インテル® SoC FPGA All-in-One ベアメタル・アプリケーション サンプル

Ver.20.1

2021 年 5 月 Rev.3

目次

本書をお読みになる前に

この資料の内容は 2021 年 5 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. はじめに

このサンプルは、インテル® SoC FPGA 向けベアメタル・アプリケーションを構築する際のスタートポイントとして、ご使用いただけるサンプルです。

ハードウェア・ライブラリー（以降、HWLib ）などベアメタル開発に必要なコードをあらかじめプロジェクト内に配置しビルド対象にしているため、ユーザーは必要なヘッダーファイルをインクルードするだけで、Makefile の編集をせずに API をご使用いただくことが可能です。

また未使用の API に関しては、リンク時に対象外としておりますのでコードサイズに影響を与えません。

本書では、以下の内容を説明しています。

- 適用要件（対応バージョン、対応ボード）
- このサンプルを使用する利点
- サンプルのディレクトリー／ファイル構成
- コンパイル設定
- サンプルの基本動作
- コマンドの追加方法
- このサンプルのメインルーチン・ソースコードの説明
- 便利なユーティリティー関数の紹介
- HWLib（ハードウェア・ライブラリー）とは
- HWLib Examples

2. このサンプルを使用する利点

通常のベアメタル・サンプル・アプリケーションでは、該当のインターフェース用の HWLib のみが使用される構成になっており、他の HWLib を使用するためには Makefile を修正して、追加の HWLib ソースを指定する必要があります。

また、 Makefile プロジェクトで提供されるため、ユーザーが追加したソースファイルについても Makefile に追記する必要があり、 インテル SoC FPGA のソフトウェア開発フローを熟知していない方にとって理解するのに時間を要すものでした。

このサンプルでは、HWLib として提供されるソースがすべて登録済みとなっており、使用したい HWLib のヘッダーファイルをインクルードすれば、すべての API を使用することができるようにしてあります。

また、プロジェクトの TOP ディレクトリーに追加されたソースファイルは、すべてコンパイル対象にする状態としてありますので、基本的に Makefile を修正すること無く、各種評価が開始できるようになっています。

3. 使用環境

3-1. 開発環境

この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 3-1】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® 10（64 bit）搭載の 64 bit マシン 本資料では、Windows® 10 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（以降、Quartus Prime）	SoC FPGA のハードウェアを開発するためのツールです。 この資料では、Quartus Prime スタンダード・エディション開発ソフトウェア v20.1 を使用しています。 ■ Quartus Prime スタンダード・エディション v20.1 ⚠ 注記： 使用するターゲットボードに搭載されている SoC FPGA に対応した Device データ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。 Quartus® Prime & ModelSim® インストール方法 ⓘ Note: この資料では、「4-6. FPGA のコンフィグレーション」において、ハードウェア・デザイン（sof ファイル）を FPGA にダウンロードするために Quartus Prime Programmer のみを使用しています。
3	インテル® SoC FPGA エンベデッド開発スイート スタンダード・エディション（以降、SoC EDS）	SoC FPGA のソフトウェアを開発するためのツールです。 SoC EDS とは別に、Arm® Development Studio Intel® SoC FPGA Edition（以下、Arm DS）のインストールも必要となります。 Arm DS は旧製品 Arm® Development Studio 5 Intel® SoC FPGA Edition（以下、DS-5）の後継であり同様の機能を提供します。 Arm DS を使用することで、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。 この資料では、SoC EDS スタンダード・エディション v20.1 を使用しています。 ■ SoC EDS スタンダード・エディション v20.1 Ⓟ Point: SoC EDS および Arm DS のインストール方法に関しては以下のサイトをご参照ください。 SoC EDS 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 ⚠ 注記： インテル® FPGA ダウンロード・ケーブル II（以降、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、Arm DS / DS-5 Intel® SoC FPGA Edition（有償版）が必要になります。 ⚠ 注記： 本バージョンのサンプル・プロジェクトは、Arm DS の使用を前提としています。DS-5 で使用したい場合には旧バージョンのサンプル・プロジェクトをご使用ください。
4	ターミナル・エミュレーション・ソフトウェア	このサンプルを使用するためには、シリアル・ターミナル・ソフトが必要です。 この資料では、「Tera Term」と呼ばれるフリーウェア・ソフトを使用しています。 ■ Tera Term のダウンロード URL ⚠ 注記： Tera Term では、ターゲットボードの UART と接続した際の有効な COM ポートに対して、以下の設定を行ってください。 ・ ボーレート 115200 bps ・ 8 ビットデータ ・ パリティなし ・ 1 ストップビット ・ フロー制御なし

3-2. 対応ターゲットボード

このサンプルでは、下記のターゲットボードが config.mk ファイル内の TARGET_BOARD にて指定可能です。

【表 3-2】このサンプルの対応ターゲットボード

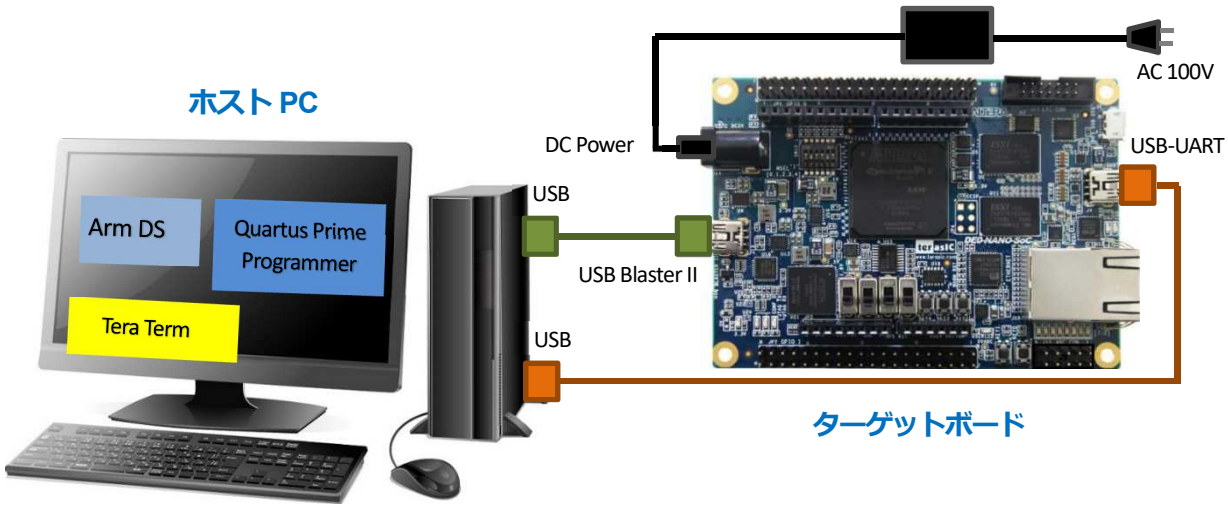
項番	ターゲットボード
1	Cyclone® V SoC 開発キット
2	インテル® Arria® 10 SoC 開発キット
3	Helio - Cyclone® V SoC キット（販売終了）
4	Sodia - Cyclone® V ST SoC 評価ボード
5	DE0-Nano-SoC Kit / Atlas-SoC Kit（販売終了）
6	DE10-Nano Kit

ⓘ Note：
本バージョンのサンプル・プロジェクトには、Helio ボード用のデータは含まれておりません。
Helio でご使用されたい場合は、旧バージョンのサンプル・プロジェクトをご使用ください。

4. このサンプルの使用方法

4-1. ターゲットボードの接続

下記にターゲットボードの接続の概要を示します。



【図 4-1】 ターゲットボードの接続

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）をターゲットボードの DC 入力コネクターに接続します。
- USB ケーブルでホスト PC とターゲットボードのオン・ボード USB-Blaster™ II コネクターを接続します。
- USB ケーブルでホスト PC とターゲットボードの USB-UART コネクターを接続します。

⚠ 注記：
Sodia ボード使用時において、 FPGA のコンフィグレーション（.sof ファイルの書き込み）およびベアメタル・アプリケーションのデバッグ・実行を行うには、別途 USB-Blaster™ II ケーブルが必要になります。
https://www.mouser.jp/ProductDetail/Intel-Altera/PL-USB2-BLASTER?qs=%2fha2pyFaduiRriBk7xhlgC3%2fgy%2faENSICIUHybT1yeHhqUx7xdgjQg==&_ga=2.221605659.284231058.1547025155-

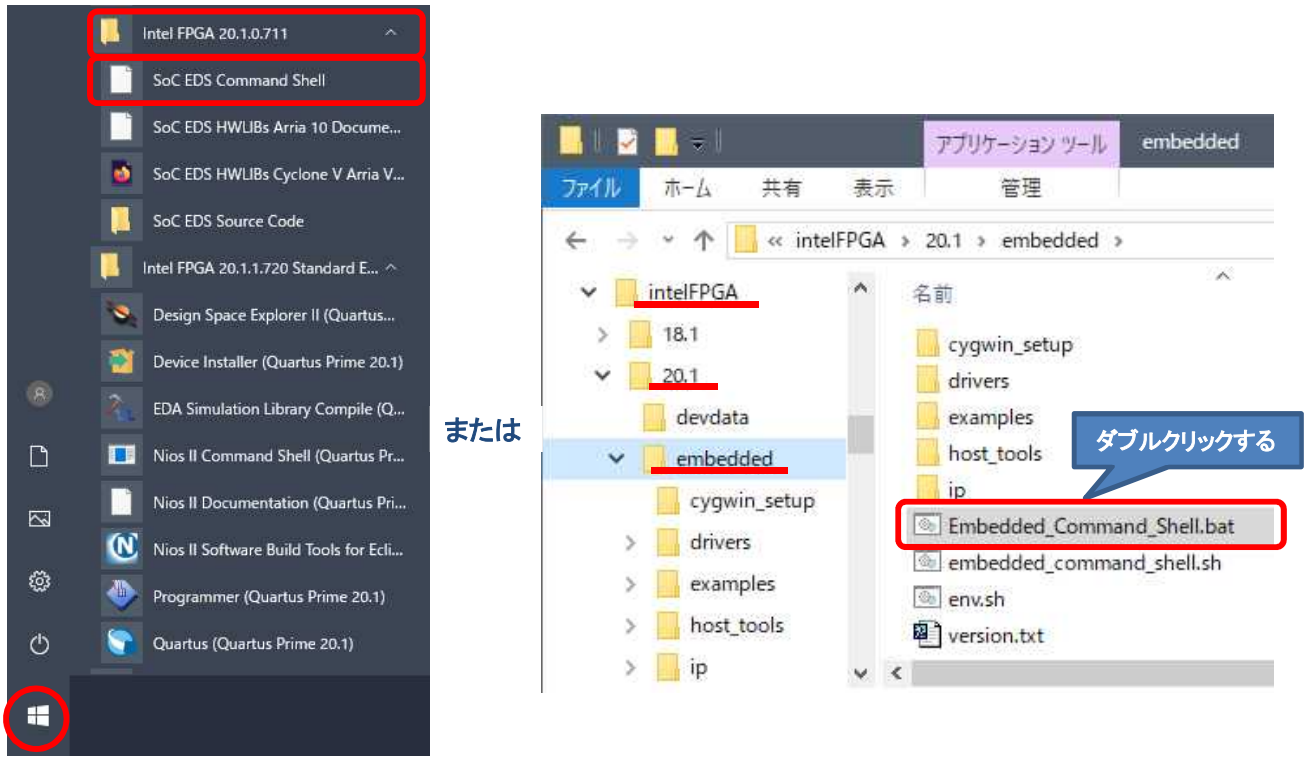
4-2. Arm DS の起動とサンプルプログラムのインポート

Arm DS を起動し、サンプル **ALT-HWLib-All-In-One_v20.1_r○.○.tgz** をインポートします。

SoC EDS に対する各種環境設定を自動的に実施するために、Arm DS 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

