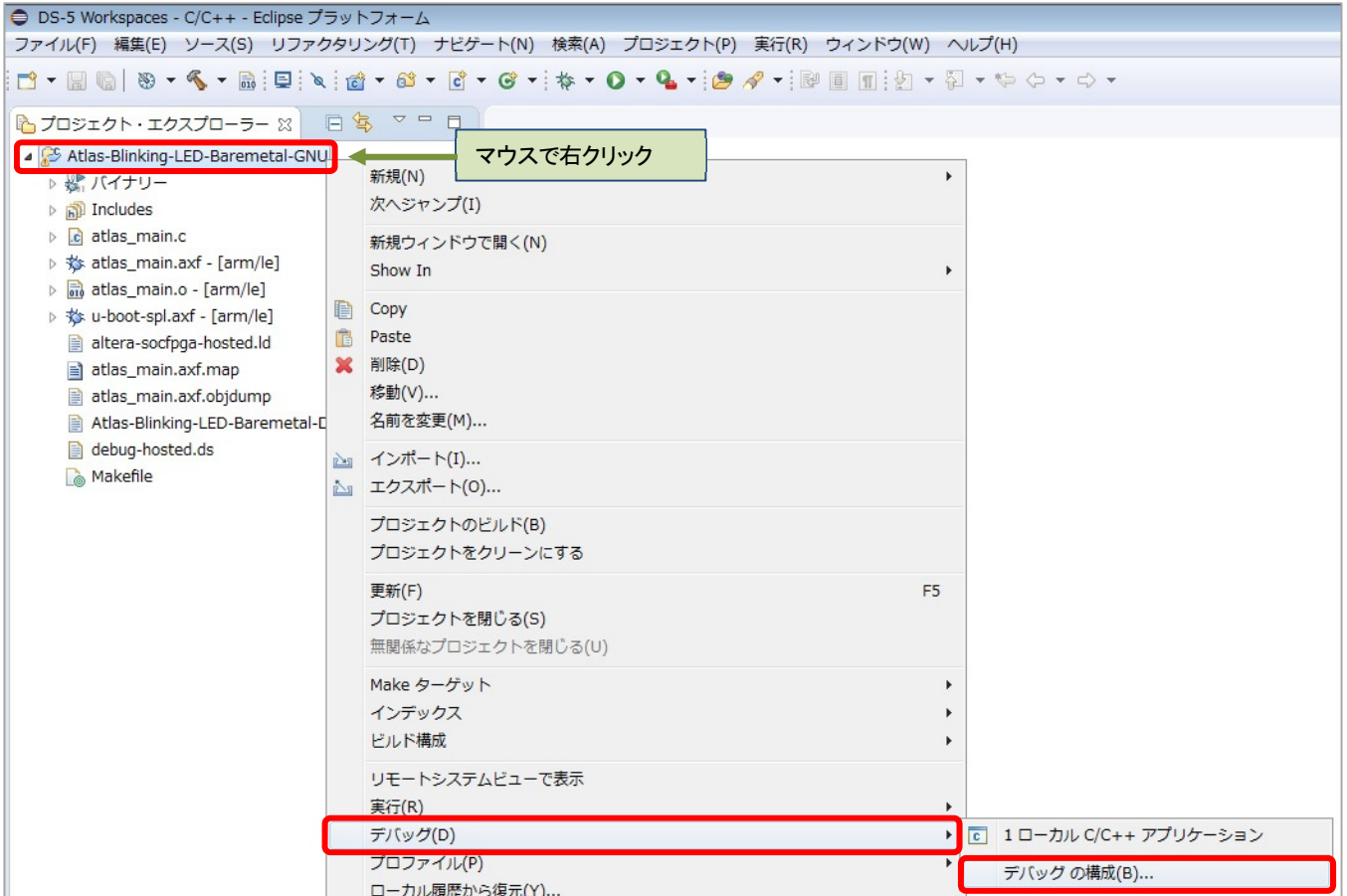
## 4-5. ベアメタル・サンプル・アプリケーションのデバッグ

次にビルドしたベアメタル・サンプル・アプリケーションをデバッグします。

デバッグを実行する前に「2. 事前準備」が完了していることを確認してください。

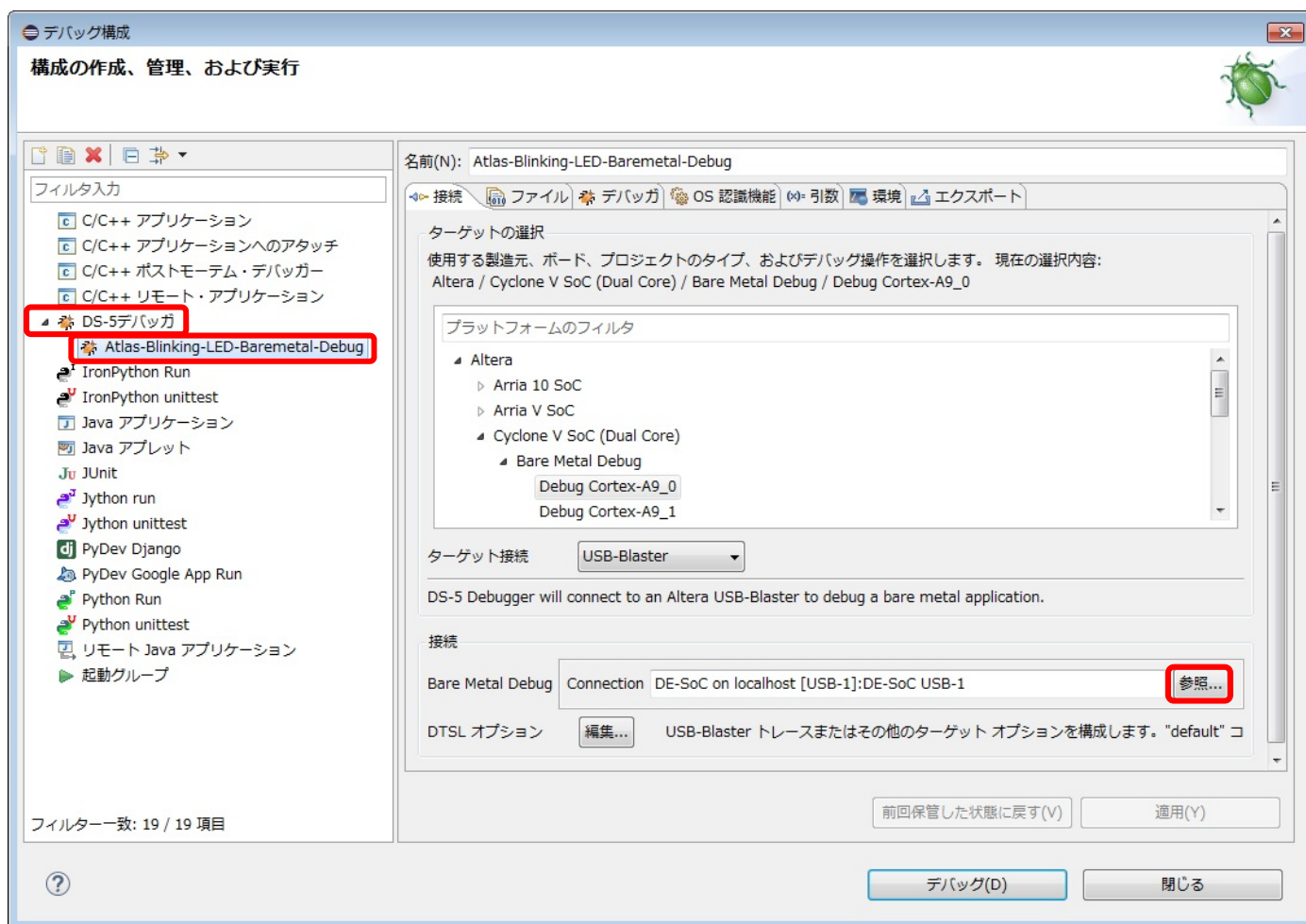
### 4-5-1. デバッグの実行

1. DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクト）をハイライトし、右クリックして「**デバッグ(D)**」⇒「**デバッグの構成(B)**」を選択します。



【図 4-12】「デバッグ(D)」 「デバッグの構成(B)」を選択

2. デバッグ構成ウィンドウにある左側のパネルから、「DS-5 デバッガ」 「Atlas-Blinking-LED-Baremetal-Debug」を選択します（表示されない場合は、DS-5 デバッガの横にある (+) 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、「Altera」 「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。
3. 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。



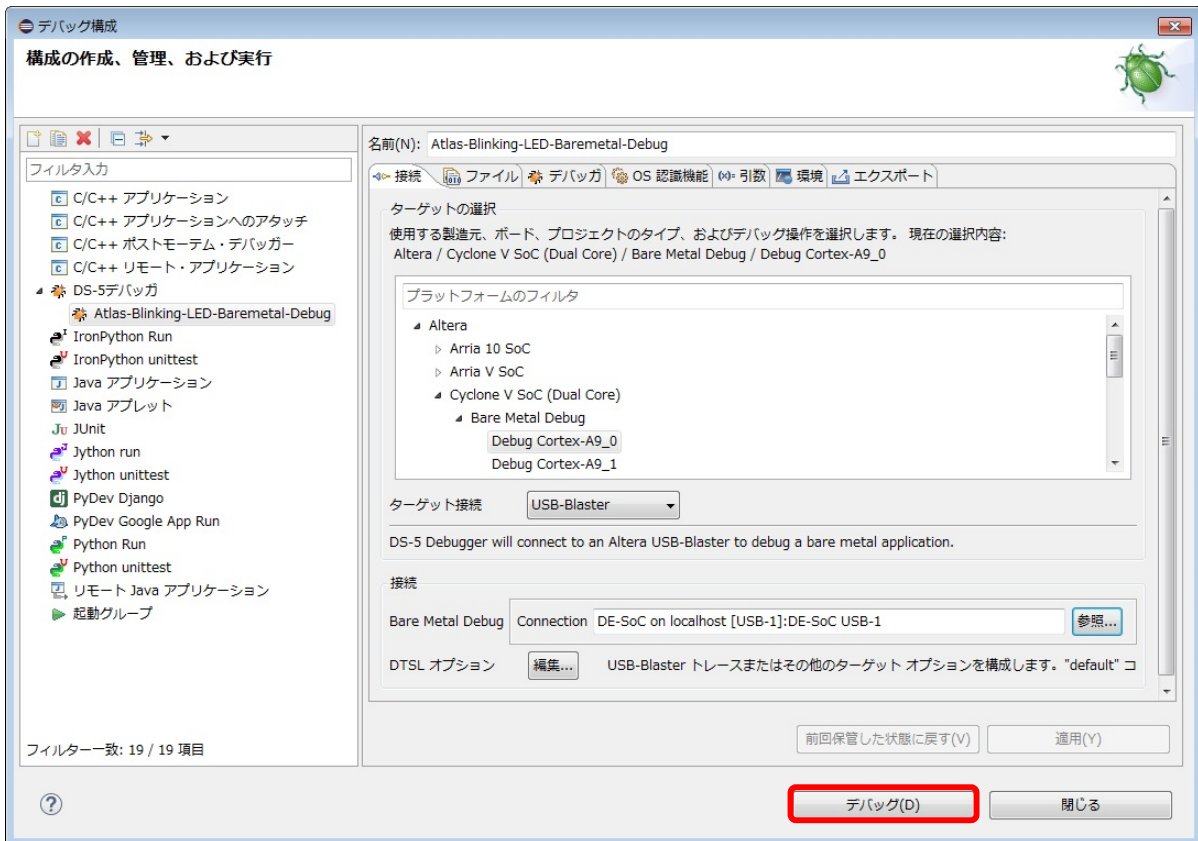
【図 4-13】 デバッグの構成

4. 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。



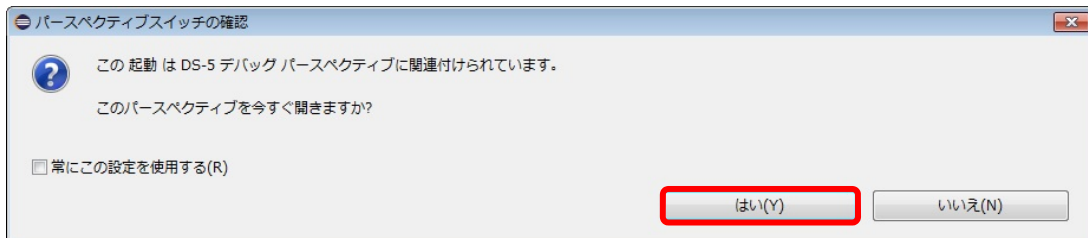
【図 4-14】 デバッグ・ケーブルの選択

5. デバッグ構成ウィンドウの右下にある **[デバッグ(D)]** ボタンをクリックします。



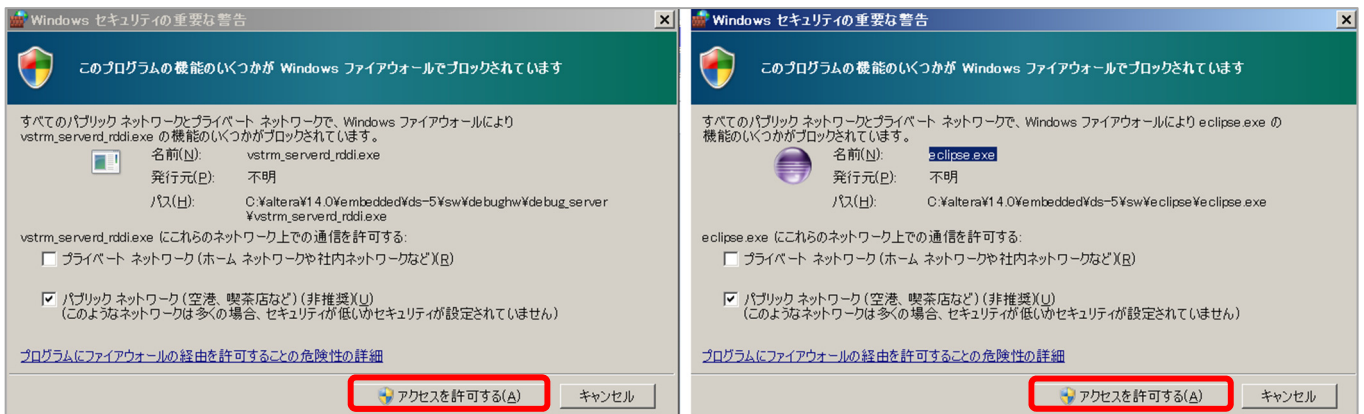
【図 4-15】 デバッグの実行

6. Eclipse は、デバッグ パースペクティブに切り替えるかどうかを尋ねます。 **[はい(Y)]** をクリックしてそれを受け入れてください。



【図 4-16】 パースペクティブスイッチの確認

Windows 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 4-17】 セキュリティの警告




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) DS-5 のライセンスが紐づけられているネットワーク・インターフェース（例えば USB-Ethernet Interface アダプタ）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。

デバッグは起動スクリプトの指示に従いセミホスティング機能を有効にした後、JTAG を経由してアプリケーションをボードにダウンロードします。

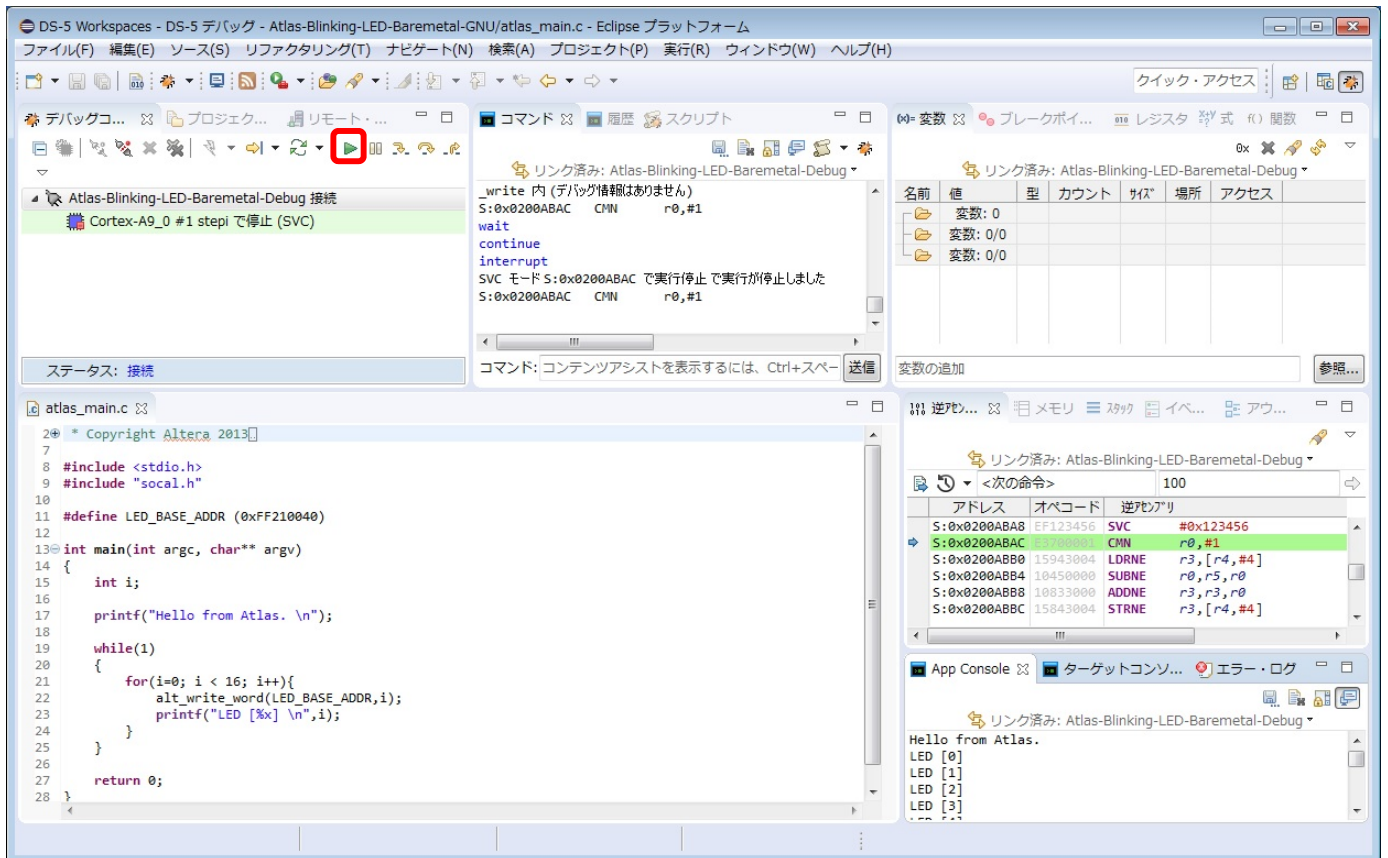
プログラム・カウンタ が main 関数に到達するとブレークされデバッグが開始できる状態となります。

この段階では、DS-5 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）。



\_\_\_ 7. 緑色の「**続行**」  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 **Hello from Atlas.** メッセージが表示されます。

更に **LED [0] ... LED [f]** メッセージが表示され、Atlas-SoC ボード上のユーザ LED（LED [3:0]）の点灯状態が変化することを確認します。



【図 4-18】 アプリケーションの実行/デバッグ

\_\_\_ 8. 「**ターゲットから切断**」  ボタンをクリックして CPU との接続を切断し、「**すべての接続の削除**」  ボタンをクリックしてターゲットを削除します。

\_\_\_ 9. メニュー・バーのショートカット・ボタン  をクリックして元の C/C++ パースペクティブに切り替えます。

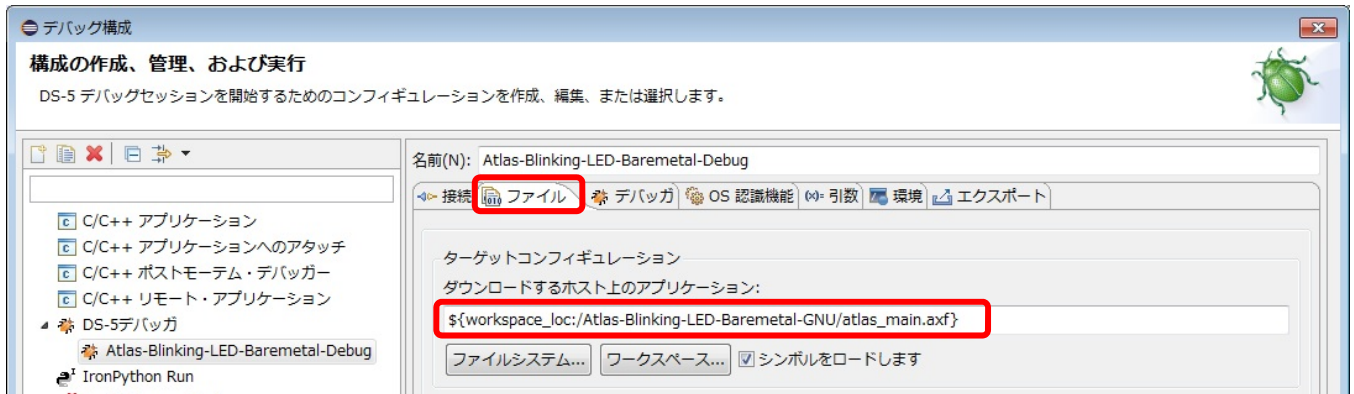
#### 4-5-2. デバッグの構成（参考）

Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクトに含まれる、デバッグ構成におけるその他の設定を以下に紹介します。

##### ① 「ファイル」タブ

ターゲット・ボードにダウンロードするアプリケーション実行ファイルを指定しています。

このサンプルでは、アプリケーション実行ファイルに atlas\_main.axf を指定しています。

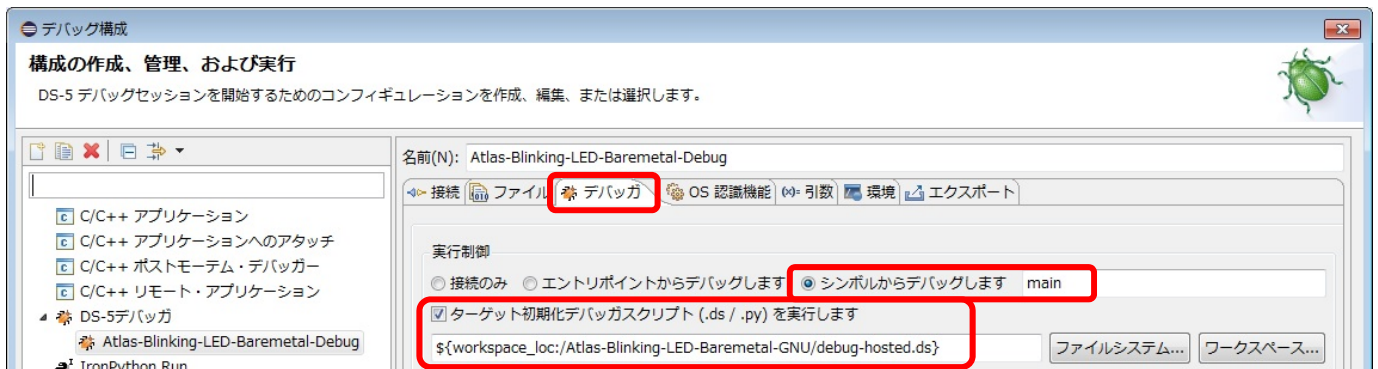


【図 4-19】 デバッグ構成ウィンドウの「ファイル」タブ

##### ② 「デバッガ」タブ

デバッグの開始方法や、デバッガ・スクリプト・ファイルを指定しています。

このサンプルでは、main 関数からデバッグを開始し、デバッガ・スクリプト・ファイルとして debug-hosted.ds を指定しています。



【図 4-20】 デバッグ構成ウィンドウの「デバッガ」タブ

### 4-5-3. デバッガ・スクリプト・ファイルでの実行内容（参考）

DS-5 では、デバッグ・コマンドを含むデバッグ・スクリプト・ファイルを使用することにより、デバッグ操作を自動化することができます。

デバッグ・スクリプト・ファイルでは、アプリケーションを実行する前に Preloader を実行して、HPS I/O ピンや SDRAM コントローラの初期化設定を行い、アプリケーションをロードして実行するような一連の操作がデバッグ・コマンドで書かれています。

サンプル・アプリケーションにおけるデバッグ・スクリプト・ファイルでの実行内容の例を下図に示します。

#### ❗ Note:

ご利用のサンプル・アプリケーションにより、デバッグ・スクリプト・ファイルの記述内容が異なる場合があります。

```

debug-hosted.ds
1 #
2 # Reset and stop the system.
3 #
4 reset system
5 wait 30s
6 stop
7 wait 30s
8 set var $Core::$R0 = 0
9
10 #
11 # Disable semihosting.
12 #
13 set semihosting enabled false
14
15 #
16 # Load the SPL preloader into memory.
17 #
18 loadfile "$sdir/u-boot-spl.axf" 0x0
19
20 #
21 # Enable semihosting.
22 #
23 set semihosting enabled true
24
25 #
26 # Delete any existing breakpoints.
27 #
28 delete
29
30 #
31 # Set a breakpoint in the SPL function spl_boot_device(). This
32 # is called right before the SPL tries to load the next stage in
33 #
34 tbreak spl_boot_device
35
36 #
37 # Set the PC to the entry point and go.
38 #
39 run
40
41 #
42 # Wait for the breakpoint.
43 #
44 wait
45
46 #
47 # Load the demo program.
48 #
49 loadfile "$sdir/helio_main.axf"
50
51 #
52 # Run the target and break at main().
53 #
54 start
  
```

### HPS のブートシーケンス



ブート ROM

Preloader

アプリケーション

Platform Designer HPS の設定から  
 ・ HPS I/O の初期化  
 ・ SDRAM コントローラ  
 など動作するために必要な処理を実行

デバッガ・スクリプトでは HPS の初期化  
 に必要な処理を実施するところ  
 (spl\_boot\_device) で一旦ブレークされ、  
 アプリケーションのロードに移行

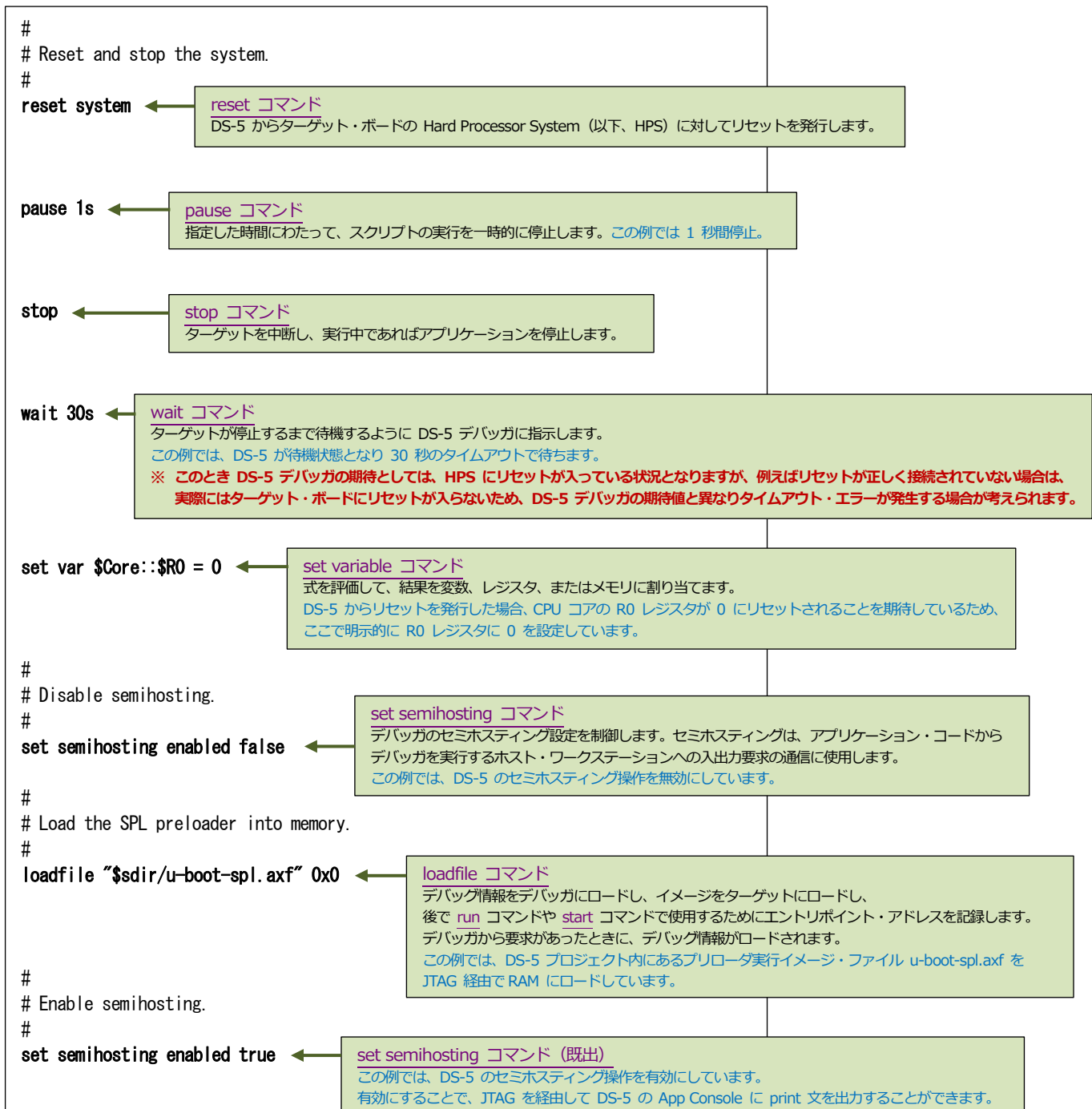
【図 4-21】 デバッガ・スクリプト・ファイルでの実行内容の例

DS-5 デバッガ・スクリプト・ファイルは、テキスト・ベースのファイルです。

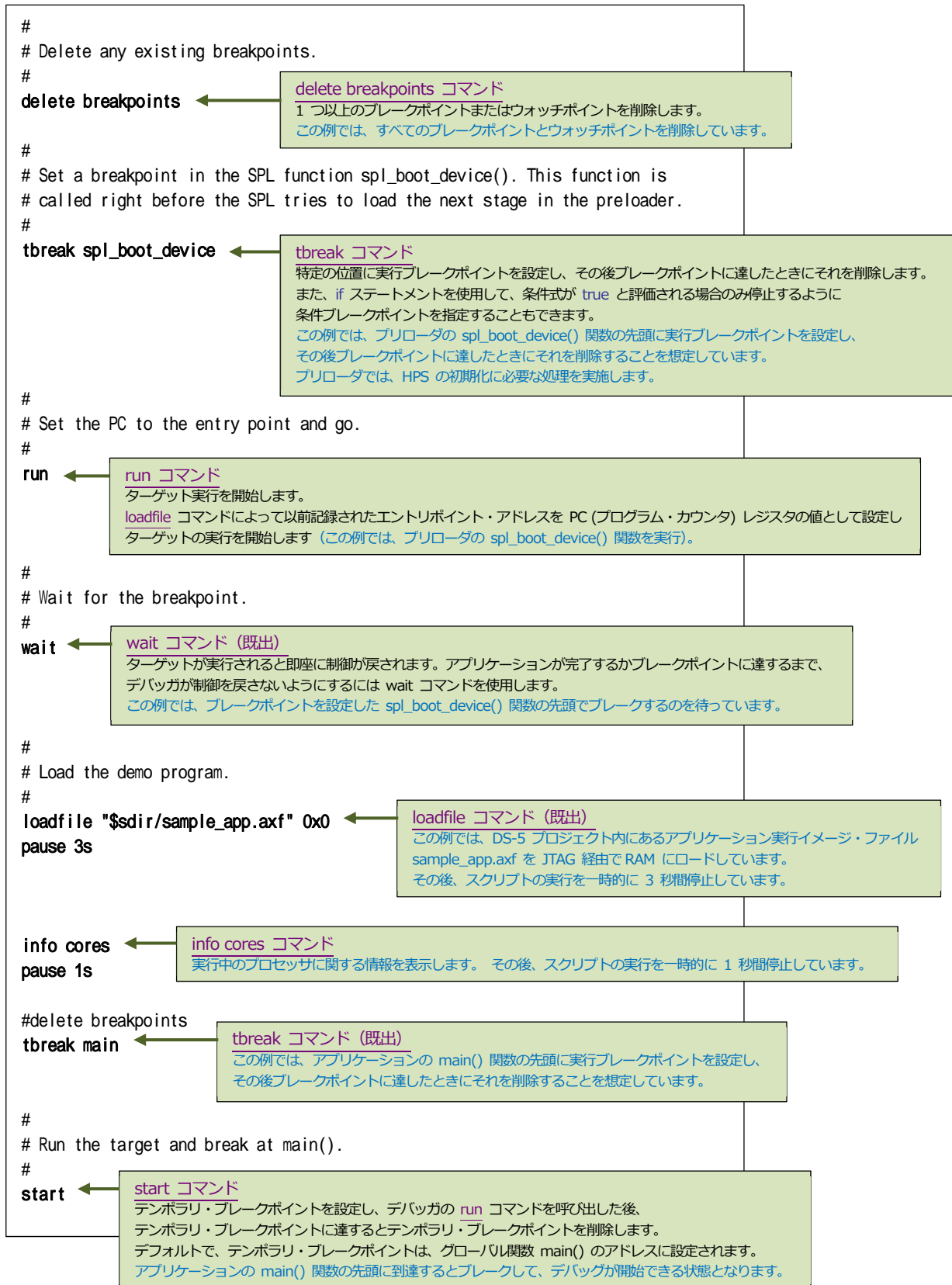
DS-5 デバッガ・スクリプト・ファイルを作成する際の注意事項としては以下が挙げられます。

- ファイル拡張子には .ds を使用することになっています。
- 1 行につき 1 つのコマンドのみを含める必要があります。
- コマンドでは大文字と小文字が区別されません。
- 必要に応じて、# を文字の先頭に付けることでコメント文を入れることもできます。

一般的な、DS-5 デバッガ・スクリプト・ファイルでは、以下のようなデバッグ・コマンドが記述されています。



【図 4-22】 デバッガ・スクリプト・ファイル内のデバッグ・コマンド例 (1)



【図 4-23】 デバッガ・スクリプト・ファイル内のデバッグ・コマンド例 (2)

 **参考:**

DS-5 でのデバッグ・スクリプト・ファイルおよびデバッグ・コマンドに関する更に詳しい情報は、以下のページが参考になります。

- 『DS-5 活用テクニック ～ デバッグ・コマンドの使い方』  
<https://service.macnica.co.jp/library/129405>
- DS-5 デバッグ・ユーザガイド  
『Arm DS-5 Debugger User Guide』 (最新英語版)  
『ARM® DS-5 デバッグユーザガイド バージョン 5.26』 (日本語版)
- DS-5 デバッグ・コマンド・リファレンス  
『Arm DS-5 Debugger Command Reference』 (最新英語版)  
『ARM® DS-5 デバッグコマンドリファレンス バージョン 5.26』 (日本語版)

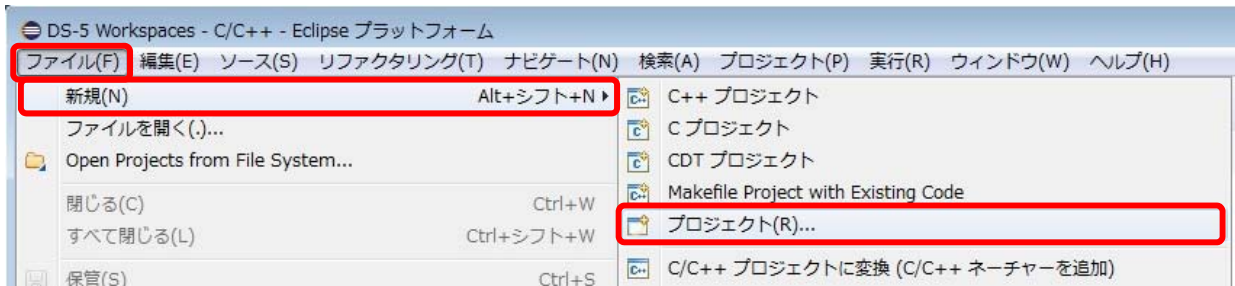
## 5. 新規にベアメタル・アプリケーションを作成して DS-5 で実行する方法