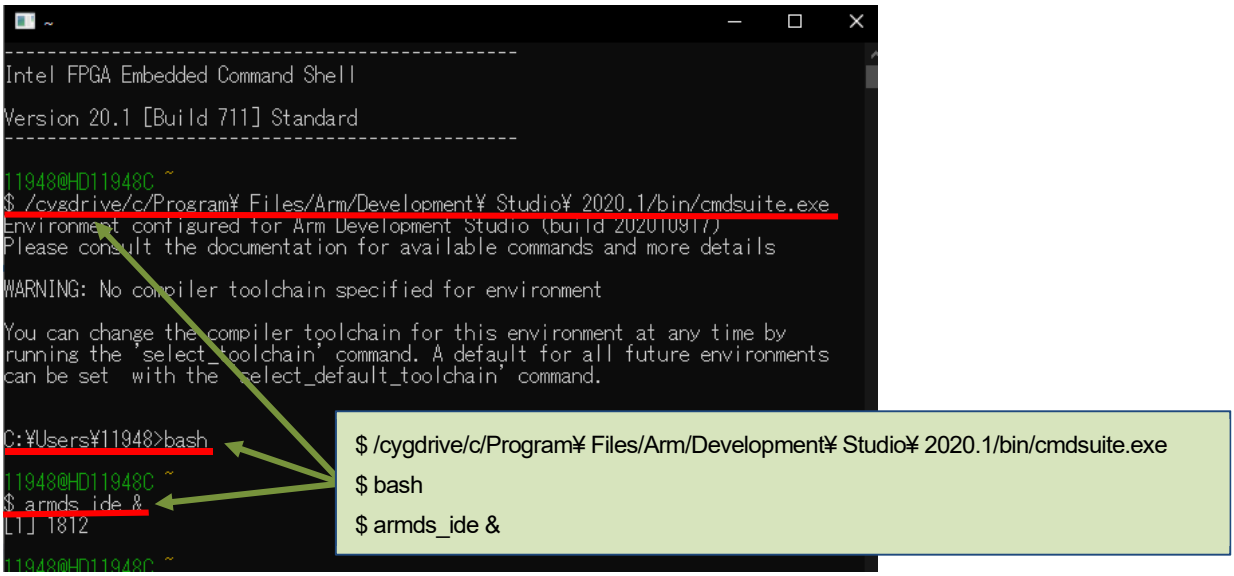
4-2-2. Arm DS の起動

____ 1. Embedded Command Shell のウィンドウが開いたら下記 1~3 のコマンドを入力して Arm DS を起動します。

① $ /cygdrive/c/Program¥ Files/Arm/Development¥ Studio¥ 2020.1/bin/cmdsuite.exe
※Arm DS のバージョンにより 2020.1 の部分は異なります。

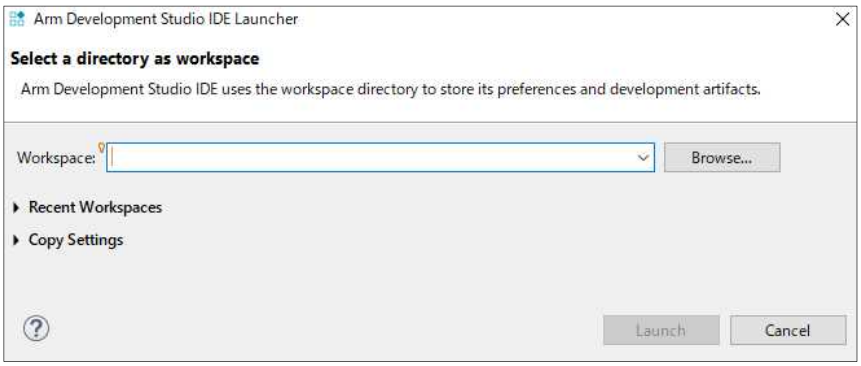
② $ bash

③ $ armds_ide &



【図 4-3】 Arm DS 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。

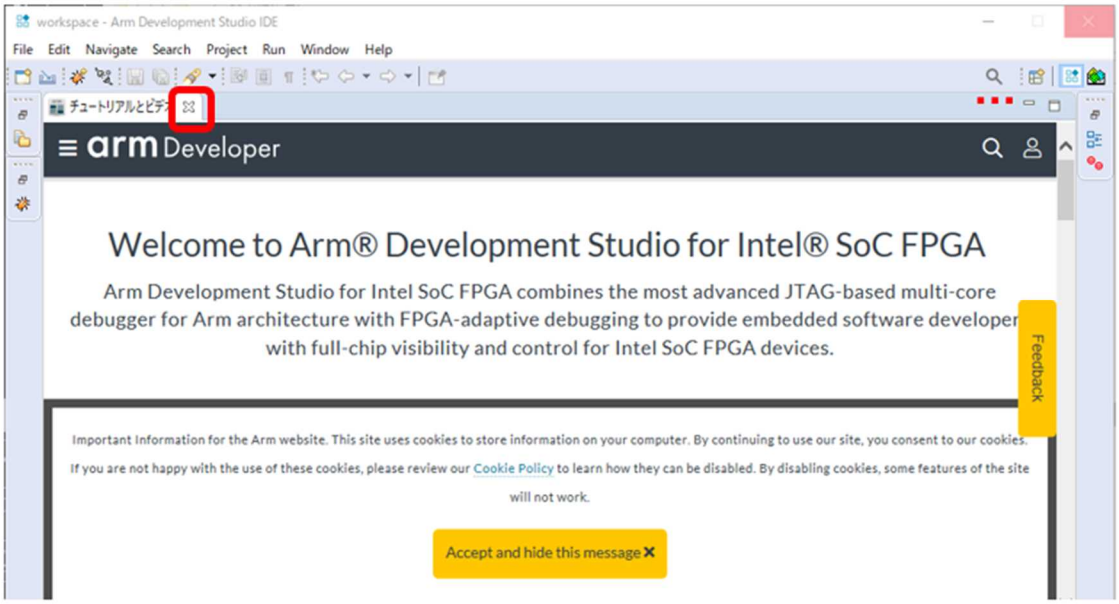


【図 4-4】 Arm DS のワークスペースの指定

____3. Arm DS 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。

尚、× マークを押して閉じるまで少し時間がかかる場合があります。

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。



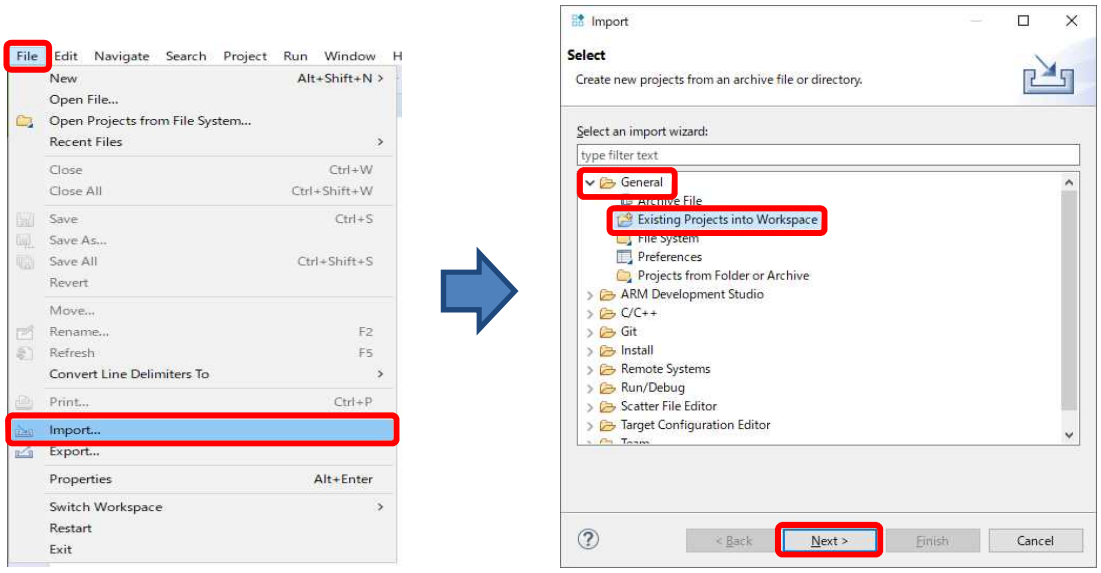
【図 4-5】 Arm DS ウェルカム画面

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

ベアメタル・サンプル・アプリケーション **ALT-HWLib-All-In-One_v20.1_r○.○** を Arm DS にインポートします。

____1. DS のメニューから「**File**」⇒「**Import**」を選択します。

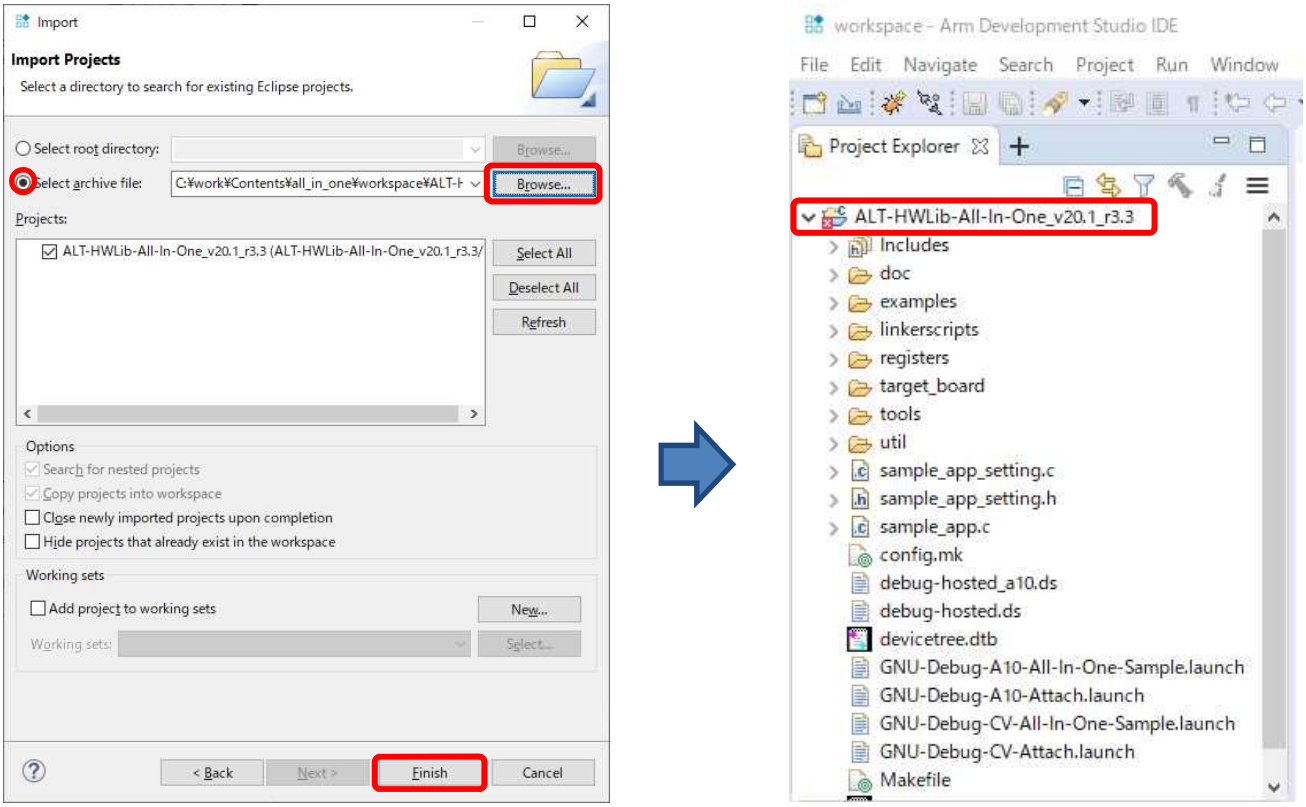
____2. 「**General**」⇒「**Existing Projects into Workspace**」を選択し、[**Next**] をクリックします。



【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。 **ALT-HWLib-All-In-One_v20.1_r○.○.tgz** 選択後、［**終了(F)**］ボタンを押します。

____4. Arm DS 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 **ALT-HWLib-All-In-One_v20.1_r○.○** が追加され、ALT-HWLib-All-In-One_v20.1_r○.○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 4-7】 サンプル・アプリケーションの追加

4-4. コンパイルの設定

このサンプルでは使用するコンパイラー、ターゲットボードの指定、およびセミホスティングの使用有無を **config.mk** ファイルにて指定可能です。

デフォルト設定としては“GNU コンパイラー”、“UART 出力（非セミホスティング）”、“システム・ヘッダーファイルを生成する”、“atlas ボード”に設定してあります。

また、DMA 関連のサンプル（sample_dma_mem.c / sample_dmac.c ）を使用する際には、USED_DMA を 1 に設定してビルドしてください。使用しない場合は USED_DMA を 0 に設定してビルドしてください。

```
##################
# Target App Name #
##################
TARGET := sample_app

###################################
# Select Compiler < Support GNU only #
###################################
COMPILER := GNU
#COMPILER := ARM

#############################################
# Select printf target (0:Semihost / 1:UART) #
#############################################
SEMIHOSTED := 1

#########################################################
# Generate system header file from sopcinfo (0:No / 1:Yes) #
#########################################################
GEN_SYS_HEADER := 1

######################
# Select Target Board #
######################
TARGET_BOARD := atlas
#TARGET_BOARD := sodia
#TARGET_BOARD := c5socdk
#TARGET_BOARD := a10socdk
#TARGET_BOARD := de10nano

##############################
# USE DMA sample (0:No / 1:Yes)#
##############################
USED_DMA := 0
```