この章では、ベアメタル・アプリケーション・プロジェクトを新規に作成して DS-5 でデバッグ / 実行する方法について説明します。

DS-5 が起動していない場合は、前出の「4-2. DS-5 の起動」の手順に従って DS-5 を起動しておきます。

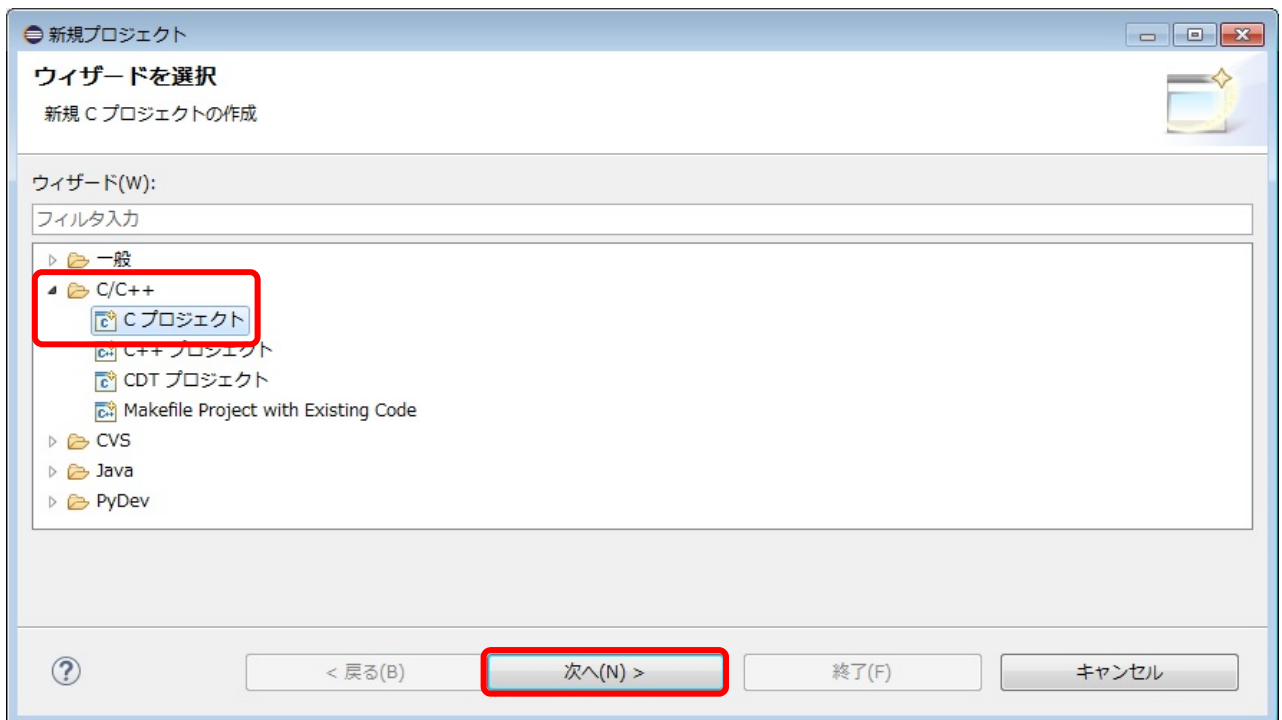
### 5-1. ベアメタル・アプリケーションの新規作成

- \_\_\_ 1. DS-5 のメニューより「**ファイル(F)**」⇒「**新規(N)**」⇒「**プロジェクト(R)...**」を実行します。



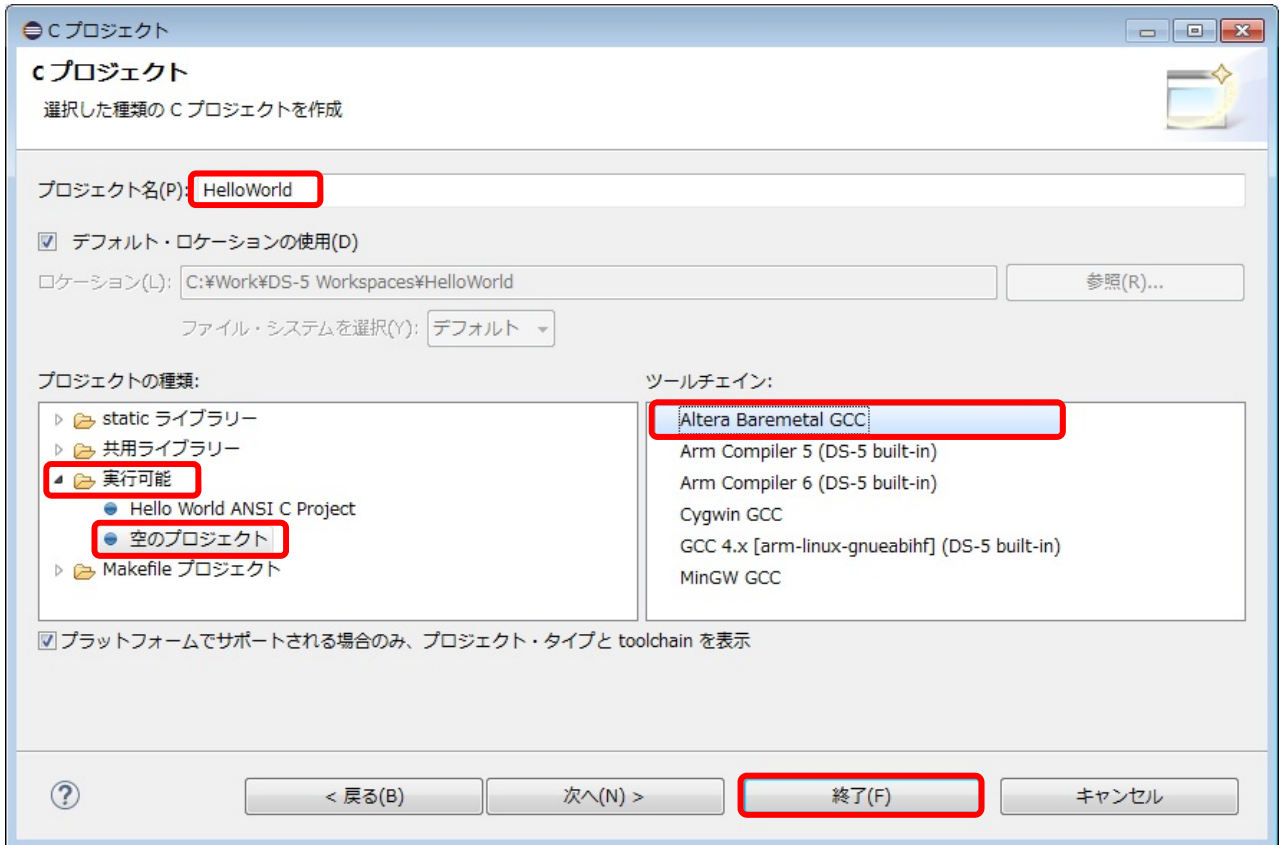
【図 5-1】「ファイル(F)」⇒「新規(N)」⇒「プロジェクト(R)…」を実行

- \_\_\_ 2. 新規プロジェクト画面にて「**C/C++**」⇒「**C Project**」を選択して「**次へ(N)**」をクリックします。



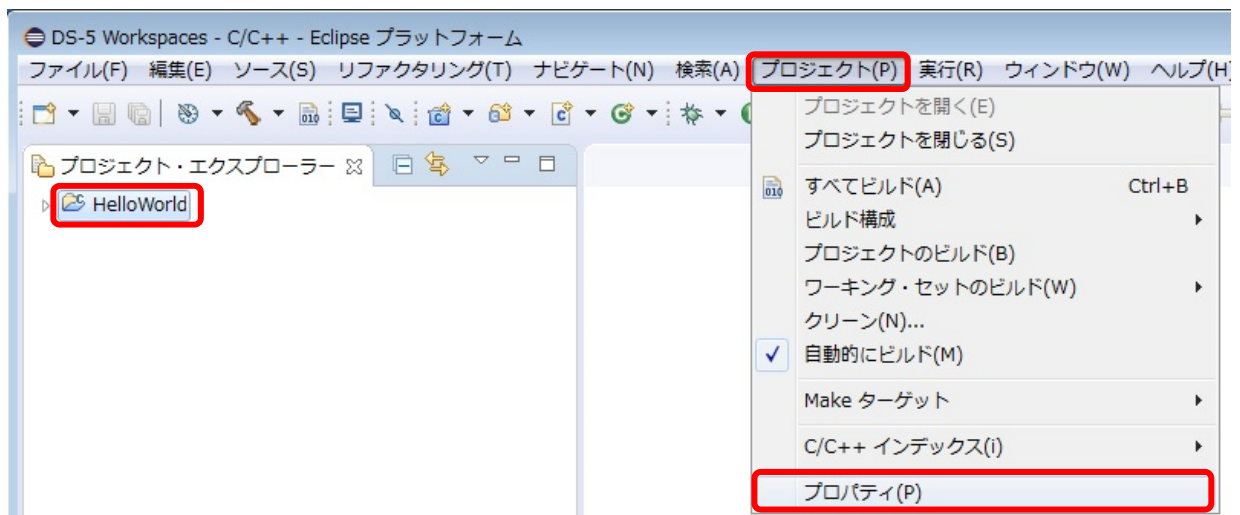
【図 5-2】新規プロジェクト画面にて「C/C++」⇒「C Project」を選択

- \_\_\_ 3. プロジェクト名(P): には「HelloWorld」を指定します。
- \_\_\_ 4. プロジェクトの種類: には「**実行可能**」 「**空のプロジェクト**」を選択します。
- \_\_\_ 5. ツールチェーン: には「**Altera Baremetal GCC**」を選択します。
- \_\_\_ 6. **[終了(F)]** をクリックするとプロジェクトが生成されます。



【図 5-3】 HelloWorld プロジェクトの生成

- \_\_\_ 7. プロジェクト・エクスプローラ上の HelloWorld をハイライトさせた状態で、DS-5 のメニューより「**プロジェクト(P)**」 「**プロパティ(P)**」を起動します。



【図 5-4】 「プロジェクト(P)」 「プロパティ(P)」を起動



- \_\_\_ 8. プロパティ画面左側のツリーより「C/C++ ビルド」 「設定」を選択します。
- \_\_\_ 9. 設定画面の ツール設定 タブ上で「GCC C リンカ」 「イメージ」を選択し、リンカのスクリプト指定欄の右側にある [参照(B)...] ボタンをクリックします。
- \_\_\_ 10. 次の場所に用意したリンカ・スクリプト・ファイルを選択し、[開く(O)] をクリックします。  
**C:¥Temp¥cycloneV-dk-oc-ram-hosted.ld**

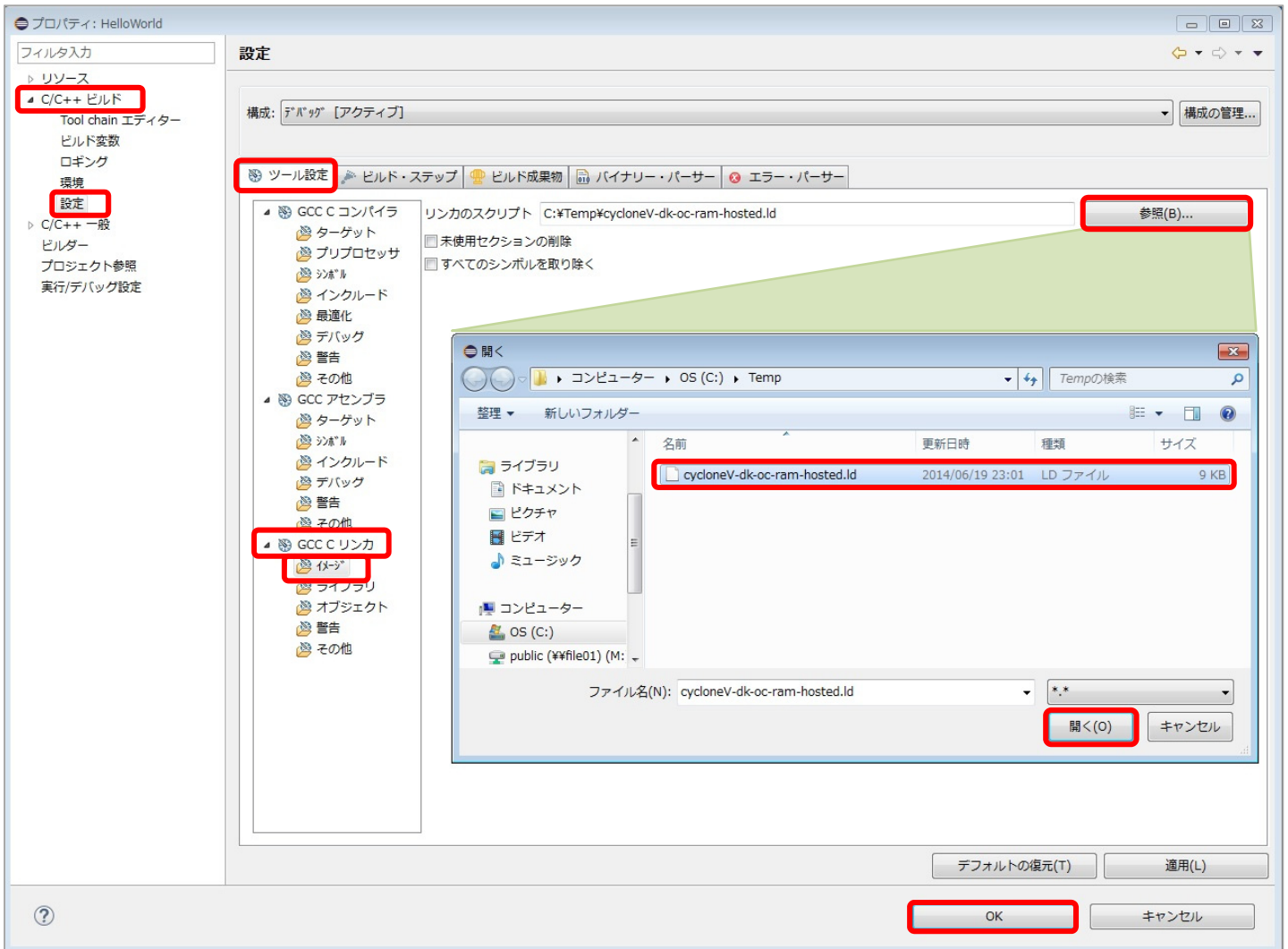
❗ **Note:**

本資料の説明では、cycloneV-dk-oc-ram-hosted.ld を C:¥Temp に格納したものとして説明しています。

このリンカ・スクリプトでは、64KB の内部 RAM (On-Chip RAM) をターゲットに指定するとともに、セミアステリングの利用をリンカへ指示します。

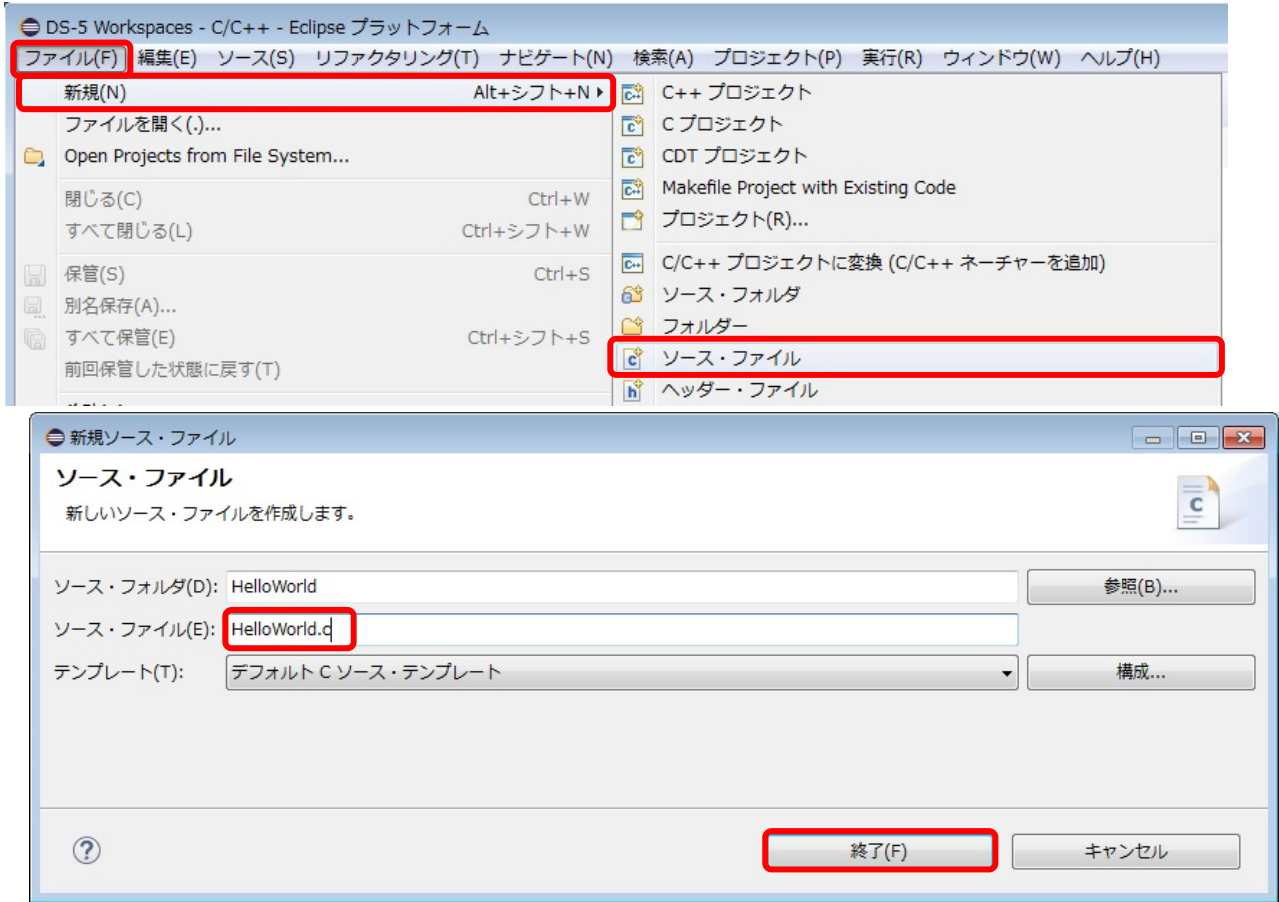
様々な領域を持ったリンカ・スクリプトのフォーマットを理解したい場合は、スクリプト・ファイルを開いて内容を参照してみてください。

- \_\_\_ 11. [OK] ボタンをクリックし、プロパティ画面を閉じます。



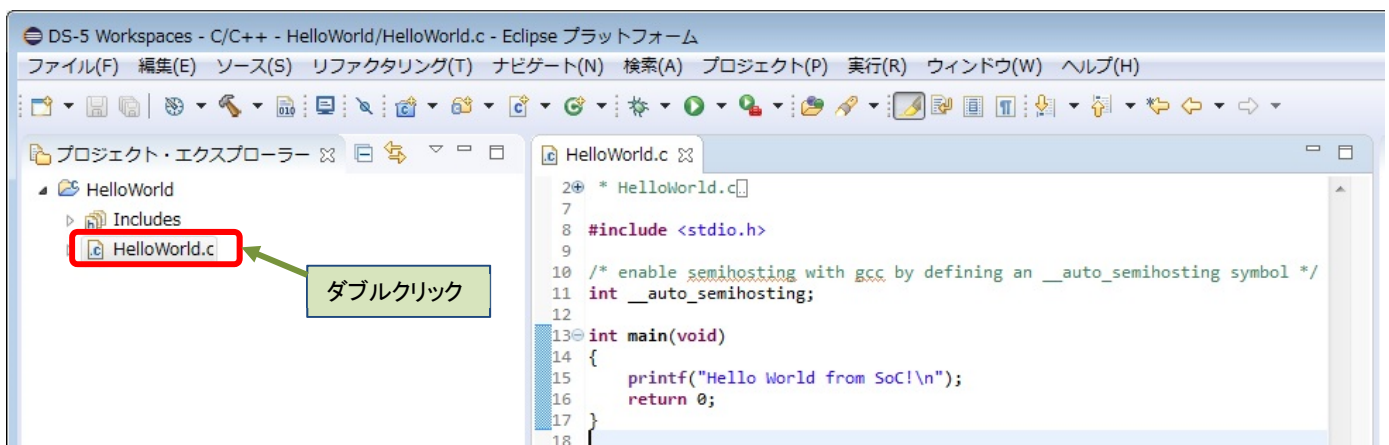
【図 5-5】 設定画面の ツール設定 タブ

- \_\_\_ 12. DS-5 のメニューより「ファイル(F)」 「新規(N)」 「ソース・ファイル」を実行します。
- \_\_\_ 13. ソース・ファイル(E): にファイル名「HelloWorld.c」と入力し「終了(F)」をクリックします。