コンパイラーの指定： GCC／ARMCC

printf 出力の指定： 0=セミホスティング／1=UART

システム・ヘッダーファイルの生成:
0=生成しない ／ 1=生成する

ターゲットボードの指定:
- atlas
- sodia
- c5socdk
- a10socdk
- de10nano

DMA 関連のサンプル（sample_dma_mem.c / sample_dmac.c）を使用:
0=使用しない
1=使用する

【リスト 4-1】 config.mk ファイルでのコンパイル設定

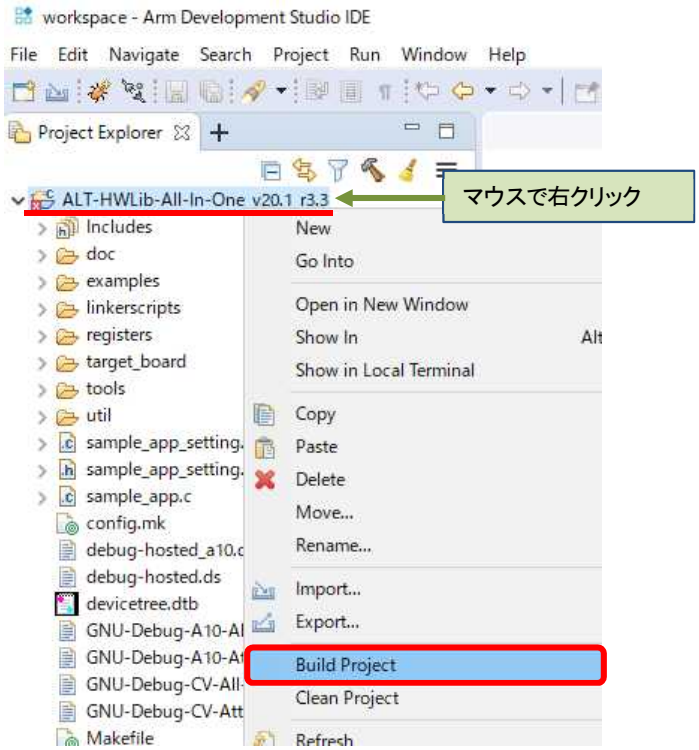
4-5. ベアメタル・サンプル・アプリケーションのビルド

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。

4-5-1. プロジェクトのビルド

ベアメタル・サンプル・アプリケーション・プロジェクト **ALT-HWLib-All-In-One_v20.1_r○.○** 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**が生成されます。

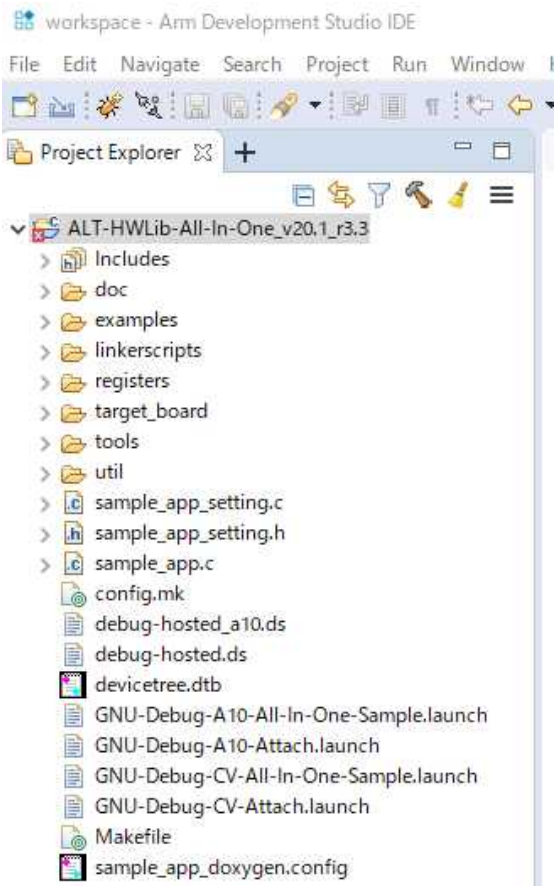


【図 4-8】 プロジェクトのビルド

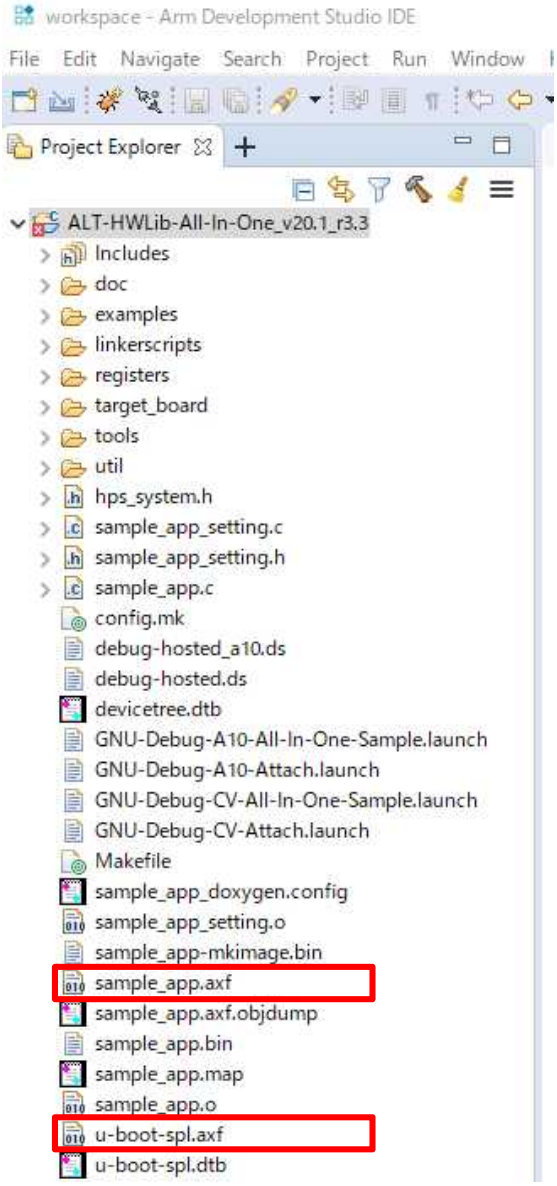
4-5-2. ベアメタル・サンプル・アプリケーション・プロジェクトのファイル構成

サンプル・アプリケーションのファイル構成を下図に示します。

ビルド前



ビルド後



【図 4-9】 サンプル・アプリケーションのファイル構成

- **sample_app.axf** ： サンプル・アプリケーションの実行可能バイナリーです。
- **u-boot-spl.axf** ： Preloader の実行可能バイナリーです。

4-6. FPGA のコンフィグレーション

次に **.sof** という拡張子のハードウェア・デザインファイルを 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。

4-6-1. 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

5. 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。

ターゲットボードの接続「4-1 ターゲットボードの接続」のセクションを参照し、ボードの接続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、 ボードに AC アダプターを接続して電源を投入してください。

____ 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。

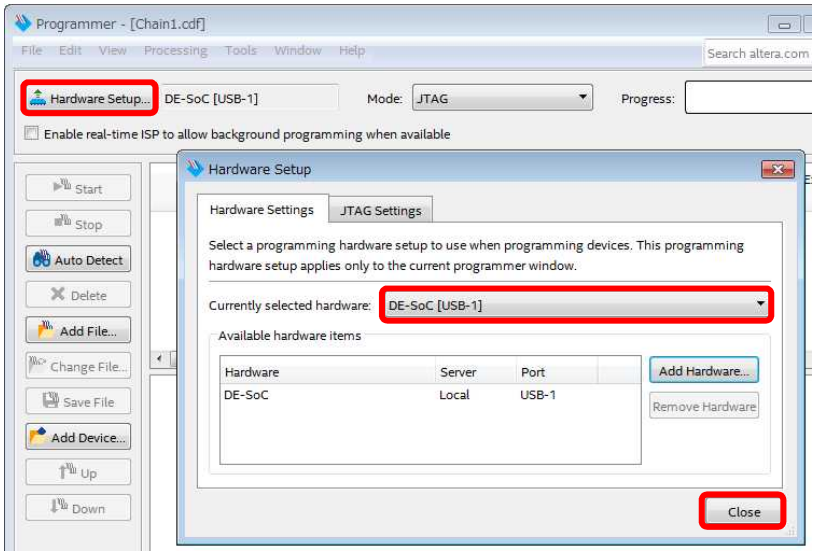
____ 2. **Programmer** 内にある［**Hardware Setup**］ボタンをクリックし、Hardware Setup ウィンドウ内の **Currently selected hardware** のプルダウンリストからプログラミング・ハードウェアを選択し、ウィンドウを Close します。

ⓘ **Note**:

*本資料の説明では、**Atlas-SoC** ボードを例として説明しているので、プログラミング・ハードウェアは下図のように **DE-SoC** を選択しています。*
プログラミング・ハードウェアはご使用のターゲットボードに応じて下表のハードウェアを選択します。

【表 5-1】ターゲットボードに対応したプログラミング・ハードウェア

項番	ターゲットボード	プログラミング・ハードウェア
1	Cyclone® V SoC 開発キット	USB-BlasterII [USB-x]
2	インテル® Arria® 10 SoC 開発キット	USB-BlasterII [USB-x]
3	Sodia – Cyclone® V ST SoC 評価ボード	USB-BlasterII [USB-x]
4	DE0-Nano-SoC ボード / Atlas-SoC ボード	DE-SoC [USB-x]
5	DE10-Nano ボード	DE-SoC [USB-x]



【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。

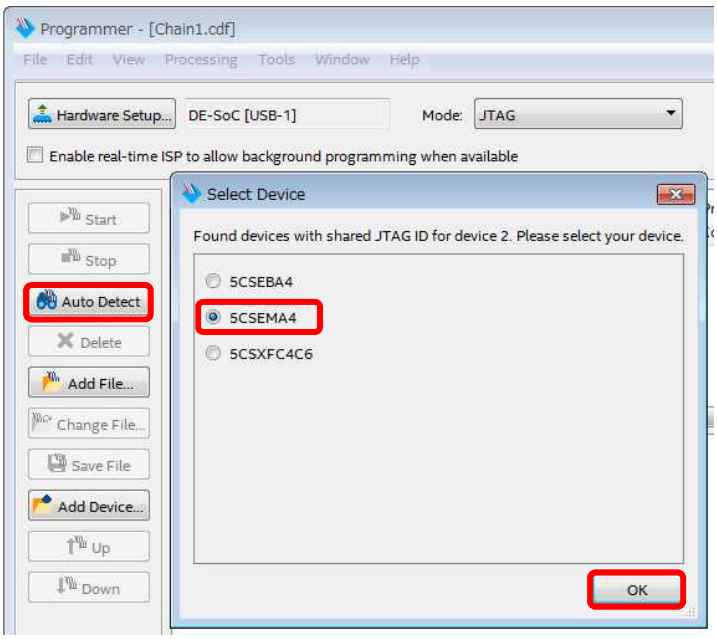
____4. Select Device ウィンドウからターゲットボードに搭載されているデバイスを選択し、[OK] をクリックします。

ⓘ Note:

*本資料の説明では、**Atlas-SoC** ボードを例として説明しているので、デバイスは下図のように **5CSEMA4** を選択しています。*
デバイスはご使用のターゲットボードに応じて下表のデバイスを選択します。

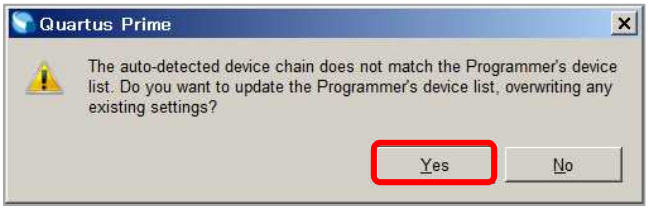
【表 5-2】ターゲットボードに対応したデバイス

項番	ターゲットボード	デバイス
1	Cyclone® V SoC 開発キット	5CSXFC6
2	インテル® Arria® 10 SoC 開発キット	10AS066N3
3	Sodia – Cyclone® V ST SoC 評価ボード	5CSTFD6
4	DE0-Nano-SoC ボード / Atlas-SoC ボード	5CSEMA4
5	DE10-Nano ボード	5CSEBA6