【図 5-6】「ファイル(F)」 「新規(N)」 「ソース・ファイル」を実行

- \_\_\_ 14. HelloWorld.c をダブルクリックして開き、以下のようにプログラムをタイプ入力します。



【図 5-7】 HelloWorld.c にプログラムをタイプ入力

`__auto_semihosting` シンボルは、現在の実行可能イメージがセミホスティング・サービスを必要とすることを、デバッガへ伝える効果を持ちます。このシンボルは、2 つのアンダースコアで始まります。

15. 編集後の HelloWorld.c を保存します (Ctrl+S キー または「**ファイル(F)**」 「**保管(S)**」を実行)。

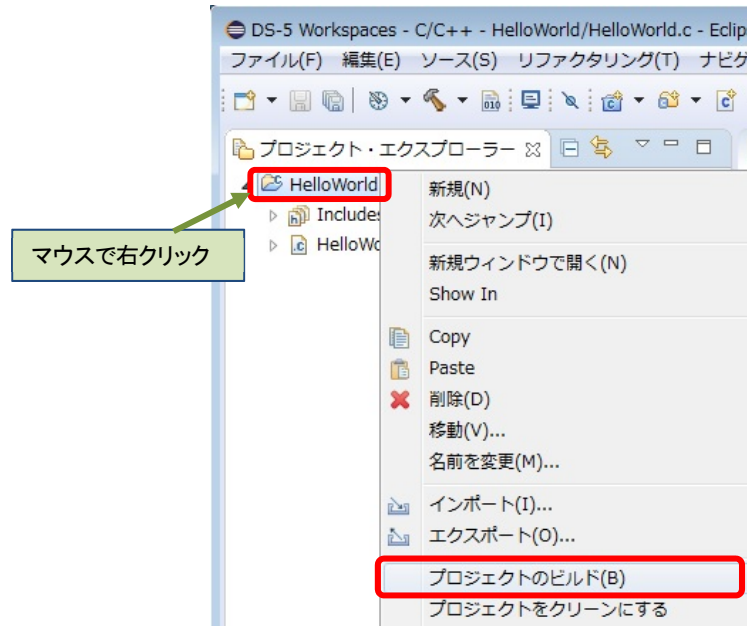


【図 5-8】 編集後の HelloWorld.c を保存

## 5-2. ベアメタル・アプリケーションのビルド

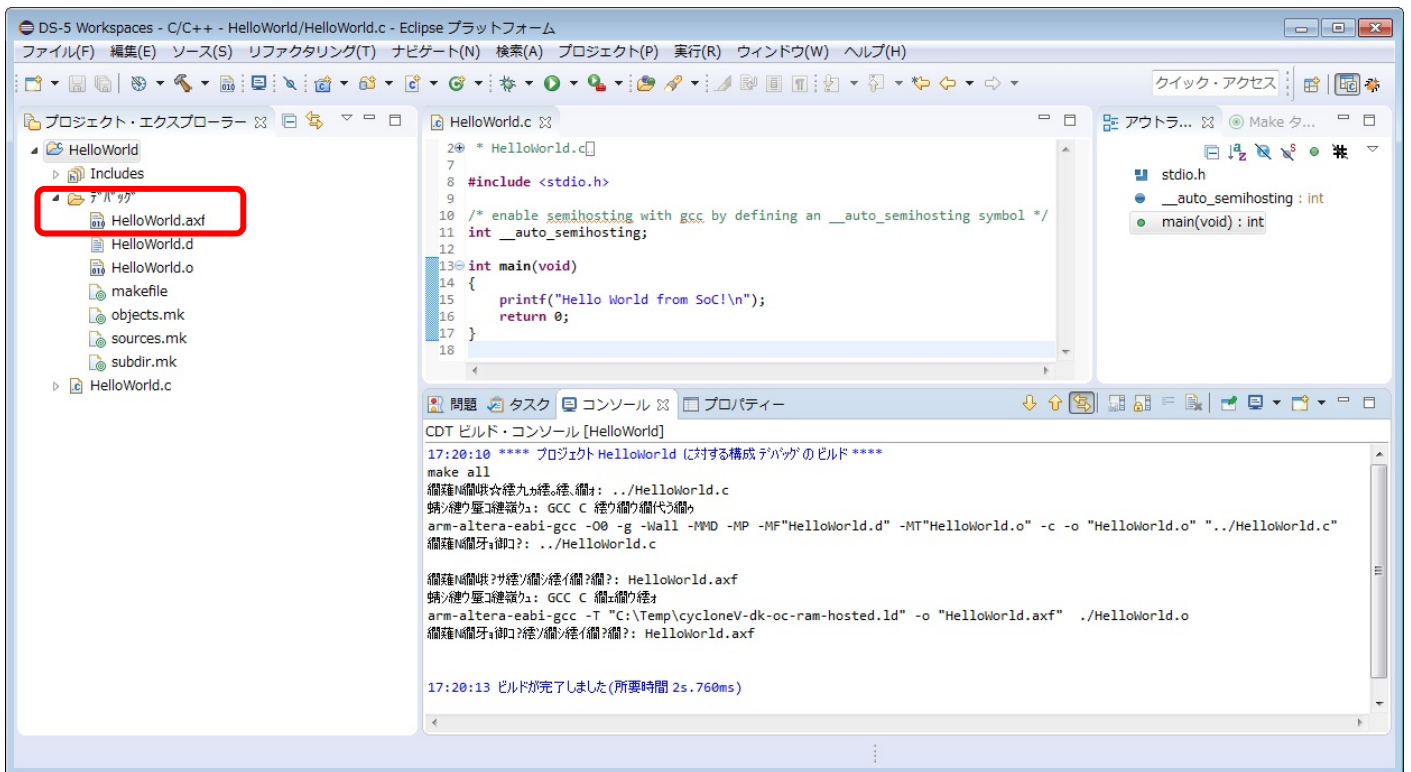
次に作成したベアメタル・アプリケーション・プロジェクトをビルドして実行できるようにします。

- \_\_\_ 1. HelloWorld プロジェクトをハイライトし、右クリックして「プロジェクトのビルド(B)」を実行します。



【図 5-9】 HelloWorld ベアメタル・アプリケーション・プロジェクトのビルド

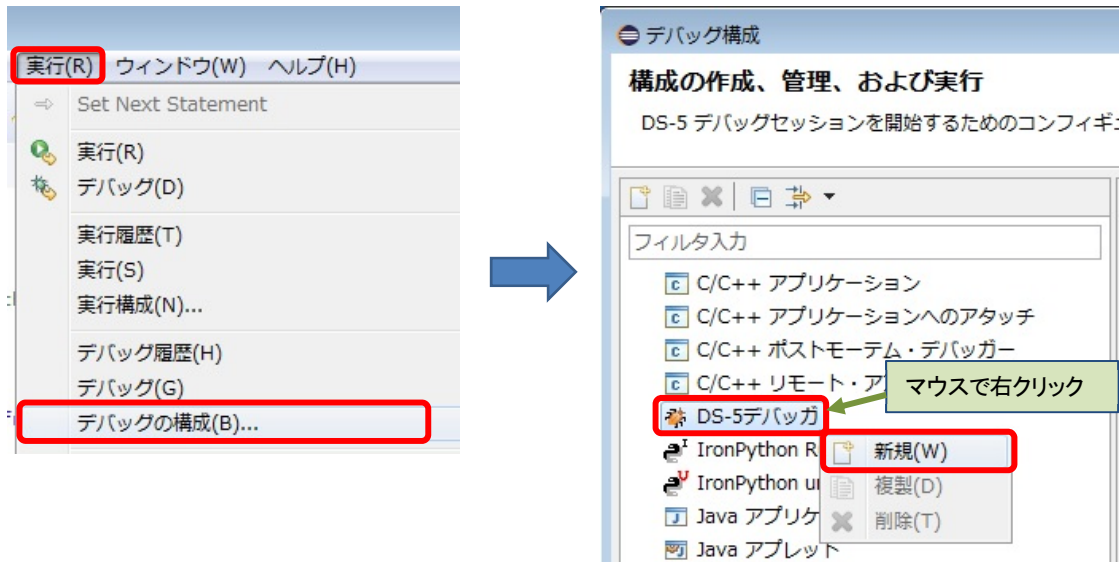
- \_\_\_ 2. コンソール・ビューにて、プロジェクトのビルドが完了した事を確認します。  
ビルドが完了すると、「デバッグ」フォルダの下に HelloWorld.axf ファイルが生成されます。



【図 5-10】 ビルドの完了

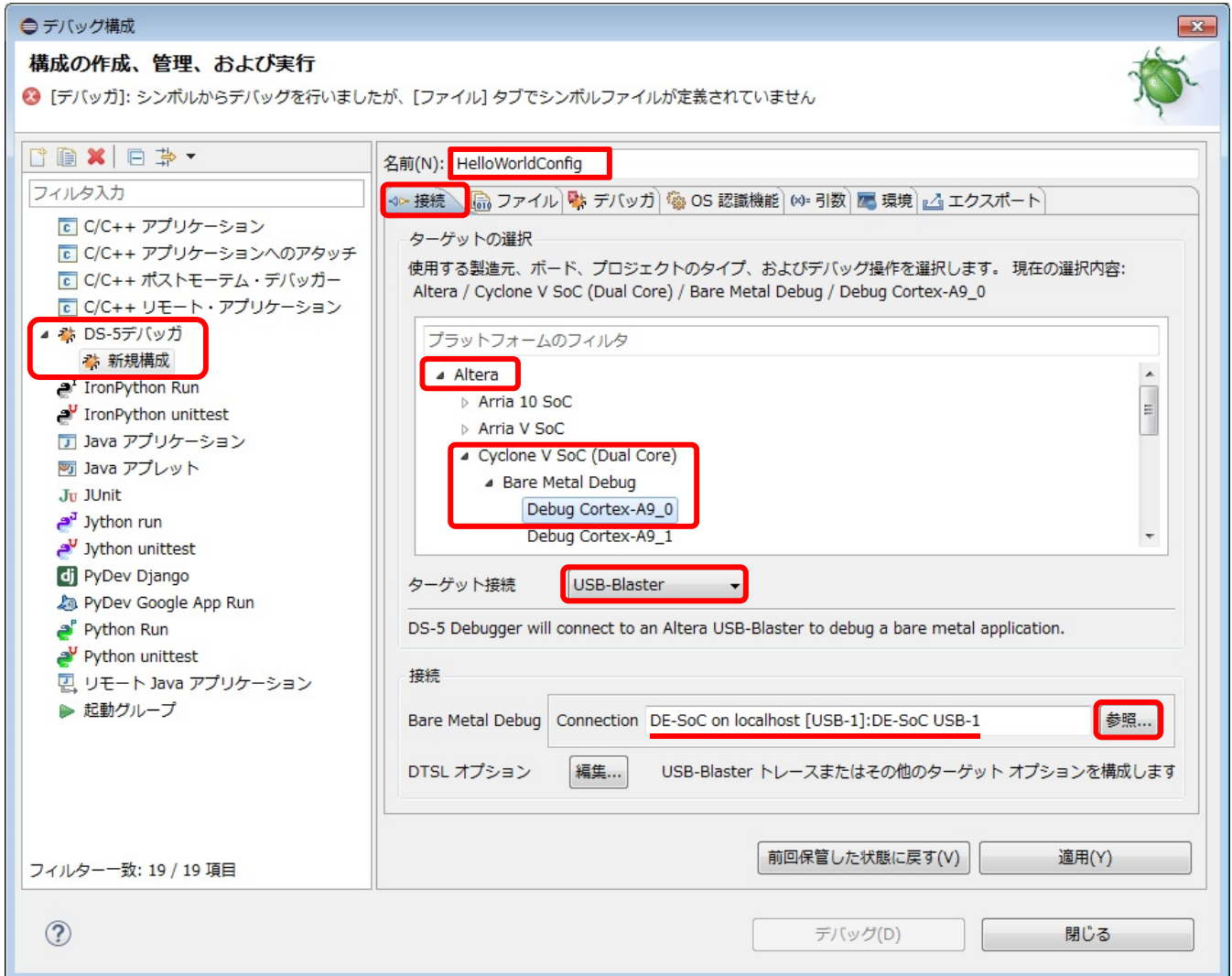
## 5-3. ベアメタル・アプリケーションのデバッグ

1. メニューの「**実行(R)**」 「**デバッグの構成(B)...**」を選択します。ここから、ターゲット・ボード上で Hello World アプリケーションを実行するための設定を行います。
2. デバッグ構成画面左側のリストより「**DS-5 デバッガ**」を選択し、**右クリック** して現れるメニューから「**新規(W)**」をクリックします。



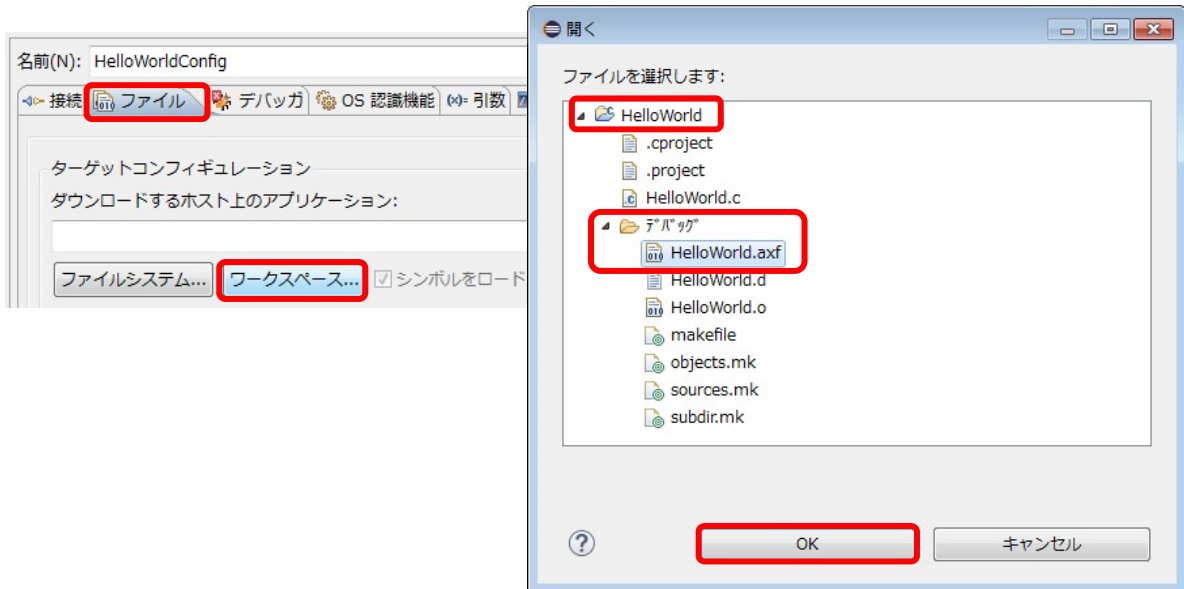
【図 5-11】 新規デバッグ構成の作成

- \_\_\_ 3. 名前(N): に「HelloWorldConfig」と指定します。
- \_\_\_ 4. 「接続」タブのターゲットに「Altera」「Cyclone V SoC (Dual Core)」「Bare Meta Debug」「Debug Cortex-A9\_0」を選択します。
- \_\_\_ 5. ターゲット接続に「USB-Blaster」を選択します。
- \_\_\_ 6. 接続の [参照...] ボタンをクリックし、接続ブラウザ画面にて「DE-SoC on localhost」を選択します。



【図 5-12】 新規デバッグ構成の「接続」タブの設定

7. 「ファイル」タブを開きます。
8. ダウンロードするホスト上のアプリケーション: の [ワークスペース...] ボタンをクリックし、「HelloWorld」 「デバッグ」 「HelloWorld.axf」を選択して、[OK] をクリックします。



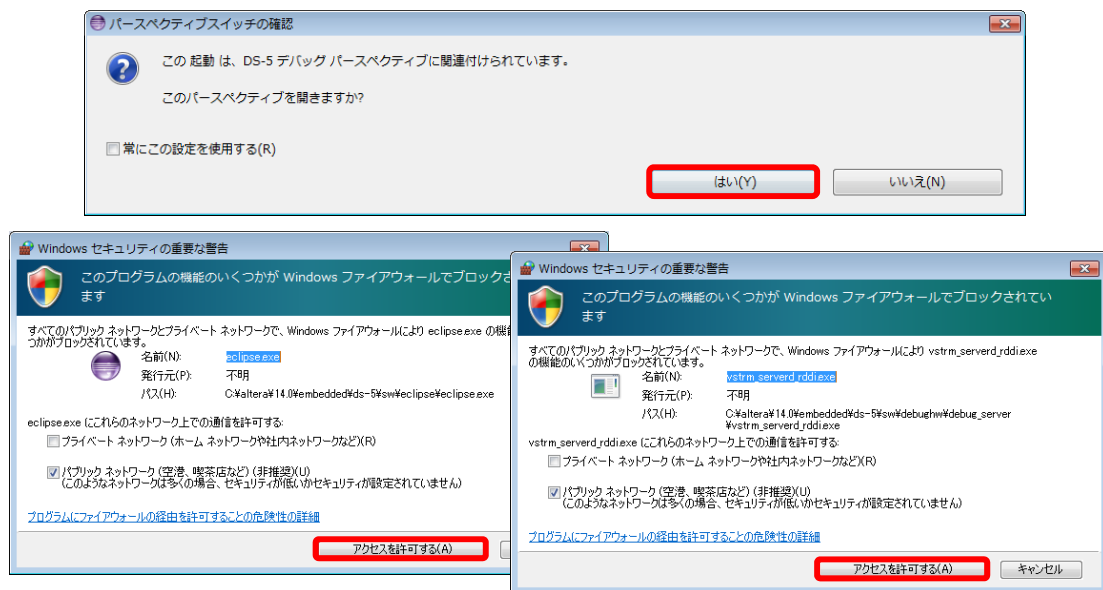
【図 5-13】新規デバッグ構成の Files タブの設定

9. [デバッグ(D)] ボタンをクリックすることで、ボードに対してアプリケーションのダウンロードを行い、デバッグ・セッションを開始します。




【図 5-14】デバッグ・セッションの開始

10. DS-5 デバッグ・パースペクティブ・スイッチの確認 プロンプトが表示されたら [はい(Y)] をクリックします。Windows 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] をクリックします。



【図 5-15】DS-5 デバッグ・パースペクティブ・スイッチの確認




この時点でアプリケーションは main() の先頭に停止します。

- \_\_\_ 11. 「**続行**」  ボタンにてアプリケーションを実行します。

*App Console* にメッセージが表示されることが確認できるはずです。



【図 5-16】 アプリケーションの実行

- \_\_\_ 12. 「**ターゲットから切断**」  ボタンをクリックしてアプリケーションを切断し、「**すべての接続の削除**」  
 ボタンをクリックしてターゲットを削除します。
- \_\_\_ 13. メニュー・バーのショートカット・ボタン  をクリックして元の C/C++ パースペクティブに切り替えます。
- \_\_\_ 14. DS-5 を終了します（任意）。



## 6. FPGA レジスタの確認方法

DS-5 にペリフェラル記述ファイル (.svd) を追加して、レジスタ・ビューから FPGA / Soft IP レジスタを見るには、以下の手順を行います。

- \_\_\_ 1. DS-5 画面左側のプロジェクト・エクスプローラーパネルにある対象プロジェクトをハイライトし、右クリックして「**デバッグ(D)**」⇒「**デバッグの構成(B)**」を選択します（「4-5-1. デバッグの実行」を参照）。
- \_\_\_ 2. 「**デバッグの構成**」ウィンドウから、「**ファイル**」タブをクリックします。
- \_\_\_ 3. 「**ディレクトリからペリフェラル記述ファイルを追加します**」を選択します。
- \_\_\_ 4. [**ファイルシステム**] ボタンをクリックして、  
 <Quartus プロジェクト>¥<Platform Designer プロジェクト>¥synthesis パスを指定します。

### ❗ **Note:**

本資料の説明では、C:¥Work に格納した Atlas-SoC ボード用のハードウェア・デザインを使用しているため、下図のパスを指定しています。

Atlas-SoC ボードの場合は、

C:¥Work¥DE0-Nano-SoC\_v.1.3.0\_HWrevD0\_SystemCD¥Demonstrations¥SoC\_FPGA¥DE0\_NANO\_SOC\_GHRD¥soc\_system¥synthesis

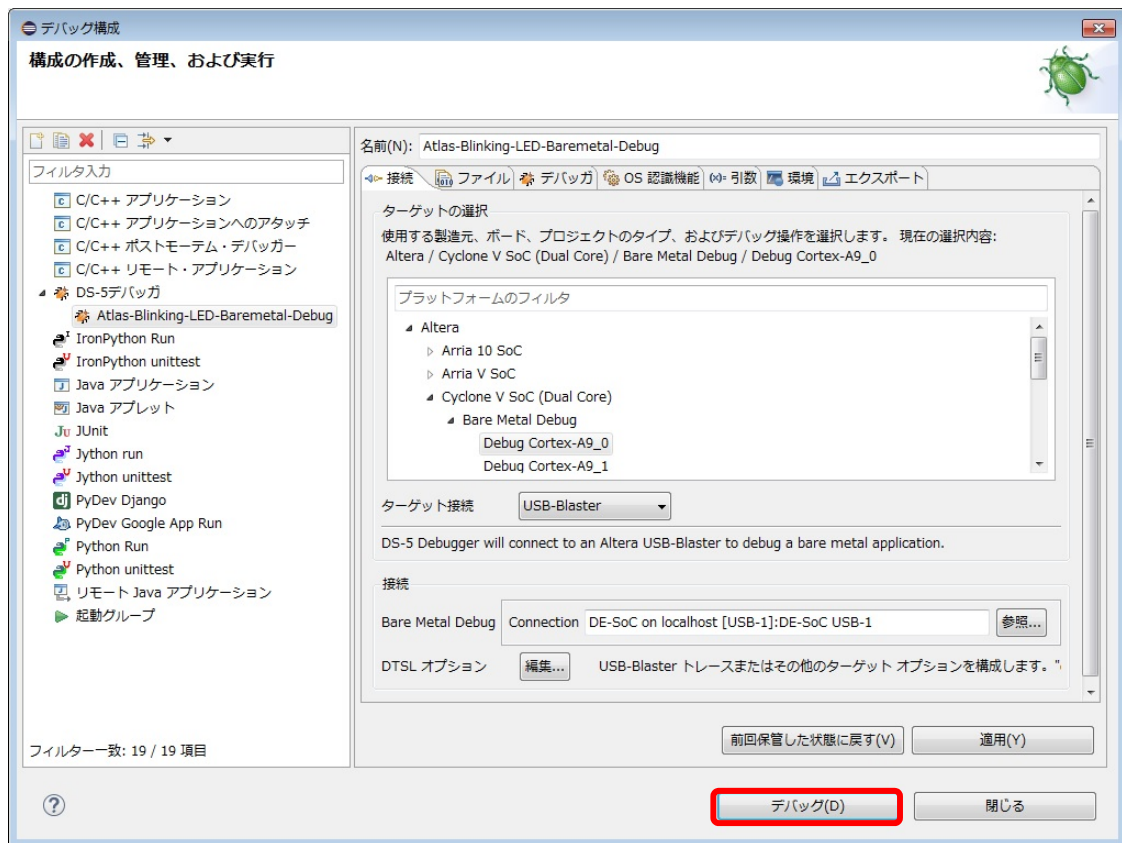
DE10-Nano ボードの場合は、

C:¥Work¥DE10-Nano\_v.1.2.2\_HWrevB2\_SystemCD¥Demonstrations¥SoC\_FPGA¥DE10\_NANO\_SOC\_GHRD¥soc\_system¥synthesis



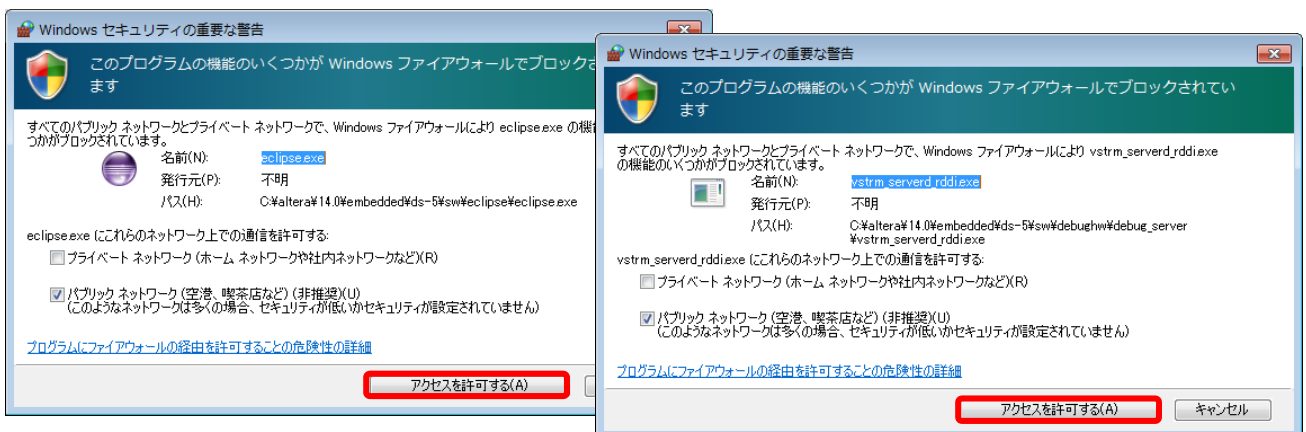
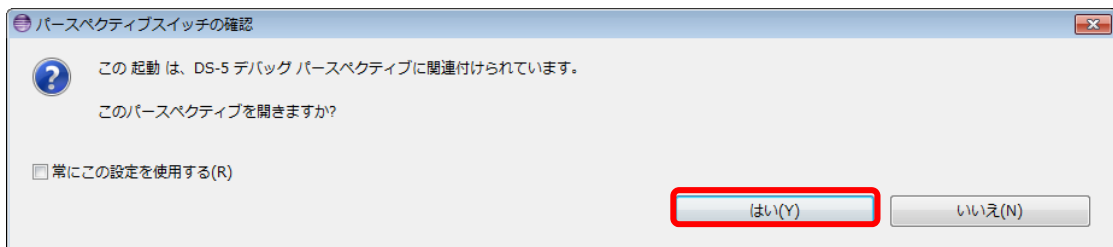
【図 6-1】 DS-5 の「デバッグの構成」でペリフェラル記述ファイルを追加

5. **【デバッグ】** ボタンをクリックして、デバッグ・スクリプトの実行を開始します。



【図 6-2】 【デバッグ】 ボタンをクリック

6. デバッグ パースペクティブへの切り替えのプロンプトが表示されたら **【はい】** を選択します。  
また Windows 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】** をクリックします。



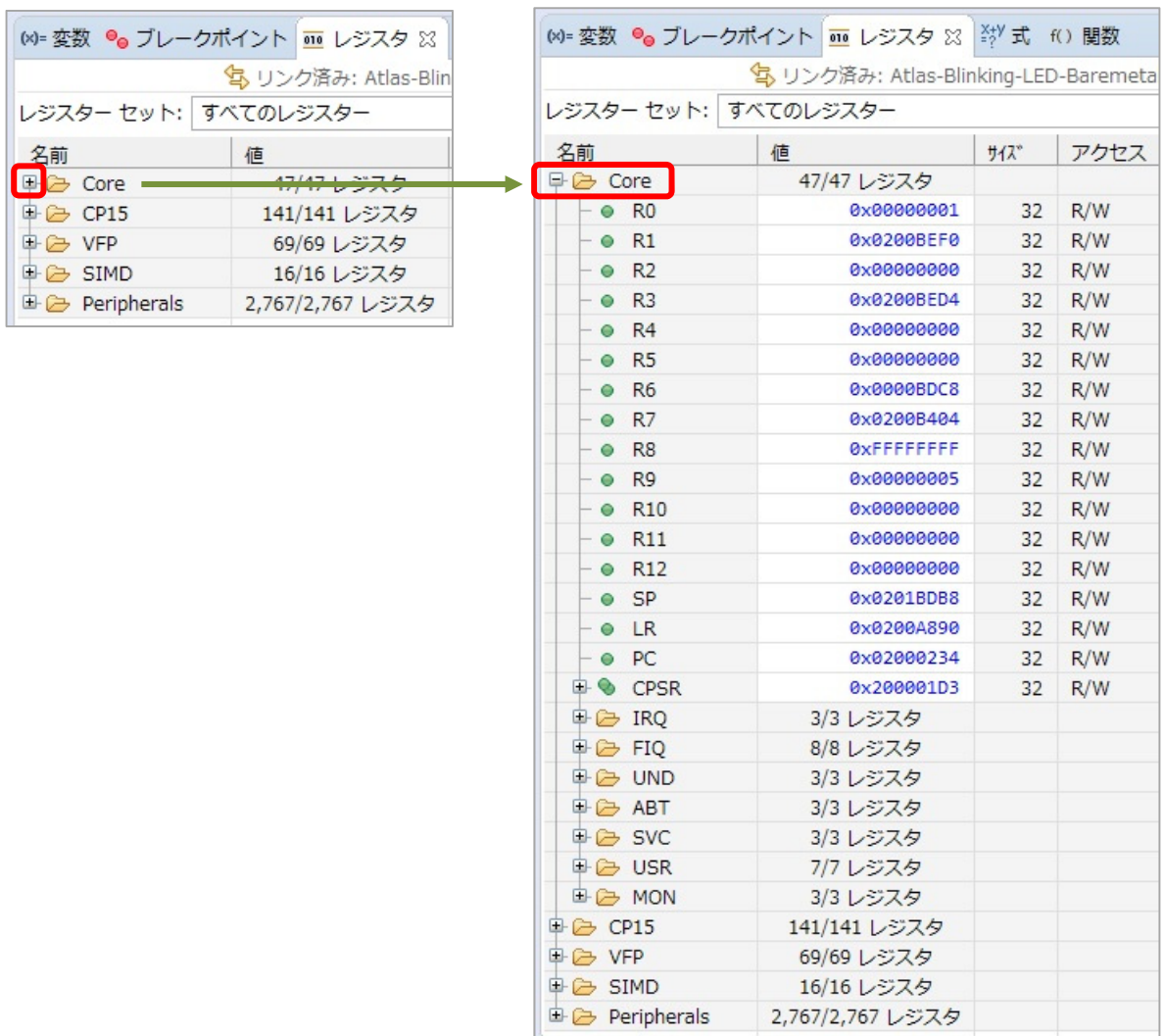
【図 6-3】 デバッグ パースペクティブへの切り替えとファイアウォールのアクセス許可

7. レジスタ・ビューを選択して FPGA と HPS のペリフェラル・レジスタを表示します。



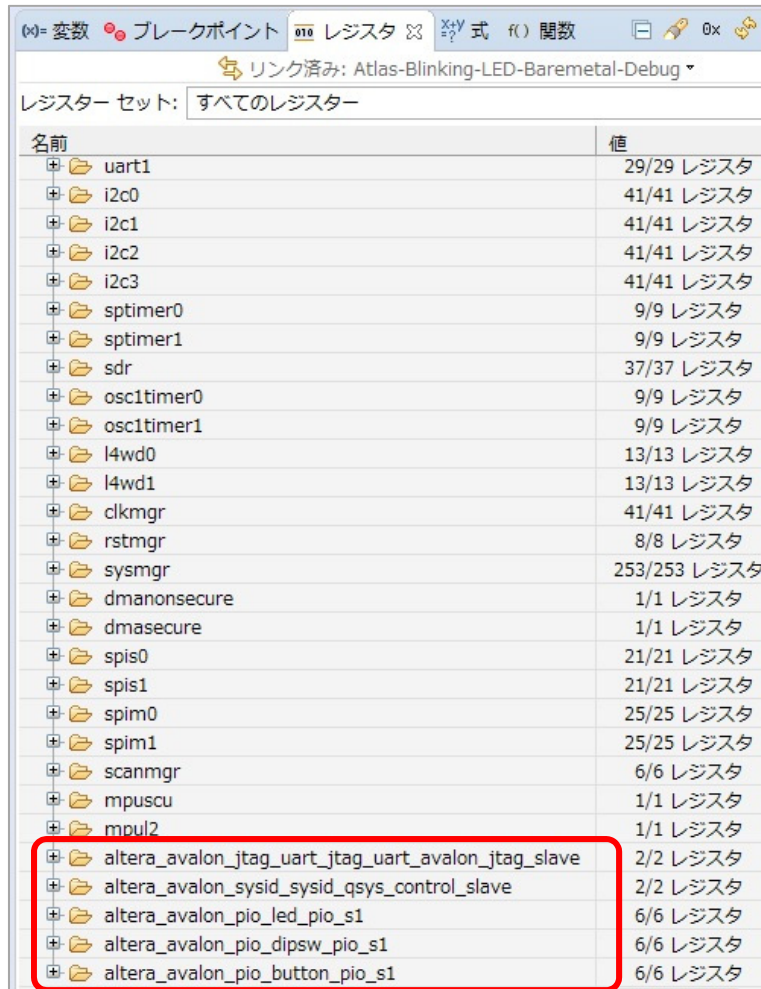
【図 6-4】レジスタ・ビューを選択

8. 「+」記号をクリックして Core レジスタを展開します。コア・レジスタがすべて表示されて編集できるようになります。



【図 6-5】Core レジスタを展開

9. Peripherals レジスタ・グループを展開してリストの最後までスクロールします。  
**altera\_avalon\_** という接頭辞が付いたものが FPGA 内のソフト IP レジスタです。  
「+」記号をクリックしてレジスタを展開することで詳細を確認することができます。



名前	値
uart1	29/29 レジスタ
i2c0	41/41 レジスタ
i2c1	41/41 レジスタ
i2c2	41/41 レジスタ
i2c3	41/41 レジスタ
sptimer0	9/9 レジスタ
sptimer1	9/9 レジスタ
sdr	37/37 レジスタ
osc1timer0	9/9 レジスタ
osc1timer1	9/9 レジスタ
l4wd0	13/13 レジスタ
l4wd1	13/13 レジスタ
clkmgr	41/41 レジスタ
rstmgr	8/8 レジスタ
sysmgr	253/253 レジスタ
dmanonsecure	1/1 レジスタ
dmasecure	1/1 レジスタ
spis0	21/21 レジスタ
spis1	21/21 レジスタ
spim0	25/25 レジスタ
spim1	25/25 レジスタ
scanmgr	6/6 レジスタ
mpuscu	1/1 レジスタ
mpul2	1/1 レジスタ
altera_avalon_jtag_uart_jtag_uart_avalon_jtag_slave	2/2 レジスタ
altera_avalon_sysid_sysid_qsys_control_slave	2/2 レジスタ
altera_avalon_pio_led_pio_s1	6/6 レジスタ
altera_avalon_pio_dipsw_pio_s1	6/6 レジスタ
altera_avalon_pio_button_pio_s1	6/6 レジスタ

【図 6-6】 FPGA 内のソフト IP レジスタ

**注記:**

SoC FPGA において、HPS と FPGA 間のバスは、以下の 3 つのブリッジから構成されています。

- ① **FPGA-to-HPS ブリッジ**: FPGA が HPS 内のスレーブに対してトランザクションを発行可能にするバス
- ② **HPS-to-FPGA ブリッジ**: HPS が FPGA 内のスレーブに対してトランザクションを発行可能にするバス
- ③ **Lightweight HPS-to-FPGA ブリッジ**: 通常、FPGA 内のソフト IP の control および status レジスタ (CSR) アクセス用に使用されます

これらのブリッジを初期化して開通することで、HPS と FPGA 間のアクセスが可能となります。

これらのブリッジを初期化していない状態で、レジスタ・ビューから FPGA 内のソフト IP レジスタを開かないでください。開いた場合、Eclipse の接続が不安定になることがあります。

## 7. カスタム・ボードへの対応方法

この章では、ユーザのカスタム・ボードでベアメタル・アプリケーションを動かすために必要な手順について説明します。

### 7-1. Arm プロセッサを含むハードウェアの設計を行う

本資料では SoC FPGA 向けハードウェア設計に関する詳細な説明はしませんが、概要としては以下のような手順となります。

- \_\_\_ 1. Quartus Prime および Platform Designer を使用し、Arm プロセッサを含むユーザのボードのハードウェアの設計を行います。
- \_\_\_ 2. Hard Processor System (以下、HPS) コンポーネントを Platform Designer システムへ追加しますが、HPS の設定には以下のような多くの重要な設定が含まれていますので、ユーザのボードに合わせた正しい設定を行ってください。
  - AXI ブリッジの有効化
  - HPS に含まれるペリフェラルの選択と有効化
  - HPS クロック設定
  - SDRAM パラメータの設定
- \_\_\_ 3. Platform Designer システムが完成したら Generate して生成します。
- \_\_\_ 4. Quartus Prime で、ピン・アサインメントの設定とプロジェクトのコンパイルを行います。

### 7-2. Preloader (プリローダ) とは？

Preloader は U-boot second program loader (以後、u-boot spl) をベースに、SoC FPGA 向けにカスタマイズが加えられたブートローダです。

(1) Preloader の役割は次の通りです。

- 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 へ次ステージのプログラムの展開・ジャンプ

(2) 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。

(3) ユーザの SoC FPGA を搭載したカスタム・ボードを動かすためには、まずこの Preloader を必ず生成してください。

### 7-3. Preloader の生成手順

以降に Preloader の生成手順を説明します。Preloader の生成方法についての更に詳しい説明は、本資料を入手したサイト内から以下の資料をご覧ください。

#### 参考:

『Preloader Generator の使用方法』

<https://service.macnica.co.jp/library/117865>

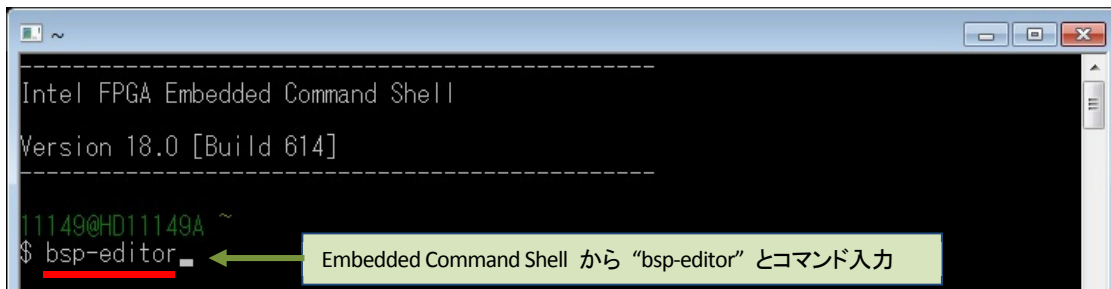
#### 7-3-1. Embedded Command Shell の起動

「4-2-1. Embedded Command Shell の起動」と同じ手順で起動します。

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（embedded フォルダ）に格納されている起動用スクリプトを実行し、Embedded Command Shell を起動します。