【図 5-2】 デバイスの選択（Atlas-SoC ボードの場合の例）

____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 チェイン上に SOCVHPS と 5CSMA4 が表示されます。SOCVHPS は HPS 側、5CSMA4 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。

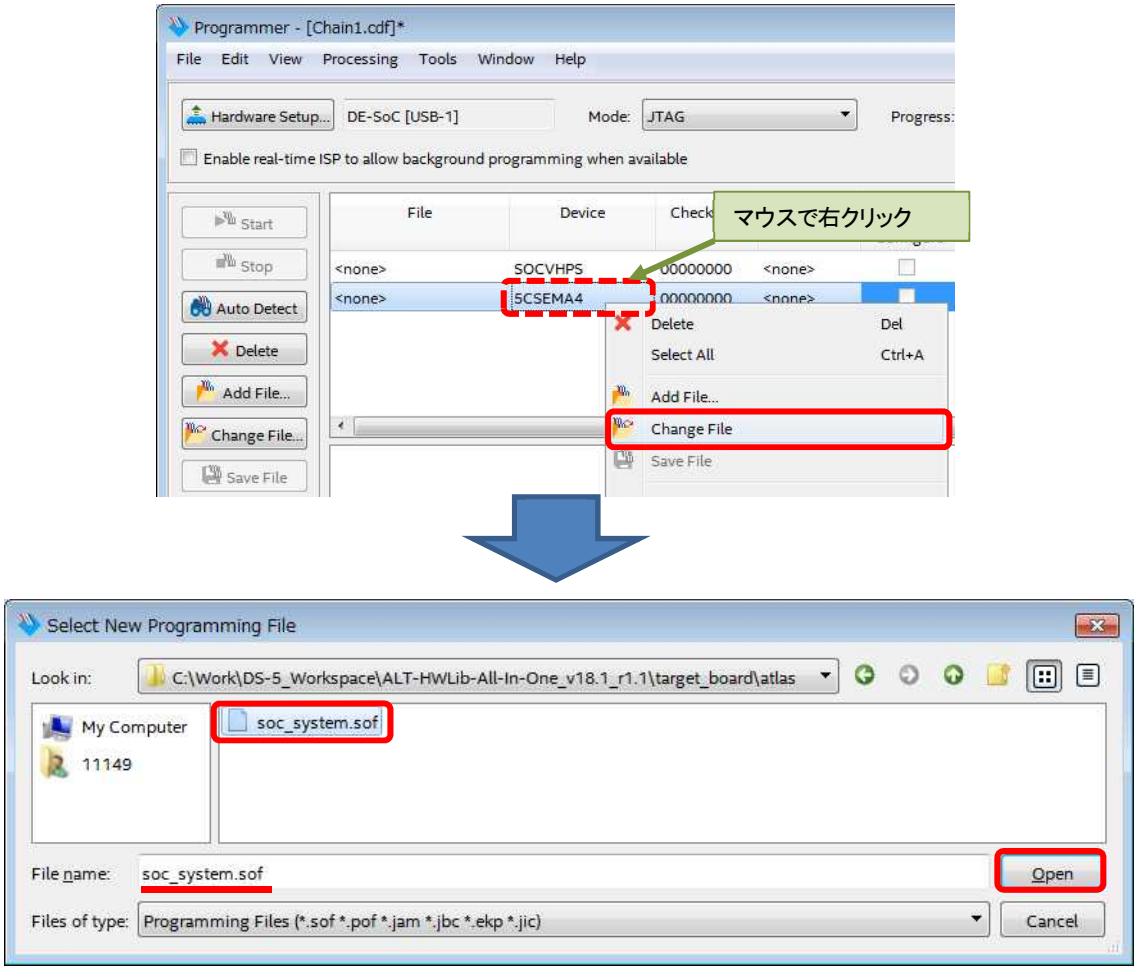
Device 欄の 5CSEMA4 上で右クリックし、「**Change File**」をクリックします。
Select New Programming File ダイアログ・ボックスにおいて、ターゲットボードに対応した **.sof** ファイルを選択します。

ⓘ Note:

*各ターゲットボードに対応した .sof ファイルは、本サンプル・プロジェクトの **target_board** ディレクトリーの下にあります。*
.sof ファイルはご使用のターゲットボードに応じて下表のファイルを選択します。

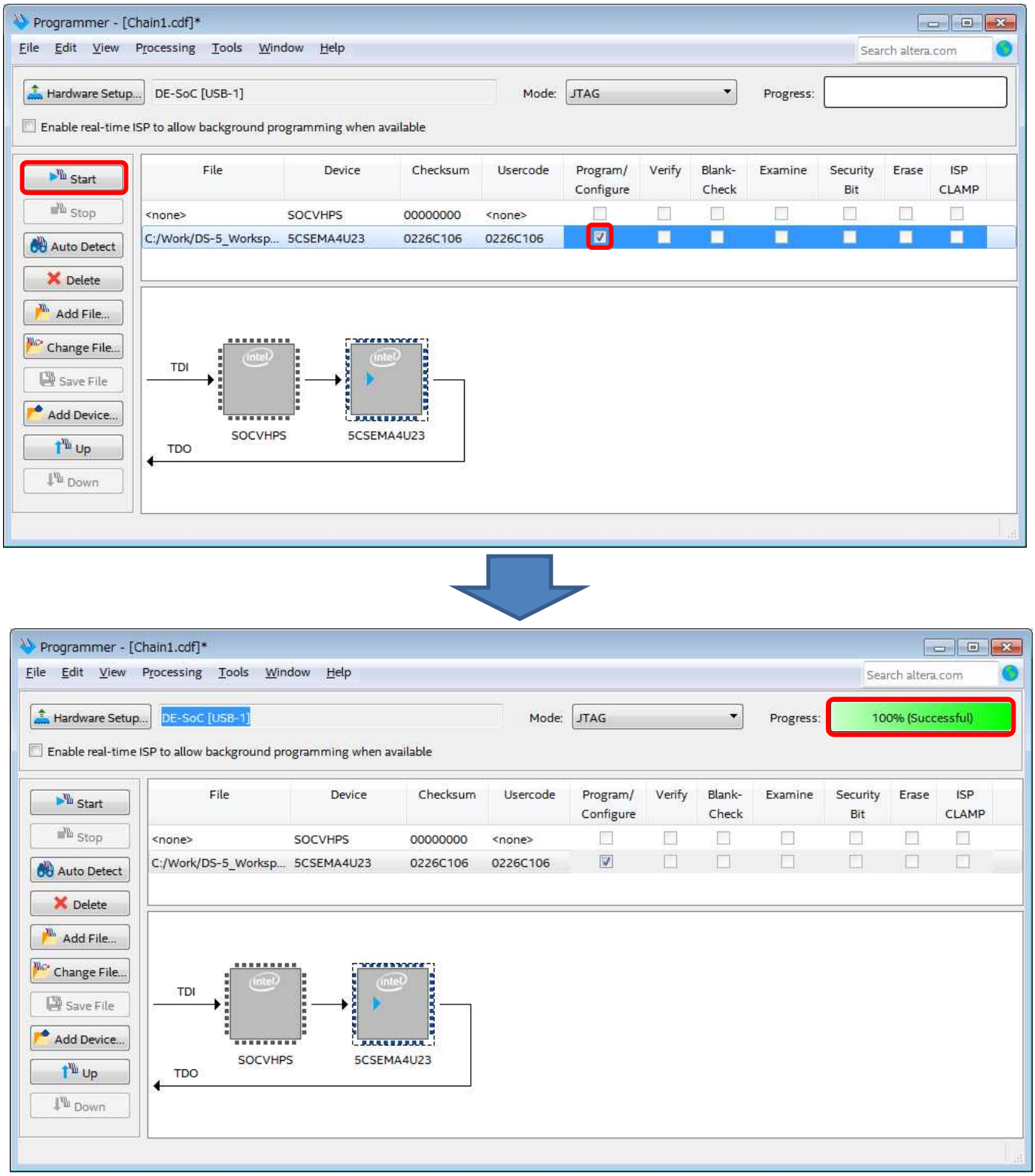
【表 5-3】ターゲットボードに対応した .sof ファイル

項番	ターゲットボード	.sof ファイル
1	Cyclone® V SoC 開発キット	c5socdk¥**soc_system.sof**
2	インテル® Arria® 10 SoC 開発キット	a10socdk¥**ghrd_10as066n2.sof**
3	Sodia – Cyclone® V ST SoC 評価ボード	sodia¥**soc_system.sof**
4	DE0-Nano-SoC ボード / Atlas-SoC ボード	atlas¥**soc_system.sof**
5	DE10-Nano ボード	de10nano¥**soc_system.sof**



【図 5-4】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 5-5】 sof のダウンロード（Atlas-SoC ボードの場合の例）

5-1. 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。

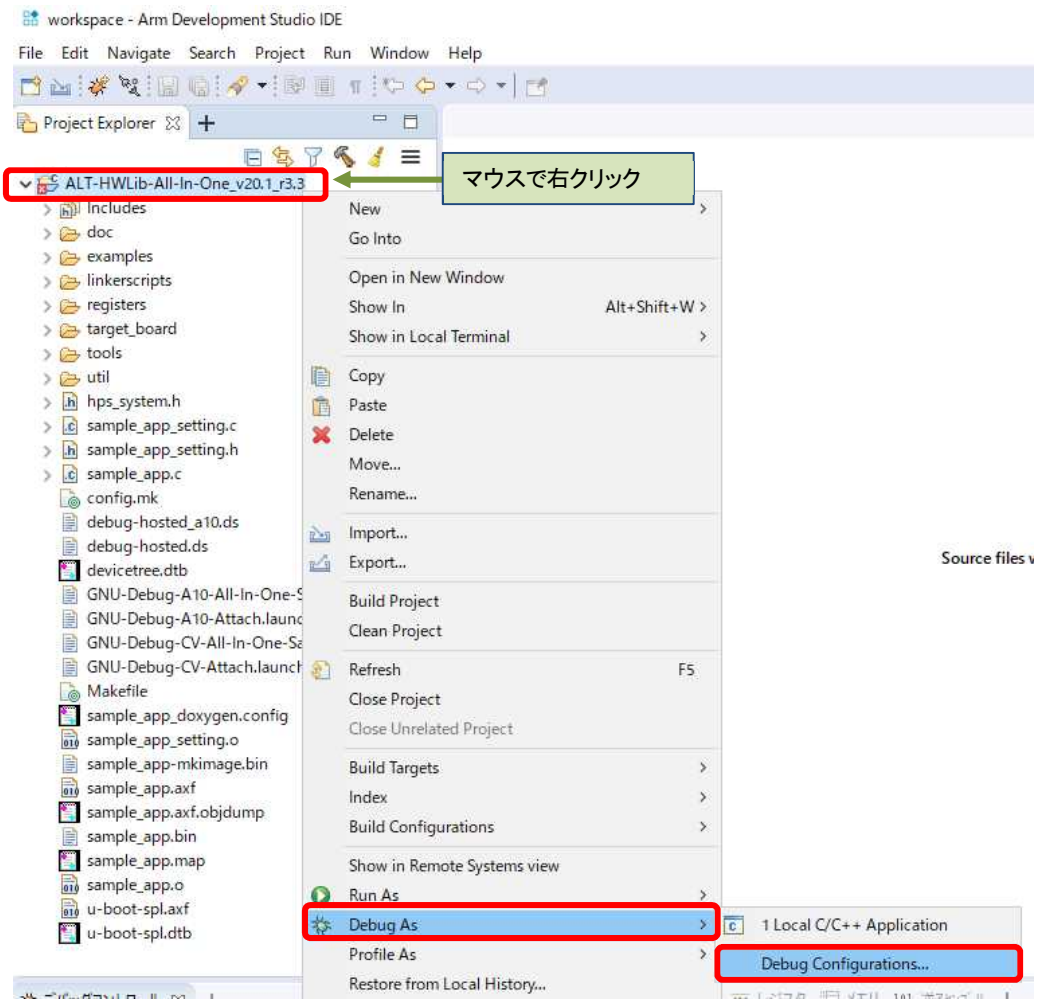
デバッグを実行する前に、「4-1 ターゲットボードの接続」のセクションを参照し、ホスト PC とターゲットボード間のケーブル接続、およびターゲットボードへの電源投入が完了していることを再度確認してください。

また、シリアル・ターミナル（この資料の説明では、Tera Term を使用）を起動して、ターゲットボードの UART と接続した有効な COM ポートに対して以下の設定を行い、ターミナル入出力が可能な状態にしておきます。

- ボーレート 115200 bps
- 8 ビットデータ
- パリティなし
- 1 ストップビット
- フロー制御なし

5-1-1. デバッグの実行

____ 1. ベアメタル・サンプル・アプリケーション・プロジェクト **ALT-HWLib-All-In-One_v20.1_r○.○** 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 5-6】「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、
「**汎用 ARM C/C++ アプリケーション**」⇒「**GNU-Debug-<device>-All-In-One-Sample**」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）。

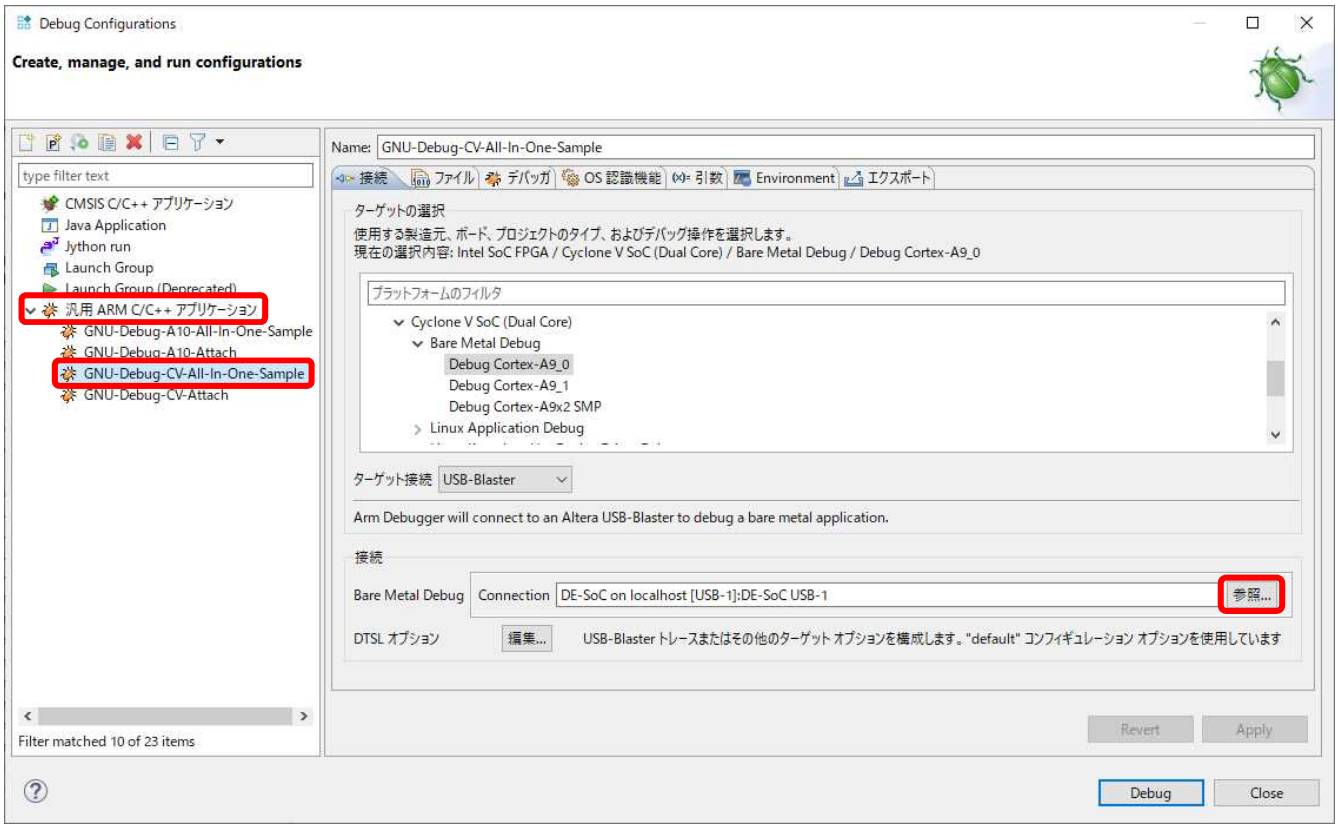
ⓘ **Note**:

*本資料の説明では、**Atlas-SoC** ボードを例として説明しているので、デバッグ構成は下図のように「**汎用 ARM C/C++ アプリケーション**」⇒「**GNU-Debug-CV-All-In-One-Sample**」を選択しています。*
*このデバッグ構成は、ターゲット接続に USB-Blaster™ II 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。*
デバッグ構成はご使用のターゲットボードに応じて下表のものを選択します。

【表 5-4】ターゲットボードに対応したデバッグ構成

項番	ターゲットボード	デバッグ構成
1	インテル® Arria® 10 SoC 開発キット	GNU-Debug-A10-All-In-One-Sample
2	Cyclone® V SoC 開発キット	GNU-Debug-CV-All-In-One-Sample
3	Sodia – Cyclone® V ST SoC 評価ボード	
4	DE0-Nano-SoC ボード / Atlas-SoC ボード	
5	DE10-Nano ボード	

____ 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 5-7】 デバッグの構成

____4. 接続ブラウザ・ウィンドウで、目的のターゲット接続をハイライトして、[**選択**] をクリックします。

ⓘ **Note**:

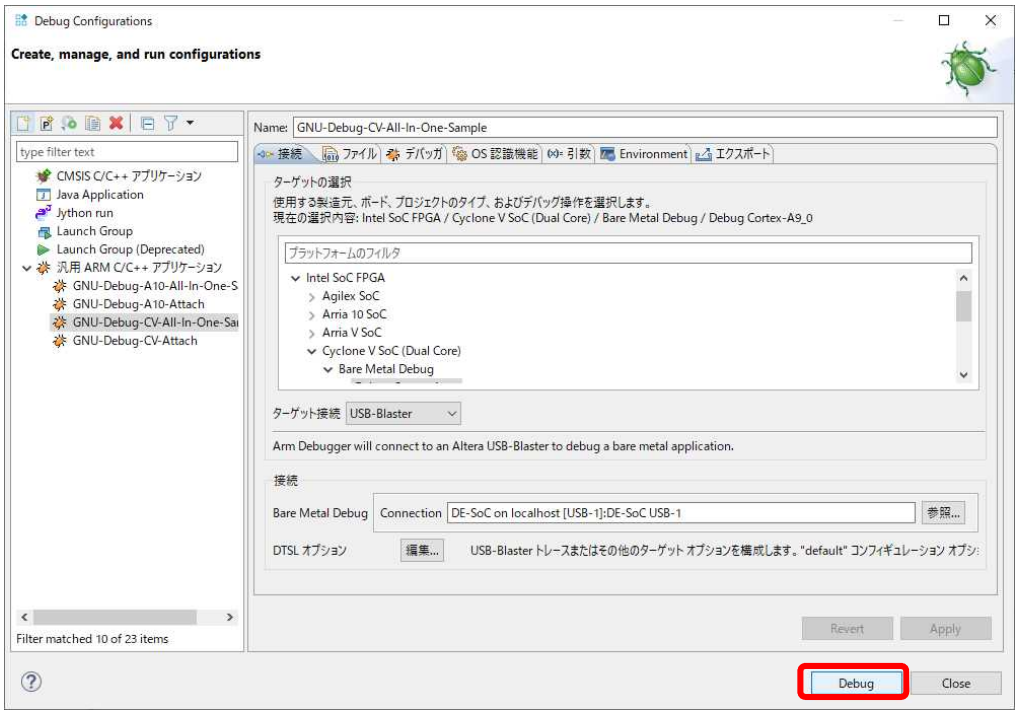
*本資料の説明では、**Atlas-SoC** ボードを例として説明しているので、ターゲット接続は下図のように **DE-SoC on localhost** を選択しています。*
ターゲット接続はご使用のターゲットボードに応じて下表を選択します。

【表 5-5】ターゲットボードに対応したターゲット接続

項番	ターゲットボード	ターゲット接続
1	Cyclone® V SoC 開発キット	USB-BlasterII on localhost
2	インテル® Arria® 10 SoC 開発キット	USB-BlasterII on localhost
3	Sodia – Cyclone® V ST SoC 評価ボード	USB-BlasterII on localhost
4	DE0-Nano-SoC ボード / Atlas-SoC ボード	DE-SoC on localhost
5	DE10-Nano ボード	DE-SoC on localhost

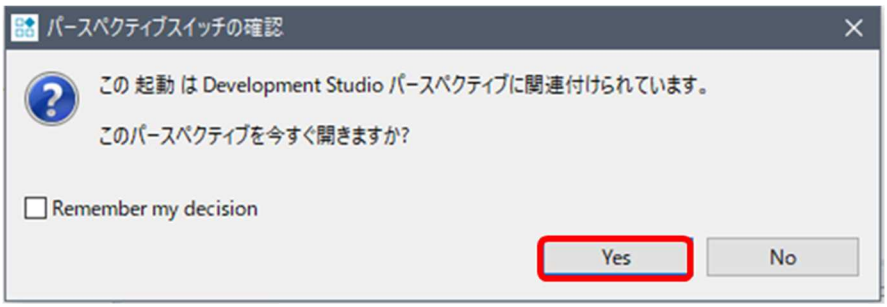
【図 5-8】 デバッグ・ケーブルの選択

____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。



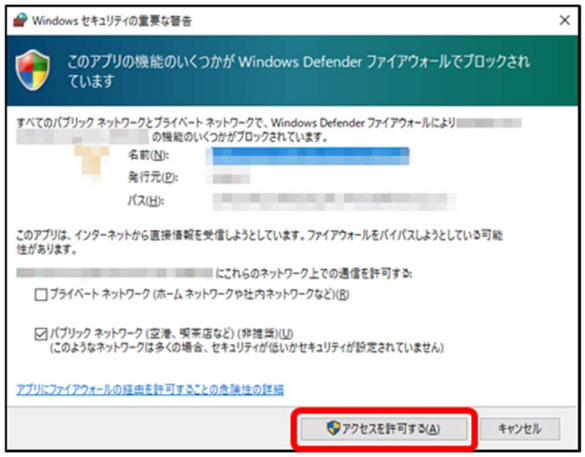
【図 5-9】 デバッグの実行

____ 6. 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 5-10】 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。

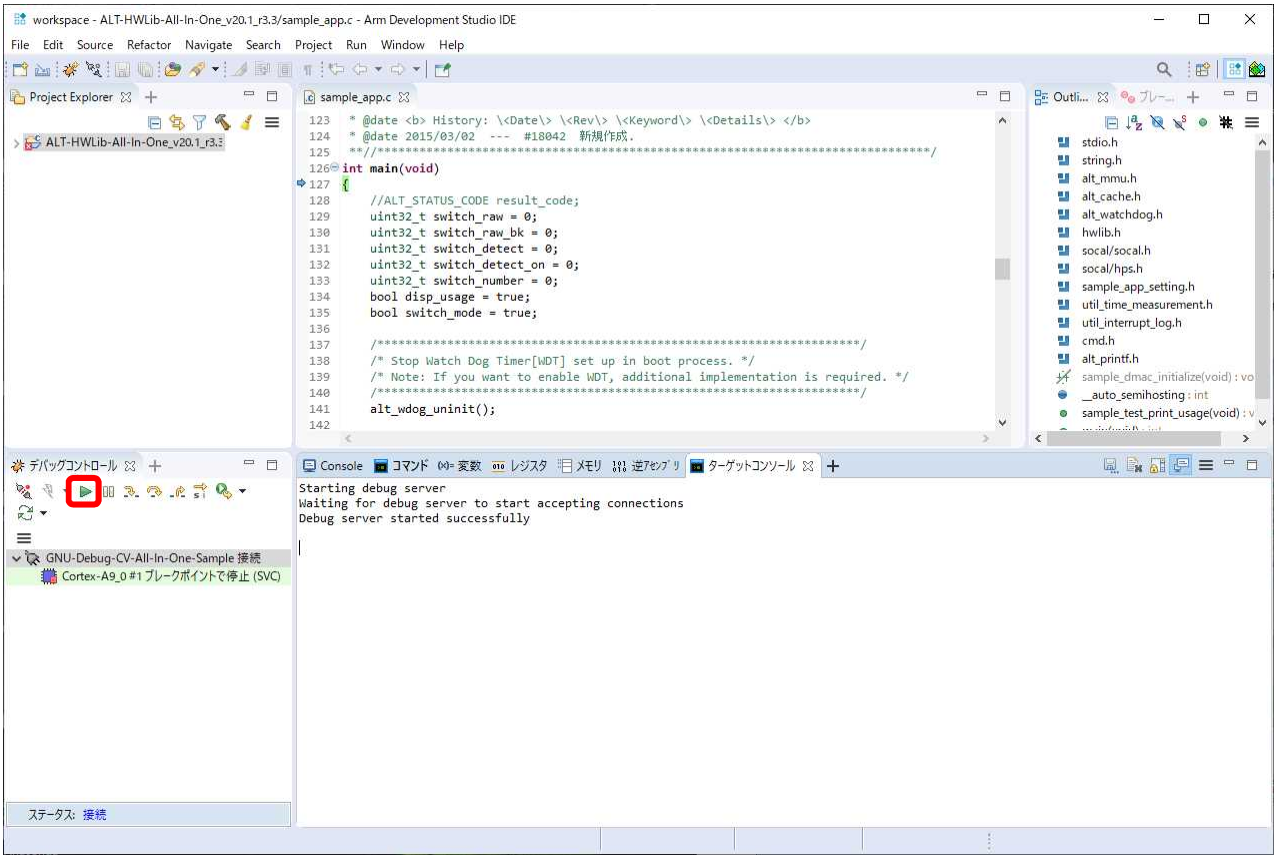


【図 5-11】 セキュリティの警告

> ⓘ **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・インタフェース（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。