#### 7-3-2. bsp-editor（Preloader Generator）の起動

下図のように Embedded Command Shell のウィンドウが開いたら **bsp-editor** とコマンド入力して、bsp-editor（Preloader Generator）の GUI を起動します。



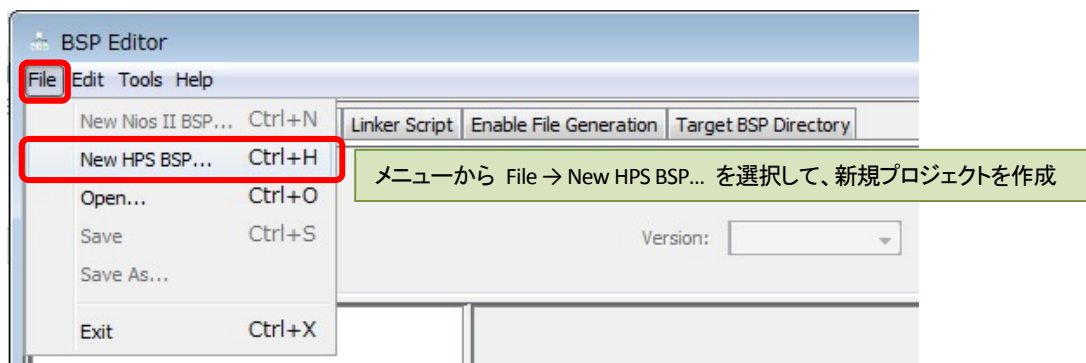
【図 7-1】 bsp-editor（Preloader Generator）の起動

#### 7-3-3. 新規 bsp プロジェクトの作成

図のように bsp-editor（Preloader Generator）の GUI が起動したら、メニューから「File」 「New HPS BSP...」を選択して、新規プロジェクトを作成します。

#### 注記:


SoC EDS v15.0 より前のバージョンでは、「File」 「New BSP...」を選択します。



【図 7-2】 新規 bsp プロジェクトの作成

## 7-3-4. ハンドオフ・ファイルの指定

1. ハードウェア開発で生成した、ハンドオフ・ファイル・フォルダのパス  
 <Quartus プロジェクト>%hps\_isw\_handoff%soc\_system\_hps\_0 を指定します。

図のように Preloader settings directory: の並びにある  を押してフォルダを指定します。

**Note:**

本資料の説明では、C:\Work に格納した Atlas-SoC ボード用のハードウェア・デザインを使用しているため、下図のパスを設定しています。

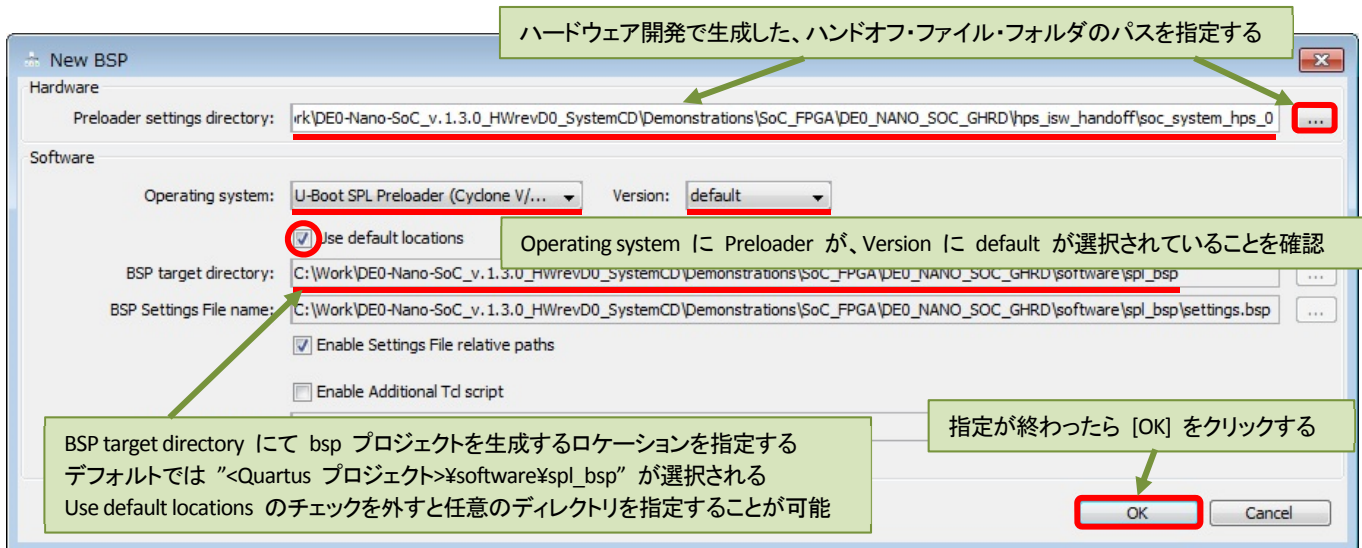
Atlas-SoC ボードの場合は、

C:\Work\DE0-Nano-SoC\_v.1.3.0\_HWrevD0\_SystemCD\Demonstrations\SoC\_FPGA\DE0\_NANO\_SOC\_GHRD\hps\_isw\_handoff\soc\_system\_hps\_0

DE10-Nano ボードの場合は、

C:\Work\DE10-Nano\_v.1.2.2\_HWrevB2\_SystemCD\Demonstrations\SoC\_FPGA\DE10\_NANO\_SOC\_GHRD\hps\_isw\_handoff\soc\_system\_hps\_0

2. 全ての指定が終わったら [OK] をクリックします。



【図 7-3】 ハンドオフ・ファイルの指定

### 7-3-5. Preloader のユーザ・オプション (Common) の設定

Common では Preloader に関する基本的な設定を行います。

(1) **spl** :

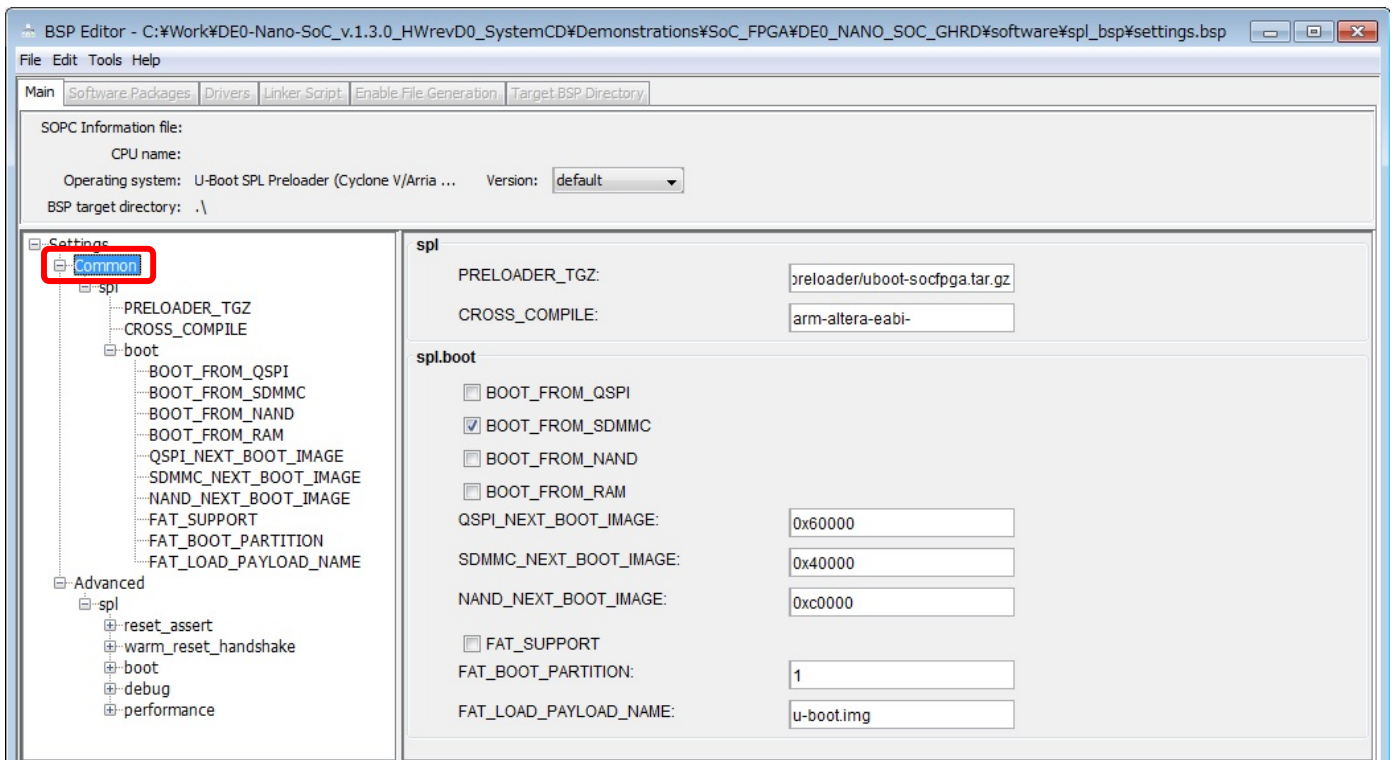
- **PRELOADER\_TGZ** :  
Preloader ソース・ファイルのアーカイブ・ファイルを指定します。基本的に変更する必要はありません。
- **CROSS\_COMPILE** :  
使用するクロス・コンパイラを指定します。基本的に変更する必要はありません。

(2) **boot** :

- **BOOT\_FROM\_QSPI** :  
Preloader に続くブートイメージを QSPI からロードする場合にチェックを入れます。
- **BOOT\_FROM\_SDMMC** :  
Preloader に続くブートイメージを SDMMC からロードする場合にチェックを入れます。
- **BOOT\_FROM\_RAM** :  
Preloader に続くブートイメージを RAM からロードする場合にチェックを入れます。FPGA 側に実装したメモリからのブートにはこの設定を利用します。
- **QSPI\_NEXT\_BOOT\_IMAGE** :  
BOOT\_FROM\_QSPI チェック時に、Preloader がロードするブートイメージの格納アドレスを指定します。
- **SDMMC\_NEXT\_BOOT\_IMAGE** :  
BOOT\_FROM\_SDMMC チェック時、Preloader がロードするブートイメージの格納アドレスを指定します。

**! 注記:**

BOOT メモリ選択は、いずれか 1 つにのみチェックを入れてください (複数にチェックを入れない)。



【図 7-4】 Preloader のユーザ・オプション (Common) の設定

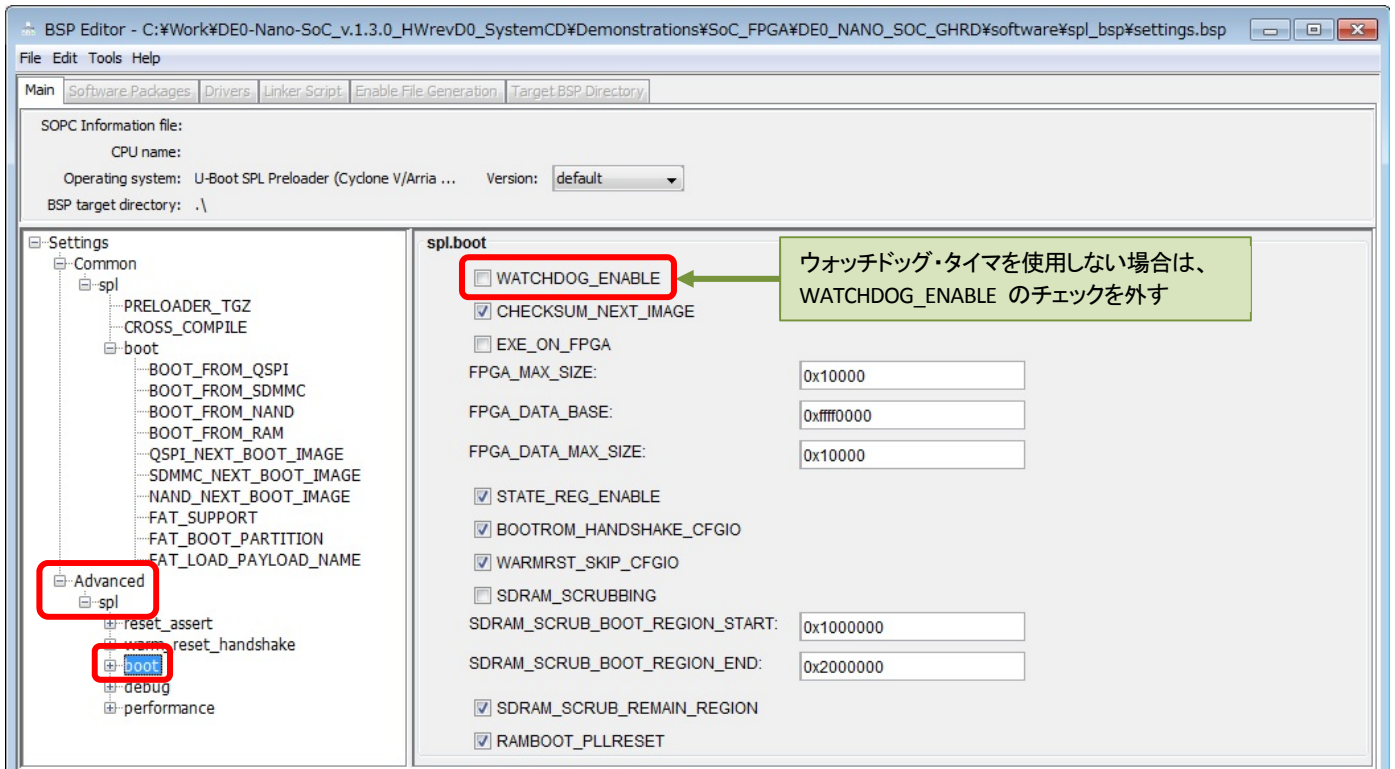


## 7-3-6. Preloader のユーザ・オプション (Advanced spl boot) の設定

Advanced spl boot ではウォッチドッグ・タイマの Disable などブート時の挙動に関して設定を行います。ベアメタル・アプリケーションを使用する場合は、WATCHDOG\_ENABLE のチェックを外します。

 **注記:**

ベアメタル・アプリケーションでウォッチドッグ・タイマを使用する場合は、ウォッチドッグ・タイマが正常に動作するようにプログラム・コードを追加してください。



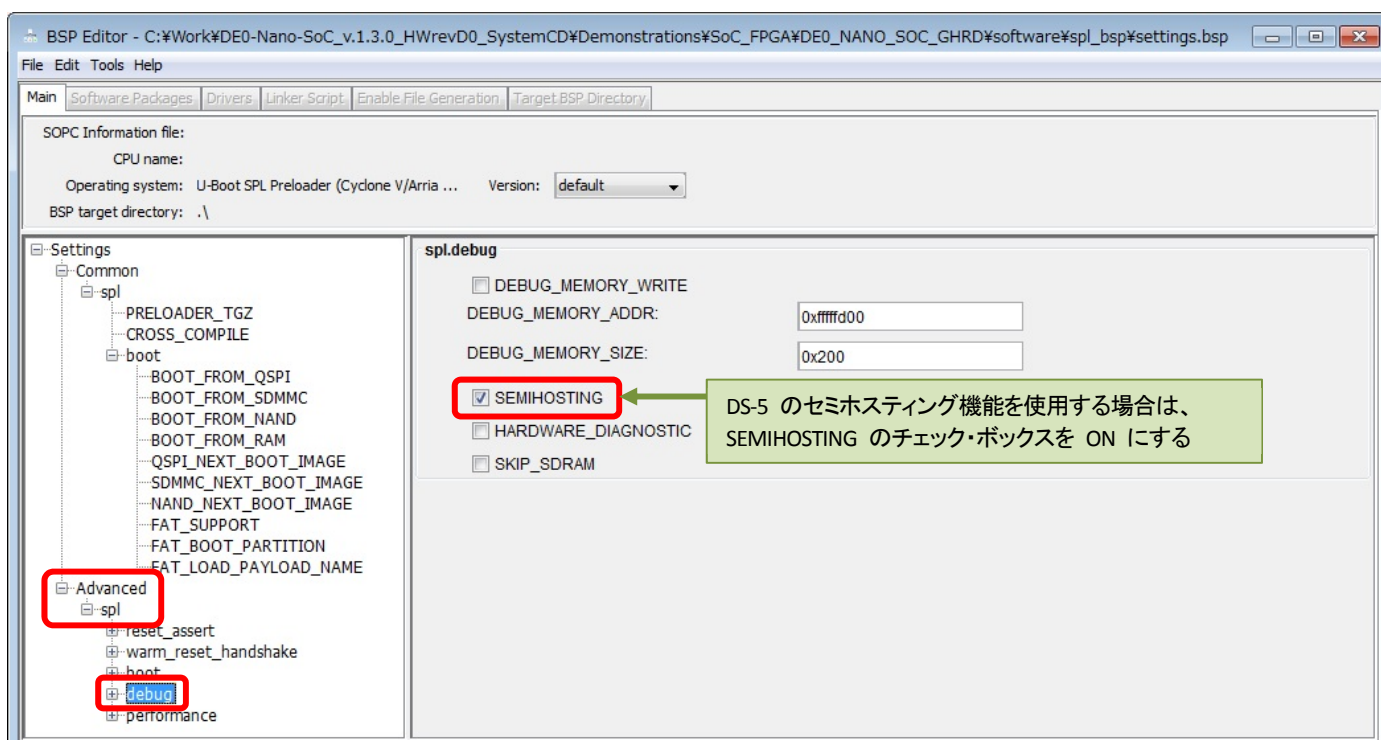
【図 7-5】 Preloader のユーザ・オプション (Advanced spl boot) の設定

## 7-3-7. Preloader のユーザ・オプション (Advanced spl debug) の設定

**Advanced spl debug** では DS-5 の Semihosting 機能のサポート有無等、デバッグ関連の設定を行います。DS-5 Intel® SoC FPGA Edition のセミホスティング機能を使用する場合は、**SEMIHOSTING** のチェック・ボックスを ON にします。