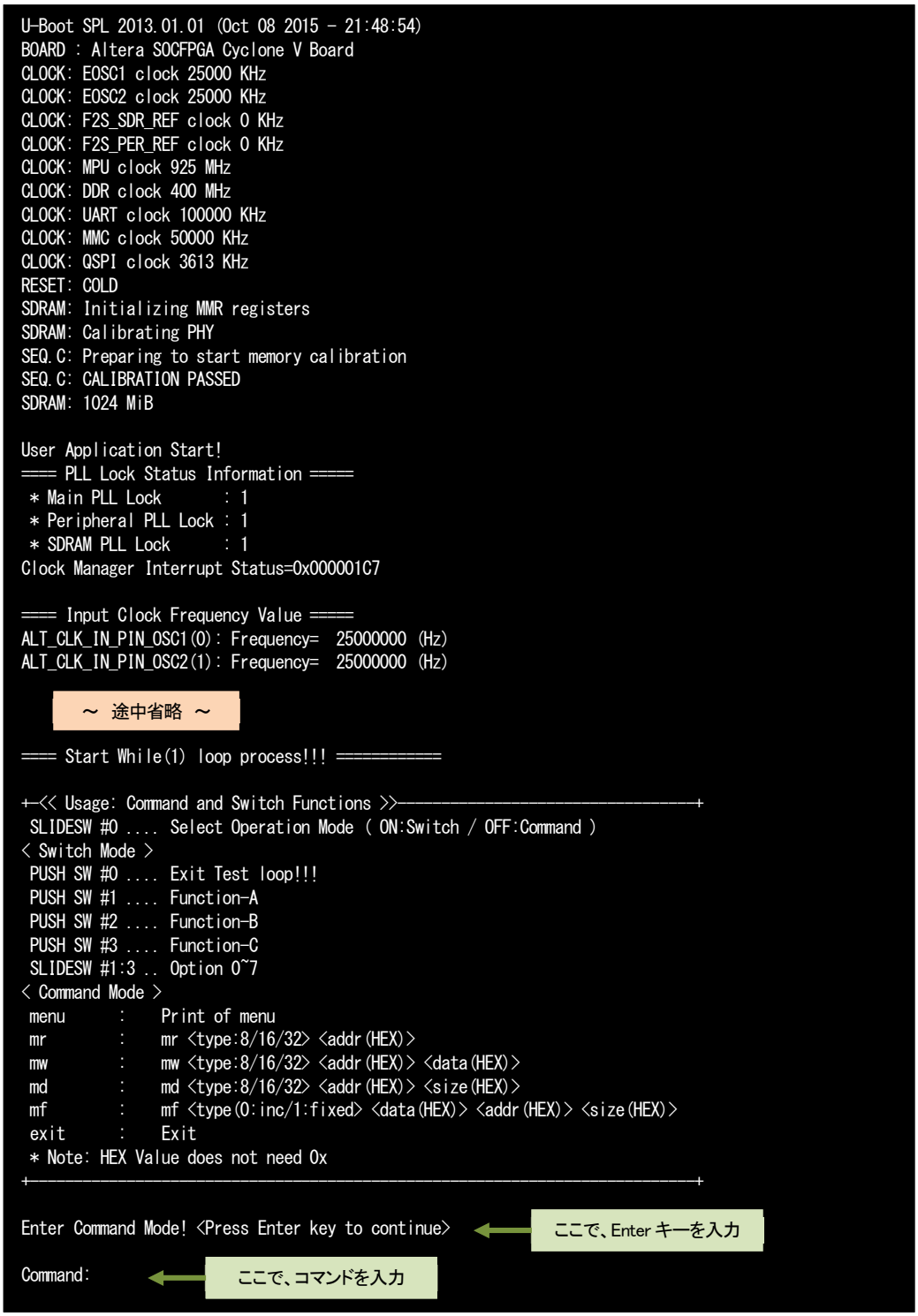
デバッガは起動スクリプトの指示に従いセミホスティング機能を有効にした後、JTAG を経由してアプリケーションをボードにダウンロードします。プログラム・カウンターが main 関数に到達するとブレークされデバッグが開始できる状態となります。この段階で、Arm DS のすべてのデバッグ機能を使用することができます（レジスターや変数の表示と編集、逆アセンブリ・コードの参照、など）。

____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、ターゲットボードの UART と接続したターミナル（Tera Term）の有効な COM ポートに対して、ベアメタル・サンプル・アプリケーションの実行内容が表示されます。



【図 5-12】 アプリケーションの実行／デバッグ

____8. ターゲットボードの DIP スイッチを「Command モード」（後述）に設定した場合は、下図のようにターミナル（Tera Term）にコマンドメニューを表示して、コマンド入力待ち状態になります。



【図 5-13】 ベアメタル・サンプル・アプリケーションの実行内容（Command モード時）

____9. デバッグを終了するには、「**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。

____10. 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。

6. このサンプルの基本動作

ターゲットボード上の DIP スイッチ の bit 0 の状態により、オペレーションを以下の 2 つのモードのいずれかに切り替えて検証することが可能です。

① ON : **Switch モード**（DIP スイッチ / PUSH スイッチ操作の確認）

② OFF : **Command モード**（コマンド入力による各種テストの実行）

6-1. Switch モード

DIP スイッチと PUSH スイッチの設定を確認して、設定状態をメッセージ出力します。

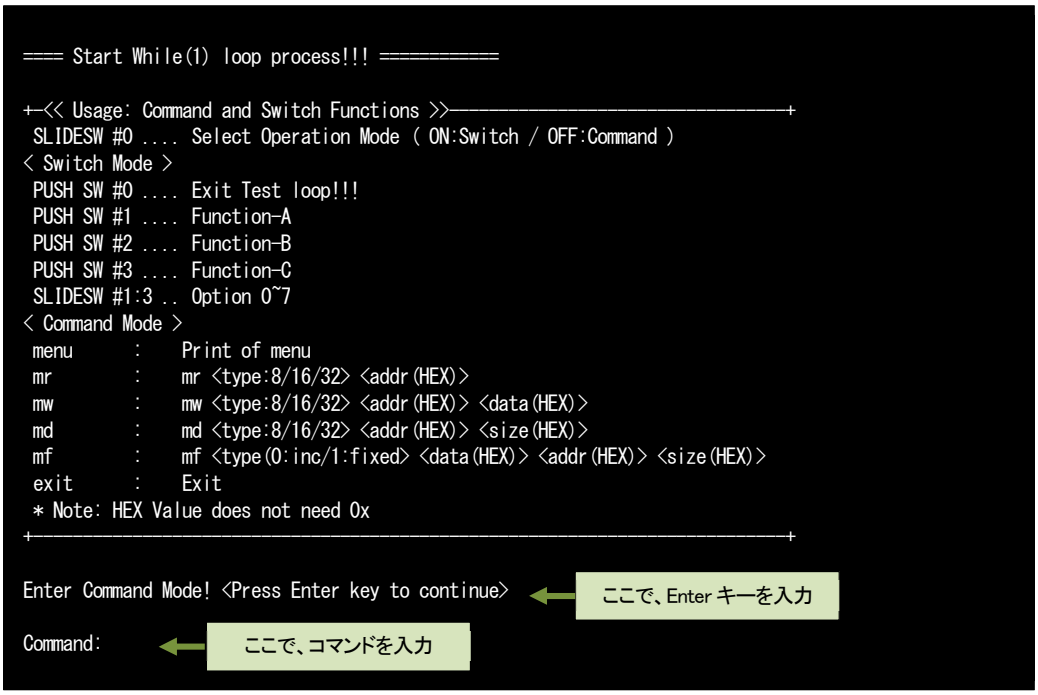
6-2. Command モード

COMMANDS_LIST commands[] に登録されているコマンドを実行可能です。デフォルトでは以下のコマンドが登録されています。

ユーザーが作成した処理をコマンドとして登録し実行させることも可能です。

【表 6-1】 デフォルトで登録されているコマンド

コマンド	コマンドライン	実行関数
メニュー表示コマンド	**menu**	cmd_menu()
メモリーリード・コマンド	**mr** <type:8/16/32> <addr(HEX)>	cmd_mem_read()
メモリーライト・コマンド	**mw** <type:8/16/32> <addr(HEX)> <data(HEX)>	cmd_mem_write()
メモリーダンプ・コマンド	**md** <type:8/16/32> <addr(HEX)> <size(HEX)>	cmd_mem_dump()
メモリーフィル・コマンド	**mf** <type(0:inc/1:fixed> <data(HEX)> <addr(HEX)> <size(HEX)>	cmd_mem_fill()
終了コマンド	**exit**	cmd_exit()



【図 6-1】 デフォルトでの Command モード・メニュー表示

7. このサンプルのメインルーチン・ソースコードの説明

以下は、このサンプルのメインルーチンの抜粋です（sample_app.c 内にあります）。

このサンプルのメインルーチン・ソースコードについて説明します。

```
/************************************************************************************
 *  includes
 ************************************************************************************/
#include <stdio.h>
#include <string.h>
#include "hwlib.h"
#include "socal/socal.h"
#include "socal/hps.h"
#include "sample_app_setting.h"
#include "util_time_measurement.h"
#include "util_interrupt_log.h"
#include "cmd.h"

/************************************************************************************
 *  externs
 ************************************************************************************/
//extern void sample_dmac_initialize(void);

void sample_test_print_usage(void)
{
    printf("¥n");
    printf("+-<< Usage: Command and Switch Functions >>-----------------------------+¥n");
    printf(" SLIDESW #0 .... Select Operation Mode ( ON:Switch / OFF:Command )¥n");
    printf("< Switch Mode >¥n");
    printf(" PUSH SW #0 .... Exit Test loop!!!¥n");
    printf(" PUSH SW #1 .... Function-A¥n");
    printf(" PUSH SW #2 .... Function-B¥n");
    printf(" PUSH SW #3 .... Function-C¥n");
    printf(" SLIDESW #1:3 .. Option 0~7¥n");
    printf("< Command Mode >¥n");
    cmd_menu(NULL);
    printf("+-----------------------------------------------------------------------+¥n¥n");

    return;
}

int main(void)
{
    //ALT_STATUS_CODE result_code;
    uint32_t switch_raw = 0;
    uint32_t switch_raw_bk = 0;
    uint32_t switch_detect = 0;
    uint32_t switch_detect_on = 0;
    uint32_t switch_number = 0;
    bool disp_usage = true;
    bool switch_mode = true;

    printf("¥r¥nUser Application Start!¥r¥n");

    // CPU and board settings.
    util_intlog_init();
#if USED_CPU0_INIT==1
      cpu0_init();
#endif

    // Initializing the dmac functions of hwlib.
    //sample_dmac_initialize();
```

sample_test_print_usage()
このサンプルテストの使用方法を表示する関数です

cmd_menu()
COMMANDS_LIST commands[] に登録されているコマンドメニューを表示します

main()
このサンプルのメイン関数です

util_intlog_init() 割り込みログ・ユーティリティーの初期化関数です

cpu0_init() CPU0 の初期化関数です

次のページに続く

```
/* ############################################## */
/* ## Implement the test setting process here!!! ## */
/* ############################################## */
util_intlog_print();

printf("==== Start While(1) loop process!!! ==========¥n");
switch_raw_bk = sample_detect_switch();
switch_raw_bk ^= SAMPLE_SWITCH_BIT_SLIDE0;
while(1)
{
    if(disp_usage)
    {
        sample_test_print_usage();
        disp_usage = false;
    }
    // ---- Check the Slide-Switch and Push-Switch. -----
    switch_raw = sample_detect_switch();
    switch_detect = switch_raw ^ switch_raw_bk;
    switch_detect_on  |= switch_detect & switch_raw;

    // Push-Switch 0
    if((switch_detect_on & SAMPLE_SWITCH_BIT_PUSH0)
        &&(!(switch_raw & SAMPLE_SWITCH_BIT_PUSHALL)))
    {
        break;     // Exit Test loop!!!
    }

    // Change operation mode ?
    if(switch_detect & SAMPLE_SWITCH_BIT_SLIDE0){
        if(switch_raw & SAMPLE_SWITCH_BIT_SLIDE0){
            printf("Enter Switch Mode!¥n");
            switch_mode = true;
        } else {
            printf("Enter Command Mode! <Press Enter key to continue>¥n");
            while(getchar() != '¥n');     /* Clear stdin buffer */
            switch_mode = false;
        }
    }
    // ==== Branch by operation mode (Command Mode or Switch Mode) ====
    if(switch_mode){
    // ---- Switch Mode ---------------------------------------- //

        // Slide-Switch
        if(switch_detect & SAMPLE_SWITCH_BIT_SLIDEALL)
        {
            switch_number = switch_raw & SAMPLE_SWITCH_BIT_NUM;
            switch_number *= 1;     // To avoid warnings.

            if(switch_detect & SAMPLE_SWITCH_BIT_SLIDE1){
                printf("SAMPLE_SWITCH_BIT_SLIDE1¥n");
            }
            if(switch_detect & SAMPLE_SWITCH_BIT_SLIDE2){
                printf("SAMPLE_SWITCH_BIT_SLIDE2¥n");
            }
            if(switch_detect & SAMPLE_SWITCH_BIT_SLIDE3){
                printf("SAMPLE_SWITCH_BIT_SLIDE3¥n");
            }
        }
```

util_intlog_print() 割り込みログ出力ユーティリティー関数です

sample_detect_switch() スイッチ状態を取得する関数です

sample_test_print_usage() このサンプルの使用方法を表示する関数です

sample_detect_switch() スイッチ状態を取得する関数です

プッシュスイッチ 0 が ON されていたら、このテストの while ループを抜けます

DIP スイッチ bit 0 が ON されていたら、オペレーションを Switch モード にします

DIP スイッチ bit 0 が OFF されていたら、オペレーションを Command モード にして Enter キー入力待ちになります

Switch モード であれば、

DIP スイッチ bit 1 が ON されていたら、メッセージを表示します

DIP スイッチ bit 2 が ON されていたら、メッセージを表示します

DIP スイッチ bit 3 が ON されていたら、メッセージを表示します

次のページに続く

```
            // Push-Switch
            if(!(switch_raw & SAMPLE_SWITCH_BIT_PUSHALL)){

                if(switch_detect_on & SAMPLE_SWITCH_BIT_PUSH1){
                    switch_detect_on &= ~SAMPLE_SWITCH_BIT_PUSH1;
                    printf("SAMPLE_SWITCH_BIT_PUSH1¥n");
                    disp_usage = true;
                }
                if(switch_detect_on & SAMPLE_SWITCH_BIT_PUSH2){
                    switch_detect_on &= ~SAMPLE_SWITCH_BIT_PUSH2;
                    printf("SAMPLE_SWITCH_BIT_PUSH2¥n");
                    disp_usage = true;
                }
                if(switch_detect_on & SAMPLE_SWITCH_BIT_PUSH3){
                    switch_detect_on &= ~SAMPLE_SWITCH_BIT_PUSH3;
                    printf("SAMPLE_SWITCH_BIT_PUSH3¥n");
                    disp_usage = true;
                }
            }
        } else {
            // ---- Command Mode ------------------------------------------ //
            if(cmd_execute())
            {
                break;  // Exit Test loop!!!
            }
        }

        util_intlog_print();
        switch_raw_bk = switch_raw;
    }
    printf("==== End While(1) loop process. ================¥n");

    util_intlog_print();

    printf("Finished running the sample !!!¥r¥n");

    return 0;
}
```

プッシュスイッチ 1 が ON されていたら、メッセージを表示します

プッシュスイッチ 2 が ON されていたら、メッセージを表示します

プッシュスイッチ 3 が ON されていたら、メッセージを表示します

Command モード であれば、

cmd_execute() コマンド入力による処理を実行します。
"exit" コマンドが入力されたら、このテストの while ループを抜けます

util_intlog_print() 割り込みログ出力ユーティリティー関数です

util_intlog_print() 割り込みログ出力ユーティリティー関数です

終わり

【リスト 7-1】 このサンプルのメインルーチン・ソースコード

8. 便利なユーティリティー関数の紹介

util ディレクトリーには便利なユーティリティー関数が用意されています。

以下に代表的なユーティリティー関数を示します。

【表 8-1】 MMU 設定 ユーティリティー関数

ファイル	MMU 設定 ユーティリティー関数	説明
l2mmu_setting.c l2mmu_setting.h	void **sample_mmu_init_and_enable** (void)	以下の処理を行い、MMU を初期化し有効にします ・ MMU を初期化します ・ MMU テーブルを作成します ・ MMU を有効にします
	int **cpu0_l2mmu_init** (void)	以下の処理を行います ・ SIMD と VFP を有効にします ・ ACTLR.SMP = 1 / NSACR.NS_SMP = 1 ・ SCU を初期化します（SCU は MPCore の共有リソースです） ・ AXI トランザクション信号を設定します（AxUSER [0] = 1）。 この設定は、ACP によるコヒーレント転送に必要です ・ GIC ディストリビューター・レジスターを初期化します ・ MMU を設定し有効にします ・ キャッシュを有効にします ・ CPU 割り込みを初期化して有効にします ・ GIC グローバル割り込みを有効にします（Core0 のみ）

【表 8-2】 メモリーアクセス・ユーティリティー関数

ファイル	メモリーアクセス・ユーティリティー関数	説明
mem_util.c mem_util.h	void **sample_memset_address32** (uint32_t* start, size_t size)	メモリーに 32 ビットアドレス値を設定します
	void **sample_memset_incrementdata** (uint32_t* start, uint32_t testdata, size_t size)	メモリーに増分値 1 のインクリメント・データを設定します
	void **sample_memset_incrementdata_4byte** (uint32_t* start, uint32_t testdata, size_t size)	メモリーに増分値 4 のインクリメント・データを設定します
	void **sample_memdmp_word** (const uint32_t* start, size_t size)	32 ビットサイズでメモリーをダンプします
	void **sample_memdmp_halfword** (const uint16_t* start, size_t size)	16 ビットサイズでメモリーをダンプします
	void **sample_memdmp_byte** (const uint8_t* start, size_t size)	8 ビットサイズでメモリーをダンプします

【表 8-3】 usleep ユーティリティー関数

ファイル	usleep ユーティリティー関数	説明
usleep_soc.c	void **usleep** (uint32_t us)	グローバルタイマーを使用して、指定したマイクロ秒のスリープを挿入します

【表 8-4】 割り込みログ関連 ユーティリティー関数

ファイル	割り込みログ関連 ユーティリティー関数	説明
util_interrupt_log.c util_interrupt_log.h	void **util_intlog_init** (void)	割り込みログ初期化処理: 最初に必ず呼び出します
	void **util_intlog_record** (ALT_INT_INTERRUPT_t kind, int opt1, int opt2)	割り込みログ記録処理: 割り込みルーチン上で呼び出します
	void **util_intlog_print** (void)	割り込みログ出力処理: 通常ルーチン上で定期的に呼び出します

【表 8-5】 時間計測ユーティリティー関数

ファイル	時間計測 ユーティリティー関数	説明
util_time_measurement.c util_time_measurement.h	void **util_time_init** (void)	時間計測プログラムを初期化します ・ クロック設定情報をプリントします ・ 測定用グローバルタイマーを設定します ・ 測定記録情報を初期化します
	void **util_time_uninit** (void)	時間測定プログラムの初期化を解除します ・ 測定用グローバルタイマーの初期化を解除します ・ すべての測定結果をプリントし、測定記録情報をクリアします
	void **util_time_record_start_point** (uint32_t index)	時間測定の開始点を記録します
	void **util_time_record_end_point** (uint32_t index)	時間測定の終了点を記録します
	void **util_time_print_result_by_counter** (uint32_t index)	計測結果をカウンターでプリントします(見出しをプリントします)
	void **util_time_print_result_by_seconds** (uint32_t index)	計測結果を秒単位でプリントします(見出しをプリントします)
	void **util_time_print_result_all** (UtilTimePrintTarget_et printby)	すべての計測結果をプリントします
	void **util_time_print_result_partial** (int startid, int endid, UtilTimePrintTarget_et printby)	指定された内容で部分計測結果を出力します
	void **util_time_print_result_all_and_clear** (UtilTimePrintTarget_et printby)	すべての計測結果をプリントし、すべての記録をクリアします

9. HWLib（ハードウェア・ライブラリー）とは

ベアメタル・アプリケーションなどで使用される HWLib は、

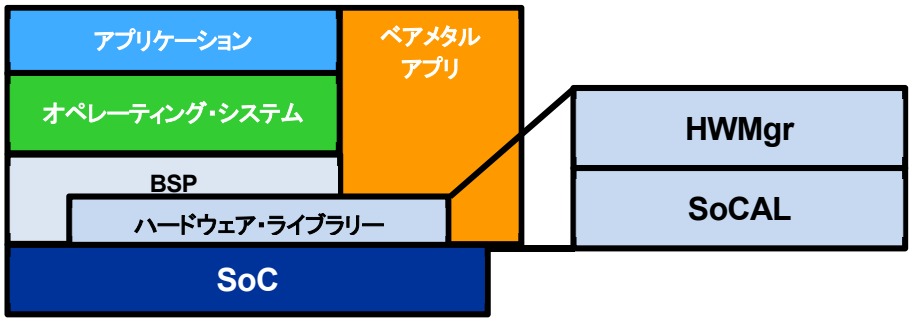
- SoC のローレベル・ソフトウェアを書く場合の複雑さを軽減します（自身で SoC のレジスター定義などを書く必要が無くなります）
- すべてのシステム・レジスターを抽象化しています
- ベアメタル・アプリケーション、OS ドライバーまたは OS カーネルなどが使用可能なレイヤーです
- システムの基本動作のためのテスト済み機能を含んでいます（例えば、クロック速度、キャッシュ設定、FPGA コンフィグレーションなどの変更）

9-1. HWLib のコンポーネント

HWLib は 2 つのコンポーネントで構成されています。

- SoC 抽象化レイヤー（SoCAL）-（ローレベル HAL）
  - ハードウェア IP レジスターにアクセスするためのマクロベースの抽象化レイヤー（ヘッダーファイル）です
  - ソフトウェアとハードウェアを分離します
- ハードウェア・マネージャー（HWMgr）
  - SoC ハードウェアへのハイレベルのアクセスを行う API の C およびアセンブリの集合体です。
  - `#include` で SoCAL ヘッダーファイルをインクルードします

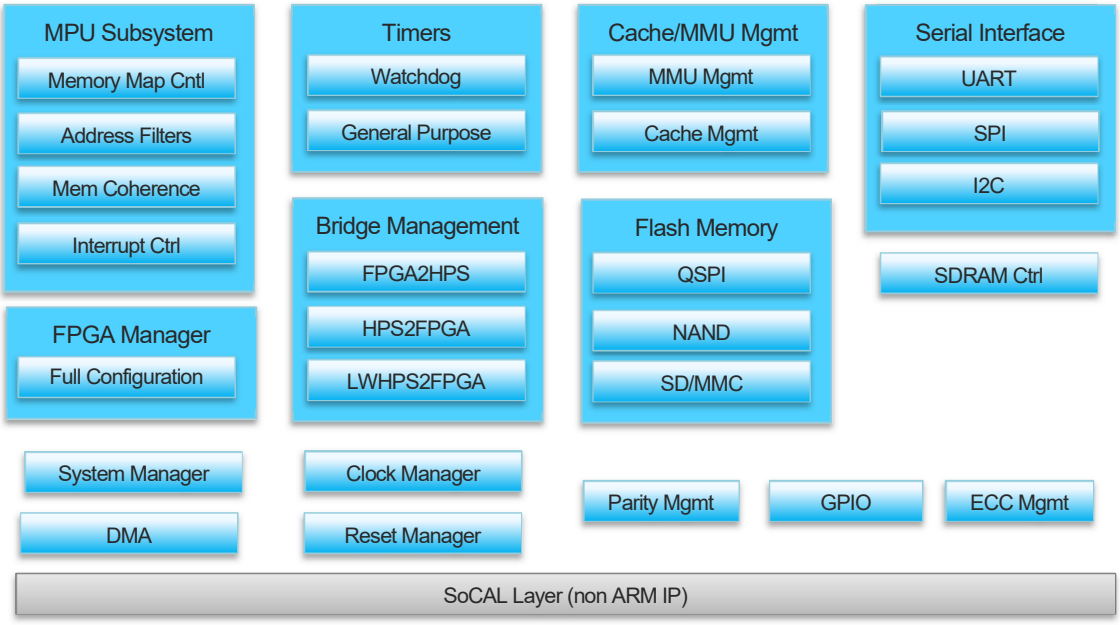
このサンプル・プロジェクトの util/hwlib ディレクトリーには、HWLib としてインテルから提供されるソースをすべて格納しており、使用したい HWLib のヘッダーファイルをインクルードすれば、すべての API を使用することが可能です。