**注記:**

DS-5 を使用せずに、スタンドアロン・ブートさせる場合は、**SEMIHOSTING** のチェック・ボックスを必ず **OFF** にしてください。



【図 7-6】 Preloader のユーザ・オプション (Advanced spl debug) の設定

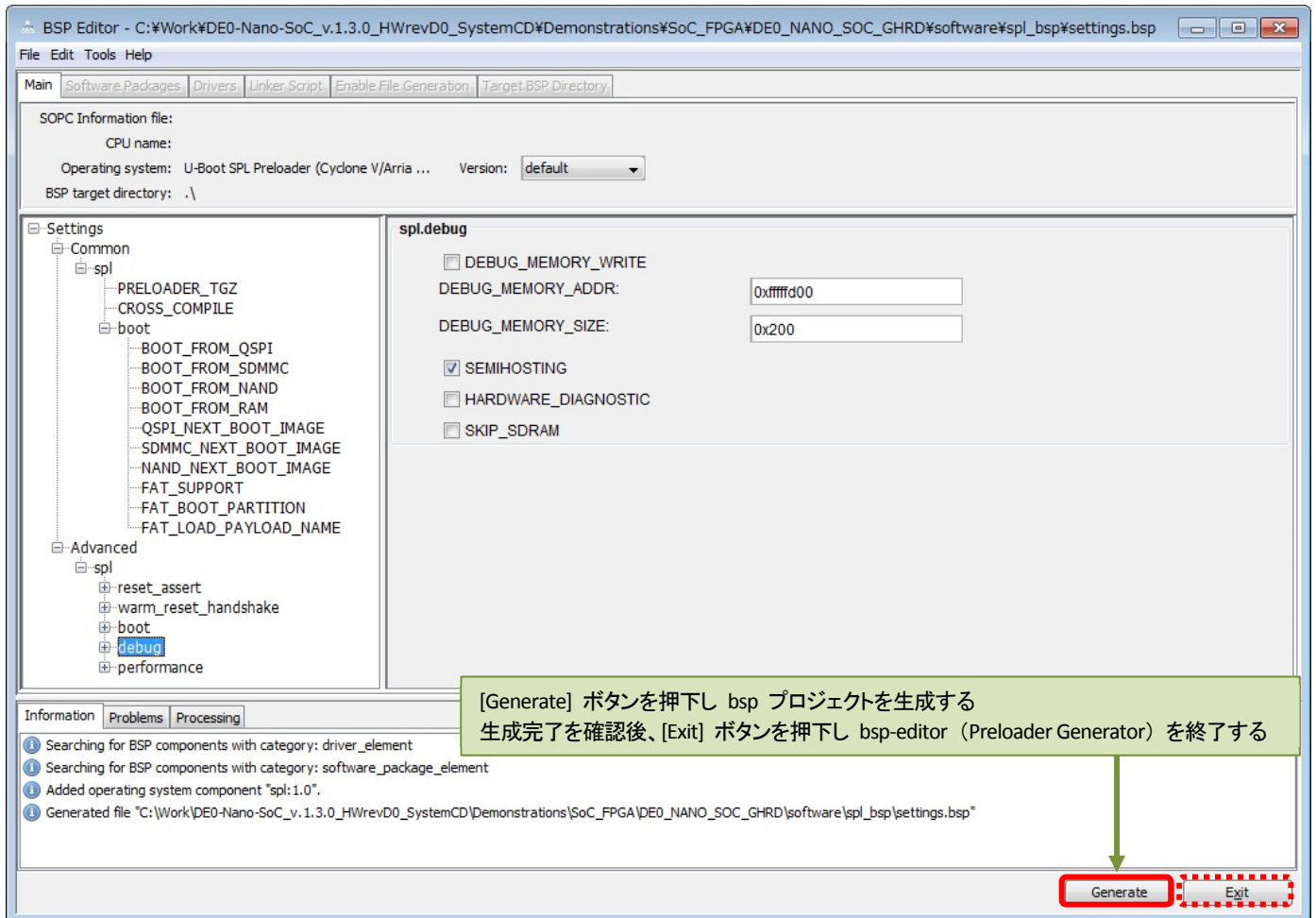
### 7-3-8. bsp プロジェクトの生成 (Generate)

右下の **[Generate]** ボタンを押下し bsp プロジェクトを生成します。

生成する bsp プロジェクトには \*.c 、 \*.h 、 Makefile を含む Preloader を生成 (ビルド) するために必要なファイルが保存されます。

これらのファイルは、「7-3-4. ハンドオフ・ファイルの指定」で BSP target directory に指定したロケーションに生成されます (例では、<Quartus プロジェクト>%software%spl\_bsp)。

生成完了を確認後、**[Exit]** ボタンを押下し bsp-editor (Preloader Generator) を終了します。



【図 7-7】 bsp プロジェクトの生成

## 7-3-9. Preloader のビルド

1. Embedded Command Shell のカレント・ディレクトリを、bsp-editor (Preloader Generator) で作成した bsp プロジェクトのディレクトリに移動します。  
Embedded Command Shell から 以下のようにコマンド入力します。

```
$ cd "<quartus プロジェクト>%software%spl_bsp" ↓
```

❗ **Note:**


本資料の説明では、C:%Work に格納した Atlas-SoC ボード用のハードウェア・デザインを使用しているため、下図のディレクトリに移動しています。

Atlas-SoC ボードの場合は、

```
C:%Work%DE0-Nano-SoC_v.1.3.0_HWrevD0_SystemCD%Demonstrations%SoC_FPGA%DE0_NANO_SOC_GHRD%software%spl_bsp
```

DE10-Nano ボードの場合は、

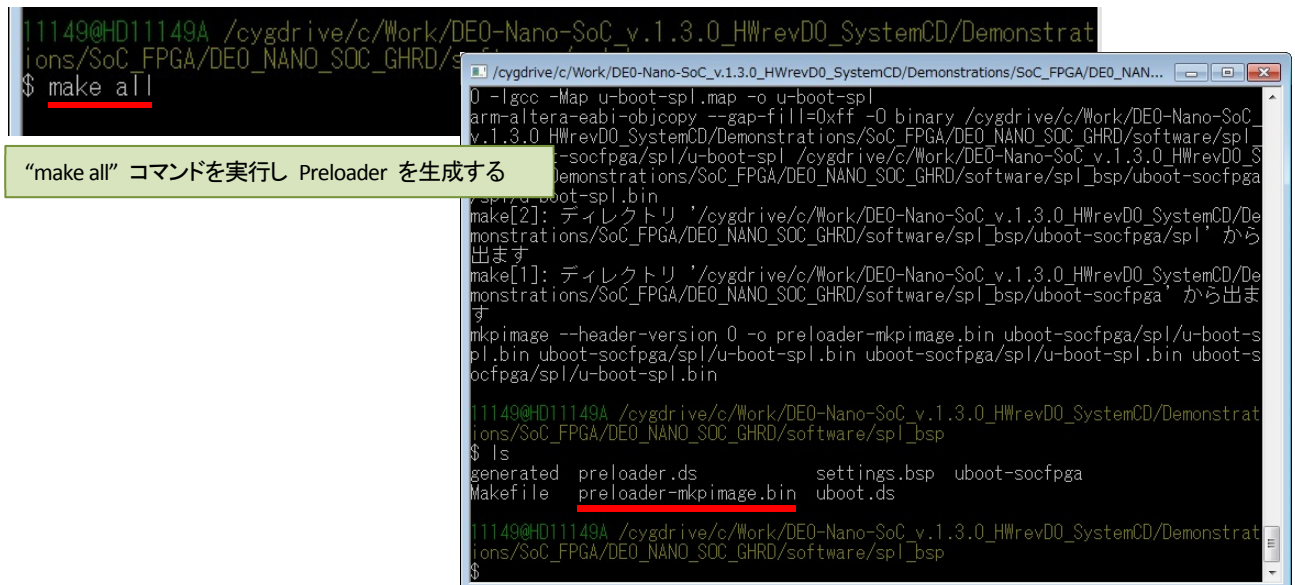
```
C:%Work%DE10-Nano_v.1.2.2_HWrevB2_SystemCD%Demonstrations%SoC_FPGA%DE10_NANO_SOC_GHRD%software%spl_bsp
```



```
11149@HD11149A ~
$ cd "C:%Work%DE0-Nano-SoC_v.1.3.0_HWrevD0_SystemCD%Demonstrations%SoC_FPGA%DE0_NANO_SOC_GHRD%software%spl_bsp"
11149@HD11149A /cygdrive/c/Work/DE0-Nano-SoC_v.1.3.0_HWrevD0_SystemCD/Demonstrations/SoC_FPGA/DE0_NANO_SOC_GHRD/software/spl_bsp
$ ls
generated Makefile preloader.ds settings.bsp uboot.ds
```

【図 7-8】 bsp プロジェクトのディレクトリに移動

2. **make all** ↓ コマンドを実行し Preloader を生成します。  
**ls** ↓ コマンドにて **preloader-mkpmimage.bin** が生成されていることを確認します。このファイルは BootROM にて参照される Preloader 用のヘッダ情報を付加したバイナリ・ファイルで、SD カードや QSPI フラッシュ・メモリへ書き込むファイルとなります。



```
11149@HD11149A /cygdrive/c/Work/DE0-Nano-SoC_v.1.3.0_HWrevD0_SystemCD/Demonstrations/SoC_FPGA/DE0_NANO_SOC_GHRD/software/spl_bsp
$ make all
11149@HD11149A /cygdrive/c/Work/DE0-Nano-SoC_v.1.3.0_HWrevD0_SystemCD/Demonstrations/SoC_FPGA/DE0_NANO_SOC_GHRD/software/spl_bsp
$ ls
generated preloader.ds settings.bsp uboot-socfpga
Makefile preloader-mkpmimage.bin uboot.ds
```

【図 7-9】 “make all” コマンドを実行

**⚠ 注記:**

ホスト PC の OS が Windows® 10 の場合、Preloader の生成でエラーが発生する場合があります。

**【エラー内容】**

この問題は、SOC EDS ツールを使用してプリローダを生成するときに発生します。

新しい HPS および BSP 設定ファイルを作成した後、以下のように make コマンドが失敗します。

```
tar zxf /cygdrive/c/intelFPGA/18.0/embedded/host_tools/altera/preloader/uboot-socfpga.tar.gz
tar: Error opening archive: Failed to open '/cygdrive/c/intelFPGA/18.0/embedded/host_tools/altera/preloader/uboot-socfpga.tar.gz'
make: *** [uboot-socfpga/.untar] Error 1
```

もしご使用の OS が Windows® 10 でエラーが発生する場合は、以下の参考情報サイトで説明されている対策が必要となりますのでご注意ください。

**【参考情報サイト】**

- ① Intel® Knowledge Base - Unable to make preloader in Windows 10  
[https://www.intel.com/content/altera-www/global/en\\_us/index/support/support-resources/knowledge-base/embedded/2018/unable-to-make-preloader-in-windows-10.html](https://www.intel.com/content/altera-www/global/en_us/index/support/support-resources/knowledge-base/embedded/2018/unable-to-make-preloader-in-windows-10.html)
- ① MACNICA フォーラム  
<https://forum.macnica.co.jp/t/topic/1191/11>

## 7-4. カスタム・ボード向けに生成した Preloader を DS-5 デバッグで使用方法

ユーザのカスタム・ボードでベアメタル・アプリケーションを動かすために必要な手順としては、主に以下の 2 つがあります。

### (1) Preloader ファイルの差し替え

カスタム・ボード向け Preloader を生成し、生成したバイナリでサンプル・プロジェクトの Preloader ファイルを差し替えます。差し替え対象のファイルは **u-boot-spl.axf** です。

### (2) Makefile の編集

Makefile 内にデフォルトの Preloader に関する処理がある場合は、Makefile から削除します。

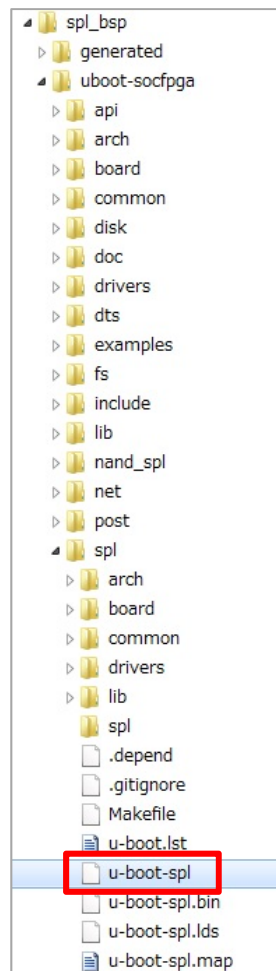
### 7-4-1. Preloader ファイルの差し替え

(1) DS-5 でプログラムを実行／デバッグするには、Arm Executable and Linkable Format (ELF) ファイルをロードします。

この形式は、Arm ELF 仕様書で説明されており、.axf というファイル拡張子を使用します。

(2) 「7-3. Preloader の生成手順」で Preloader を生成しましたが、このとき、

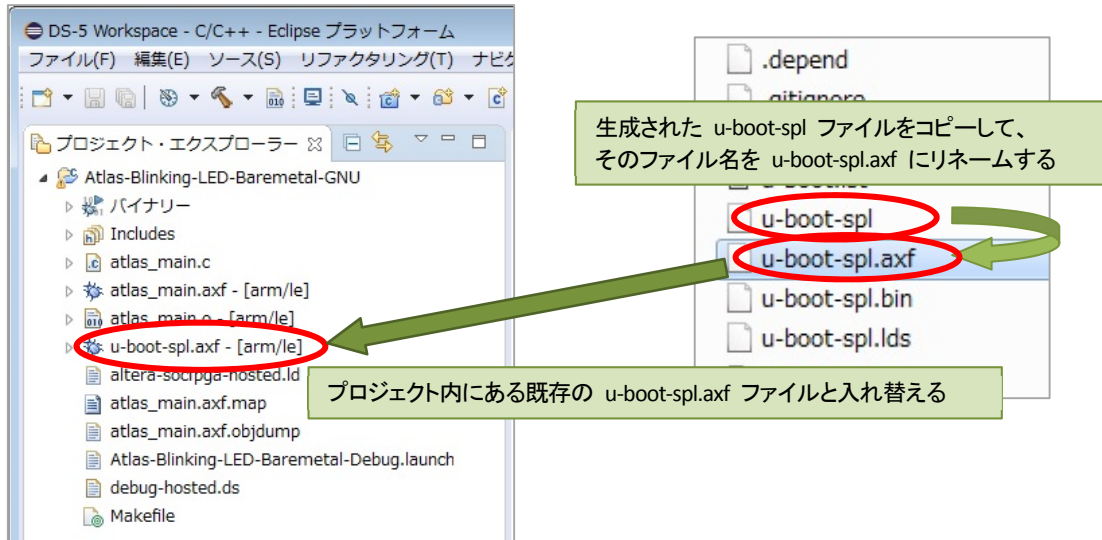
<Quartus プロジェクト>\software\spl\_bsp\uboot-socfpga\spl フォルダの下に **u-boot-spl** ファイルが生成されます。このファイルが Preloader の ELF ファイルとなります。



【図 7-10】 生成された u-boot-spl ファイル

- (3) カスタム・ボード向けに生成した Preloader の実行可能バイナリ・ファイル **u-boot-spl** をコピーして、そのファイル名を **.axf** 拡張子をつけて **u-boot-spl.axf** とリネームします。

そのファイルを例えば **Atlas-Blinking-LED-Baremetal-GNU** ベアメタル・サンプル・プロジェクト内にある既存の **u-boot-spl.axf** ファイルと入れ替えれば、ユーザのカスタム・ボード向けの Preloader を使用して DS-5 で実行 / デバッグを行うことが可能です。



【図 7-11】 u-boot-spl ファイルをコピーして u-boot-spl.axf とリネームし差し換える

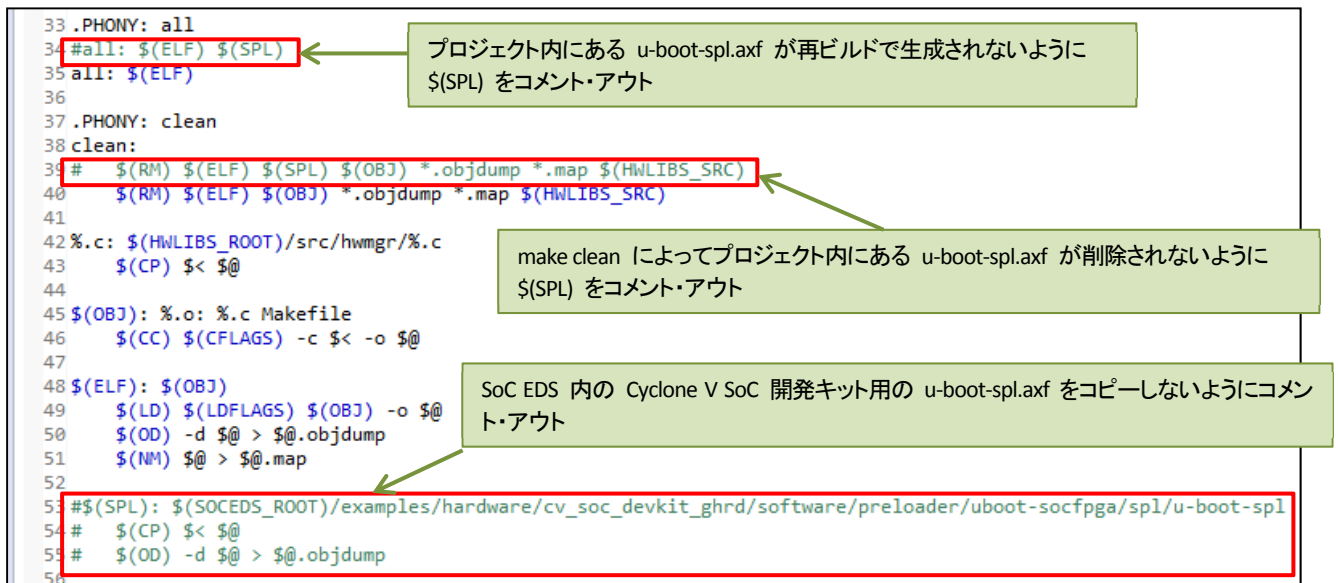
#### 7-4-2. Makefile の編集

必要に応じて **Makefile** を編集して Preloader に関する部分をコメント・アウトおよび編集します。

#### ⓘ Note:

ご利用のサンプル・アプリケーションにより、Makefile の記述内容が異なる場合があります。

- (1) SoC EDS に格納される Cyclone® V SoC 開発キットの Preloader をコピーしないように変更します。
- (2) `make clean` を実行した際にコピーしたカスタム・ボード向け Preloader 実行可能バイナリ **u-boot-spl.axf** が削除されないように変更します。



【図 7-12】 Makefile の編集例

## 8. ベアメタル・アプリケーションを SD カードからスタンドアロン実行する例

この章では、ベアメタル・アプリケーションを SD カードからスタンドアロン実行できるようにするために必要な手順について説明します。

### 8-1. SD カードの準備

SD カードを作成し、SD カード内の Preloader を生成したものに入れ替えます。

#### ⚠ 注記 1:

ベアメタル・アプリケーションをスタンドアロン・ブートさせる場合は、bsp-editor (Preloader Generator) にて “SEMIHOSTING” のチェック・ボックスを必ず OFF にして生成した Preloader を使用してください。

#### ⚠ 注記 2:

ベアメタル・アプリケーションを使用する場合は、bsp-editor (Preloader Generator) にて “WATCHDOG\_ENABLE” のチェック・ボックスを OFF にして生成した Preloader を使用してください。また、ベアメタル・アプリケーションでウォッチドッグ・タイマを使用する場合は、ウォッチドッグ・タイマが正常に動作するようにプログラム・コードを追加してください。

Cyclone® V SoC FPGA 評価キット (インテル純正ボード、3rd ベンダ製ボード) では、通常 Linux を起動させるための SD カード・イメージ・ファイルが提供されています。

インテルの Cyclone® V SoC FPGA Development Board の SD カード・イメージ・ファイルは、以下の [RocketBoards.org](https://rocketboards.org) のページで公開されています。

ダウンロードして解凍すると、SD カード・イメージ・ファイル (.img) があります。

<https://releases.rocketboards.org/release/2017.10/gsr/bin/linux-socfpga-gsr-17.1std-cv.tar.gz>

[https://releases.rocketboards.org/release/2018.10/gsr/cv\\_gsr/sdimage.tar.gz](https://releases.rocketboards.org/release/2018.10/gsr/cv_gsr/sdimage.tar.gz)

- (1) SD カードの場合、Preloader は Windows のファイル・システムではアクセス不可能な領域に格納されています。

従って、まずは [RocketBoards.org](https://rocketboards.org) に公開されている SD カード・イメージを microSD カードに書き込んでください。これにより、microSD カード内に Preloader を書き込むためのパーティションが作成されます。

SD カード・イメージ・ファイルを microSD カードに書き込むには、フリーソフト「Win32 Disk Imager」などをご利用ください。

Win32 Disk Imager

<https://sourceforge.net/projects/win32diskimager/>

SD カード・イメージの書き込み方法については、下記リンクのビデオを参照ください。

<https://www.youtube.com/watch?v=0rckwOGTH5U&t=3s>

- (2) Preloader バイナリ・ファイルの書き込み方法は、SoC EDS ツールのバージョンにより異なります。

- **SoC EDS v13.1 以前のバージョン**

基本的に Linux の dd ユーティリティを使用します (Linux OS がインストールされた PC が必要です)。

- **SoC EDS v14.0 以降のバージョン**

インテルが提供する alt-boot-disk-util というツールを使用して書き込むことができます。



### 8-1-1. SoC EDS v13.1 以前のバージョン で Preloader を SD カードに書き込むには

- (1) SoC EDS v13.1 以前では SoC EDS 自体に Preloader バイナリを書き込むためのツールが無いいため、基本的に Linux の dd ユーティリティ・コマンドを使用します。そのため **Linux OS がインストールされた PC** が必要です（仮想マシンでも可能）。
- (2) Preloader を部分的にアップデートするには、Linux PC に SD カードをマウントし dd コマンドを実行します。書き込む Preloader のバイナリ・ファイルは「7-3-9. Preloader のビルド」の手順で生成したヘッダ情報が付加された **preloader-mkpmimage.bin** です。

1. **Linux PC** に SD カードを接続すると、この例では sdb が見えるようになります（PC の SD カード・スロットで見えない場合は、USB カード・リーダ経由で試してください）。

```
$ sudo cat /proc/partitions
```

#### 【 Ubuntu での表示例 】

```
11      0   1048575 sr0
 8      0  100663296 sda
 8      1   96468992 sda1
 8      2           1 sda2
 8      5   4191232 sda5
 8     16   7822336 sdb
 8     17   5120000 sdb1
 8     18  15360000 sdb2
 8     19    102400 sdb3
```

#### 【 CentOS での表示例 】

```
 8      0  33554432 sda
 8      1   5120000 sda1
 8      2  33041408 sda2
253     0  30973952 dm-0
253     1   2064384 dm-1
 8     16   7822336 sdb
 8     17   5120000 sdb1
 8     18  15360000 sdb2
 8     19    102400 sdb3
```

2. dd コマンドを実行して Preloader バイナリを書き込みます。

```
$ sudo dd if=preloader-mkpmimage.bin of=/dev/sdb3 bs=64k seek=0
```

### 8-1-2. SoC EDS v14.0 以降のバージョン で Preloader を SD カードに書き込むには

- (1) SoC EDS v14.0 からは、インテルが提供する **alt-boot-disk-util** というツールを使用して書き込むことができます。
- (2) このツールは Windows 上から Embedded Command Shell のウィンドウからコマンド入力することで実行可能です。
- (3) Preloader を部分的にアップデートするには、**Embedded Command Shell から** 下記のコマンドを実行します。

#### ⚠ 注記:

下記コマンドにおいて、**-d F** は Windows PC に挿した SD カードのドライブが **F ドライブ**である場合の指定例です。SD カードのドライブは実際のご使用環境に合わせて指定してください。

```
$ alt-boot-disk-util -p preloader-mkpmimage.bin -a write -d F
```

## 8-2. DS-5 プロジェクトへの追加ファイル

ベアメタル・アプリケーションをスタンドアローン実行できるようにするために、次に説明する 2 つのファイル（`startup.s`、`altera-socfpga-unhosted.ld`）を用意して、DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU ベアメタル・サンプル・プロジェクト）へ追加します。

### (1) `startup.s`

スタートアップ・ルーチンにおいて割り込み禁止にすることで、ユーザ・アプリケーション実行時に一旦割り込みを無効化してハングアップを防ぎます（`u-boot` で有効化した割り込みが、ユーザ・アプリケーション実行後に入らないようにするための対処です）。

`startup.s` のソース・コードの例を以下に示します。テキスト・エディタで記述して `startup.s` という名前で保存し、DS-5 プロジェクトに追加してください。

```
.global __reset

.text

__reset:
    mrs r0,cpsr          /* Interrupt disable */
    orr r0,r0,#0xC0     /* Interrupt disable */
    msr cpsr_cxsf,r0    /* Interrupt disable */

    mrc p15,0,r0,c1,c0,2 /* read CPACR */
    orr r0,r0,#0x00f00000 /* Enable NEON/VFP user access */
    mcr p15,0,r0,c1,c0,2 /* Write to CPACR */
    isb
    mov r0,#0x40000000   /* Enable VFP/NEON Hardware */
    mcr p10,7,r0,cr8,cr0,0 /* Write to fpexc */

    b _start
```

【リスト 8-1】 `startup.s` のソース・コードの例

### (2) `altera-socfpga-unhosted.ld`

例として、Atlas-Blinking-LED-Baremetal-GNU ベアメタル・サンプル・プロジェクト内にある既存の `altera-socfpga-hosted.ld` リンカ・スクリプト・ファイルをコピーして、`altera-socfpga-unhosted.ld` にファイル名を変更し、以下の 2 箇所を変更します。

これにより、上記 (1) で追加したスタートアップ・ルーチン `startup.s` の `__reset` を参照するようになります。

```
/*GROUP(-lgcc -lc -lcs3 -lcs3hosted -lcs3arm)*/          /* オリジナルをコメント・アウト */
GROUP(-lgcc -lc -lcs3 -lcs3unhosted -lcs3arm)          /* lcs3unhosted に変更 */

/* PROVIDE(__cs3_reset = __cs3_reset_generic); */      /* オリジナルをコメント・アウト */
PROVIDE(__cs3_reset = __reset);                        /* 追加した startup.s の __reset を参照するように変更 */
```

【リスト 8-2】 `altera-socfpga-unhosted.ld` リンカ・スクリプト・ファイル（変更箇所）

### 8-3. DS-5 プロジェクトの Makefile の修正

ベアメタル・アプリケーションをスタンドアロン実行できるようにするために、DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクト）の **Makefile** を追記・修正します。

これにより、startup.s アセンブラ・ソース・コードを追加して、リンカ・スクリプト・ファイルとして altera-socfpga-unhosted.ld を使用するようになります。

既存の Makefile に対して以下の箇所を追記・修正します。



#### 注記:

オリジナルの Makefile は、Makefile\_org のようにリネームして保存しておくことをお奨めします。

```

ASM_SRC := startup.s                                # 追記

ASFLAGS := -march=armv7-a -mcpu=cortex-a9          # 追記

#LINKER_SCRIPT := altera-socfpga-hosted.ld         # オリジナルをコメント・アウト
LINKER_SCRIPT := altera-socfpga-unhosted.ld       # 追記

AS := $(CROSS_COMPILE)as                          # 追記

#OBJ := $(patsubst %.c,%.o,$(C_SRC))              # オリジナルをコメント・アウト
C_OBJ := $(patsubst %.c,%.o,$(C_SRC))            # 追記
ASM_OBJ := $(patsubst %.s,%.o,$(ASM_SRC))        # 追記
OBJ := $(C_OBJ) $(ASM_OBJ)                       # 追記

#$(OBJ): %.o: %.c Makefile                        # オリジナルをコメント・アウト
$(C_OBJ): %.o: %.c Makefile                      # 追記
    $(CC) $(CFLAGS) -c $< -o $@

$(ASM_OBJ): %.o: %.s Makefile                    # 追記
    $(AS) $(ASFLAGS) -c $< -o $@

```

【リスト 8-3】 Makefile の追記・修正箇所

#### 8-4. SD カード実行のためのソース・ファイルの変更

ベアメタル・アプリケーションをスタンドアローン実行する際、**必要に応じて** アプリケーションのソース・コードを修正します。

ここでは例として、Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクト内にある既存の `atlas_main.c` ソース・ファイルを一部追記・修正しています。

#### ❗ Note:

- ※ `printf()` 文を使用してコンソールに文字出力するには、UART 出力関数の作成・追加が必要となりますが、この例では作成・追加を行いませんので、`printf()` 文によるコンソール文字出力は実行されません。
- ※ `printf()` 文が機能しない状態では、LED の点滅が速く目視で確認しづらくなるため、この例では `for (j=0; j < 0x10000; j++);` によるディレイを追加しています。

```
int main(int argc, char** argv)
{
    int i;
    int j; /* 追加 */

    printf("Hello from Atlas. \n");

    while(1)
    {
        for(i=0; i < 16; i++){
            alt_write_word(LED_BASE_ADDR, i);
            printf("LED [%x] \n", i);
            for (j=0; j < 0x10000; j++); /* ディレイを追加 */
        }
    }

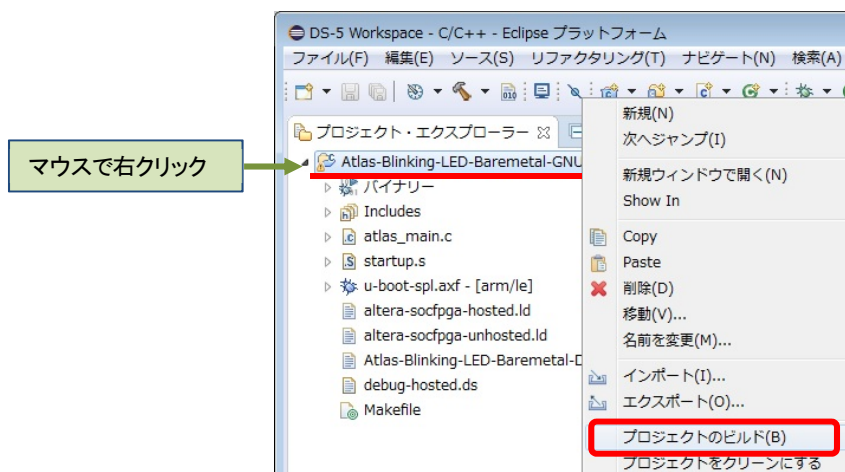
    return 0;
}
```

【リスト 8-4】 atlas\_main.c ソース・ファイルの変更

#### 8-5. ベアメタル・アプリケーションのビルド

ここまでの手順が全て終わったら、DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクト）を右クリックして **プロジェクトのビルド(B)** を実行します。

ビルドが完了すると、ベアメタル・アプリケーションの `.axf` ファイル（この例では、`atlas_main.axf`）が生成されます。



【図 8-1】 プロジェクトのビルド

## 8-6. ベアメタル・アプリケーションの実行バイナリの作成と起動

ベアメタル・アプリケーションをスタンドアローン実行させるには、次の 2 つの方法があります。

- (1) SD カードの **FAT 領域** にアプリケーションの実行バイナリ・ファイルを配置して **u-boot から起動** する方法 (7-6-1 節で説明)
- (2) SD カードの **u-boot 領域** にアプリケーションの実行バイナリ・ファイルを格納して **Preloader から起動** する方法 (7-6-2 節で説明)

### 8-6-1. アプリケーションを SD カードの「FAT 領域」に配置して“u-boot から起動”する方法

u-boot からユーザ・アプリケーションを起動する場合は、アプリケーションの実行バイナリ・ファイルを生成し、それを Windows から認識可能な FAT パーティションに対してコピーすることになります。

以下に手順を説明します。

1. **Embedded Command Shell から** 以下のコマンドを実行して、DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU サンプル・プロジェクト）ディレクトリに移動して、ベアメタル・アプリケーションの実行バイナリ・ファイル（この例では、atlas\_main.bin）を作成します。

```
$ cd "C:\Work\DS-5 Workspace\Atlas-Blinking-LED-Baremetal-GNU"
$ arm-altera-eabi-objcopy -v -O binary atlas_main.axf atlas_main.bin
```

2. 生成されたベアメタル・アプリケーション実行バイナリ・ファイル（この例では、atlas\_main.bin）を SD カードの FAT 領域にコピーします。
3. アプリケーションが FPGA 側にアクセスする場合は、FPGA をコンフィグレーションしておきます。
4. SD カードをボードに取り付けて、ボードを WARM リセットします。u-boot のコンソールにオート・ブートのカウント・ダウンが表示されるので、Enter キーを入力してカウント・ダウンを停止させます。
5. u-boot のプロンプトから 以下のコマンドを実行するとアプリケーションが起動します。

#### 注記:

下記コマンド例は、アプリケーションのスタート・アドレスが 0x02000000 から作成されている場合の例です。アプリケーションのスタート・アドレスは、リンカ・スクリプト・ファイル (.ld) 内の **ORIGIN** で定義されています。

```
# fatload mmc 0 0x02000000 atlas_main.bin
# go 0x02000000
```

6. 前述の u-boot のコマンドを自動化する場合には、u-boot のスクリプト機能が使用可能です。

コマンドをテキスト・ファイルとして保存し、mkimage ツールを使ってヘッダ情報を付加したものを FAT パーティションに格納しておきます。

u-boot がロード対象とするスクリプト名は **u-boot.scr** となっています。

- ① 例えば、下記のような内容でテキスト・ファイル script.txt を作成します。

```
run bridge_enable_handoff;
fatload mmc 0 0x02000000 atlas_main.bin;
go 0x02000000;
```

#### 【リスト 8-5】 script.txt の記述例

- ② mkimage ツールを実行して script.txt から u-boot.scr を生成します。

u-boot で実行するファイルは、mkimage コマンドで u-boot 用のフォーマットに変換する必要があります。

[Embedded Command Shell から](#) 以下のコマンドを実行します。

```
$ mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "My script" -d script.txt u-boot.scr
```

#### ⚠ 注記:

**SoC EDS v13.1 以前のバージョン** では、デフォルトで mkimage ツールのパスが通っていないため、DS-5 プロジェクト（この例では、Atlas-Blinking-LED-Baremetal-GNU ベアメタル・サンプル・プロジェクト）内に mkimage.exe ファイルをコピーしておく必要があります。

mkimage.exe は、bsp-editor で生成した以下のフォルダにあります。

**spl\_bsp/uboot-socfpga/tools/mkimage** (spl\_bsp は bsp-editor で指定したフォルダです)

また、プロジェクト内にコピーした mkimage.exe を実行する際は、コマンド・ラインにおいて **mkimage** と入力してください。

- ③ SD カードを PC の SD カード・スロットに取り付けて、生成された u-boot.scr を SD カードの FAT 領域にコピーします。
- ④ SD カードをボードに取り付けて、ボードを WARM リセットすると、u-boot 実行後にベアメタル・アプリケーションが自動的に起動します。

## 8-6-2. アプリケーションを SD カードの「u-boot 領域」に格納して“Preloader から起動”する方法

Preloader からユーザ・アプリケーションを起動する場合は、アプリケーションの実行バイナリ・ファイルを生成し、それを SD カードの u-boot 領域に対してコピーすることになります。

以下に手順を説明します。

- \_\_\_ 1. **Embedded Command Shell から** 前述「8-6-1. \_\_\_ 1」のコマンドを実行してベアメタル・アプリケーションの実行バイナリ・ファイル（この例では、atlas\_main.bin）を作成します。
- \_\_\_ 2. 以下のコマンドを実行して u-boot 領域に格納するベアメタル・アプリケーション実行バイナリ・ファイル（この例では、atlas\_main.img.bin）を作成します。

### ⚠ 注記:

下記コマンド例は、アプリケーションのスタート・アドレスが 0x02000000 から作成されている場合の例です。アプリケーションのスタート・アドレスは、リンカ・スクリプト・ファイル (.ld) 内の **ORIGIN** で定義されています。

```
$ mkimage -A arm -O u-boot -T standalone -C none -a 0x02000000 -e 0 ¥  
-n "baremetal image" -d atlas_main.bin atlas_main.img.bin ↵
```

- \_\_\_ 3. SD カードの u-boot 領域にベアメタル・アプリケーション実行バイナリ・ファイル（この例では、atlas\_main.img.bin）を書き込みます。

ご使用の SoC EDS のバージョンに応じて、下記 ① または ② の方法で SD カードの u-boot 領域に実行バイナリ・ファイルを書き込みます。

- ① **SoC EDS v13.1 以前** のバージョンでは、SoC EDS 自体に バイナリ・ファイルを書き込むためのツールが無いので、基本的に Linux の dd ユーティリティ・コマンドを使用します。

そのため Linux OS がインストールされた PC が必要です。

u-boot 領域を部分的にアップデートするには、Linux PC に SD カードをマウントし dd コマンドを実行します。

- a) **Linux PC に** SD カードを接続して以下のコマンドを実行すると、この例では sdb が見えるようになります（PC の SD カード・スロットで見えない場合は、USB カード・リーダー経由で試してください）。

```
$ sudo cat /proc/partitions ↵
```

8	16	7822336	sdb
8	17	512000	sdb1
8	18	1536000	sdb2
8	19	10240	sdb3

- b) 以下の dd コマンドを実行して SD カードの u-boot 領域にベアメタル・アプリケーションの実行バイナリ・ファイルを書き込みます。

```
$ sudo dd if=atlas_main.img.bin of=/dev/sdb3 bs=64k seek=4 ↵
```

- ② SoC EDS v14.0 以降 のバージョンでは、Embedded Command Shell から 以下の alt-boot-disk-util コマンドを実行して、SD カードの u-boot 領域に実行バイナリ・ファイル（この例では、atlas\_main.img.bin）を書き込みます。

 **注記:**

下記コマンドにおいて、-d F は Windows PC に挿した SD カードのドライブが F ドライブである場合の指定例です。SD カードのドライブは実際のご使用環境に合わせて指定してください。

```
$ alt-boot-disk-util -b atlas_main.img.bin -a write -d F ↓
```

- \_\_\_ 4. アプリケーションが FPGA 側にアクセスする場合は、FPGA をコンフィグレーションしておきます。
- \_\_\_ 5. SD カードをボードに取り付けて、ボードを WARM リセットすると、Preloader 実行後に ベアメタル・アプリケーションが自動的に起動します。



## 改版履歴

Revision	年月	概要
1	2016 年 5 月	初版、15.0
2	2018 年 9 月	① 書式変更 ② Windows® 10 使用の際の Preloader 生成における注記を追加 ③ リンク URL 修正
3	2019 年 1 月	① 18.0 に対応 ② DS-5 デバッグ・スクリプトの内容について追記 ③ 演習ファイルも 18.0 対応とし Atlas /DE10-Nano で使えるように変更 ④ 全体の文書構成を見直し

### 免責およびご利用上の注意

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

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