【図 9-1】 HWLib のコンポーネント

9-2. HWLib の構成（API が用意されている機能）

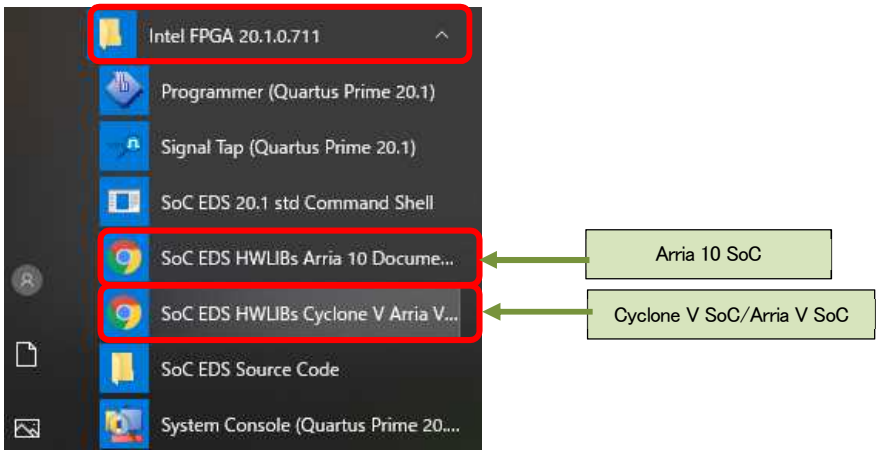
下図のような HWLib の API が用意されています。



【図 9-2】 HWLib API

9-3. HWLib に関するドキュメント

- SoCAL 関連ドキュメントの保存場所
  - *<SoC EDS installation directory>*/ip/altera/hps/altera_hps/doc/<device_name>/socal/html/index.html
    - **<device_name>** Cyclone V / Arria V 用: **soc_cv_av**
    - Arria 10 用: **soc_a10**
- HW マネージャ関連ドキュメントの保存場所
  - *<SoC EDS installation directory>*/ip/altera/hps/altera_hps/doc/hwmgr/**<device name>**/index.html
- Windows のスタート・メニューからもアクセス可能です



【図 9-3】 Windows のスタート・メニューから HWLib ドキュメントへのアクセス

10. HWLib Examples

examples ディレクトリーには以下のような各種 HWLib を使用したソフトウェア・ソースコードが用意されています。

- sample_cache_manage.c（キャッシュ管理サンプルプログラム）
- sample_clock_manager.c（クロック・マネージャー・サンプルプログラム）
- sample_dma_mem.c（DMA 転送サンプルプログラム）
- sample_dmac.c（HPS DMA（DMA-330）を使用したサンプルプログラム）
- sample_ecc.c（ECC 管理サンプルプログラム）
- sample_globaltmr.c（グローバルタイマー・サンプルプログラム）
- sample_gpio.c（GPIO サンプルプログラム）
- sample_gptmr.c（General-Purpose タイマー・サンプルプログラム）
- sample_interruptctrlSGI.c（割り込みコントローラー（主に SGI）サンプルプログラム）
- sample_time_measurement.c（時間測定を実装するサンプルプログラム）
- sample_watchdog.c（ウォッチドッグ・タイマー・サンプルプログラム）

これらの Example ソースコード・ファイルをプロジェクトの TOP ディレクトリーにコピーすることで、Makefile を修正すること無く Arm DS のコンパイル対象としてビルドすることが可能です。

また、これらの Example ソースコードを利用した評価方法としては、以下のような方法があります。

① 所望の Example ソースコードを参考に、sample_app.c ファイル内の main() 関数内に HWLib API を使用したユーザーコードを直接記述して実行・評価を行う。

② 所望の Example ソースコード内の sample_<機能名>_test_cmd() 関数を、sample_app.c ファイル内の main() 関数内から呼び出すように記述して実行・評価を行う。

③ 所望の Example ソースコード内の sample_<機能名>_test_cmd() 関数を、「11-1. Command モード時に実行するユーザーコマンドの追加方法」の手順にしたがって、util/cmd.c ファイル内に記述して、コマンド入力により実行・評価を行う。

以降のページから examples ディレクトリー内の各種サンプルについて概要を説明します（詳細については、各ソースコード・ファイルおよび readme.txt を参照ください）。

⚠ **注記：**

本 HWLib Example では、HPS 側の PUSH スイッチ（以下、PUSHSW）4 つと、SLIDE スイッチ（以下、DIPSW）4 つを操作することでソフトウェアの動作を切り替える仕組みを実装しております。

*但し、ターゲットボードに **Atlas-SoC / DE0-Nano-SoC / DE10-Nano** 開発ボードを選択した場合、HPS 側には前述のスイッチが不足する状態となるため、以下の対応でご利用頂く実装としています。*

- PUSHSW 0 … FPGA 側の PUSHSW（KEY0、KEY1）を同時押しすることで PUSHSW 0 を意味します
- PUSHSW 1 … FPGA 側の PUSHSW（KEY0）を単押しすることで PUSHSW 1 を意味します
- PUSHSW 2 … FPGA 側の PUSHSW（KEY1）を単押しすることで PUSHSW 2 を意味します
- PUSHSW 3 … HPS 側の USER PUSHSW（KEY2）を単押しすることで PUSHSW 3 を意味します
- DIPSW 0:3 … FPGA 側の DIPSW（SW0、SW1、SW2、SW3）

*また、**Arria 10 SoC** 開発ボードを選択した場合は、全てのスイッチ（PUSHSW×4、DIPSW×4）を HPS 側ではなく FPGA 側をご利用頂く実装としています。*

⚠ **注記：**

***sample_gpio.c**（GPIO サンプルプログラム）は **Arria® 10 SoC** 開発キット（a10socdk）には未対応です（GPIO 接続の HPS 用ユーザースイッチが用意されていないため）。*

10-1.sample_cache_manage.c（キャッシュ管理サンプルプログラム）

・ 【表 10-1】 sample_cache_manage.c ソースファイル

ソースファイル	sample_cache_manage.c
TOP 関数名	`int sample_cache_manage_test_cmd(char* options)`
概要	キャッシュ管理サンプルプログラム
機能	HWLib にて下記のカテゴリーに分類されている全ての API を試行します。 Cache Management API ＋ System Level Cache Management API ＋ L1 Cache Management API ＋ L2 Cache Management API
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_cache_manage_init(); → ターゲットボードの HPS 用 DIPSW, PUSHSW のための GPIO 設定を行います（テストプログラムの操作用） → L1 および L2 Cache を全て有効にする HWLib API を実行します。 → 動作確認用に割り込みコントローラー（GIC）の設定変更を行います。 L2 Cache Combined IRQ“ALT_INT_INTERRUPT_L2_COMBINED_IRQ”を有効に設定します。 上記の割り込みは L2 Cache Controller からの異常通知（3 種類）を全て束ねた（OR）条件で発行されます。 本サンプルでは、Cache 異常を引き起こす仕組みは実装していません。 何かしらの方法で Cache 異常を再現した場合に割込みが発生し、下記のコンソールメッセージが出力されます（未検証）。 ″[INTERRUPT]L2 Cache Combined Interrupt is occurred!! status=0x0000****″ → L2 Cache Controller に対して割り込み通知機能を有効にする設定を行う（HWLib を使用して Enable） ② sample_cache_manage_test_main(); → テストプログラムを実行します。無限ループ内で以下の処理を行います。
備考	詳細は sample_cache_manage_readme.txt および sample_cache_manage.c を参照ください。

DISPSW [4321]	PUSHSW0 の押下	PUSHSW1 の押下	PUSHSW2 の押下	PUSHSW3 の押下
xxx1	無限ループを終了（テストプログラムの終了）	–	–	キャッシュ効果の検証用に関数 mul_f32_test_funciton を起動して処理時間の計測および表示を実行 ⚠ **注記：** 本サンプルでは MMU の設定を行っていないため、キャッシュの効果は確認できません。必要に応じて MMU の設定を追加してご確認ください。
xxx0	L1, L2 Cache 機能を全て Enable/Disable にする API を実行（SW 押下毎に Enable と Disable を交互に実行）	–	–	
0010	–	alt_cache_system_invalidate を実行	alt_cache_l1_data_invalidate_all を実行	
0100	–	alt_cache_system_clean を実行	alt_cache_l1_data_clean_all を実行	
1000	–	alt_cache_system_purge を実行	alt_cache_l1_data_purge_all を実行	
0011	–	–	alt_cache_l2_data_invalidate_all を実行	
0101	–	–	alt_cache_l2_data_clean_all を実行	
1001	–	–	alt_cache_l2_data_purge_all を実行	

10-2.sample_clock_manager.c（クロック・マネージャー・サンプルプログラム）

・ 【表 10-2】 sample_clock_manager.c ソースファイル

<table>
<tr><td>ソースファイル</td><td>sample_clock_manager.c</td></tr>
<tr><td>TOP 関数名</td><td>int sample_clkmgr_test_cmd(char* options)</td></tr>
<tr><td>概要</td><td>クロック・マネージャー・サンプルプログラム</td></tr>
<tr><td rowspan="2">機能</td><td>HPS 用 DIPSW 1 – 4 の切り替え契機で Main PLL の M (1 – 4096) を変更するサンプルです（HPS の Main クロック周波数切り替えを試します）。HPS 用 DIPSW 1 – 4 の値に応じて Main クロックを下記の通りに変更します。
<table>
<tr><th>DISPSW
[4321]</th><th>Main PLL の M
(1 – 4096)</th><th>mpu_clk
(MHz)</th></tr>
<tr><td>0000</td><td>4</td><td>50</td></tr>
<tr><td>0001</td><td>8</td><td>100</td></tr>
<tr><td>0010</td><td>12</td><td>150</td></tr>
<tr><td>0011</td><td>16</td><td>200</td></tr>
<tr><td>0100</td><td>20</td><td>250</td></tr>
<tr><td>0101</td><td>24</td><td>300</td></tr>
<tr><td>0110</td><td>28</td><td>350</td></tr>
<tr><td>0111</td><td>32</td><td>400</td></tr>
<tr><td>1000</td><td>36</td><td>450</td></tr>
<tr><td>1001</td><td>40</td><td>500</td></tr>
<tr><td>1010</td><td>44</td><td>550</td></tr>
<tr><td>1011</td><td>48</td><td>600</td></tr>
<tr><td>1100</td><td>52</td><td>650</td></tr>
<tr><td>1101</td><td>56</td><td>700</td></tr>
<tr><td>1110</td><td>60</td><td>750</td></tr>
<tr><td>1111</td><td>64</td><td>800</td></tr>
</table>
</td></tr>
<tr><td>HWLib にて下記のカテゴリーに分類されている全ての API を試行します。
The Clock Manager API
＋ Clock Manager Status
＋ Safe Mode Options
＋ PLL Bypass Control
＋ Clock Gating Control
＋ Clock Source Selection
＋ Clock Frequency Control
＋ Clock Manager Interrupt Management
＋ Clock Group Configuration
上記カテゴリーの全ての API を試行するとともに、3 種の PLL（Main PLL、Peripheral PLL、SDRAM PLL）のコンフィグレーション情報を表示して目視可能としています。また、Clock Manager から発生する割り込み（3 種の PLL の Lock / Unlock）についても発生契機でコンソール表示を行うように対応しています。</td></tr>
<tr><td>サンプル関数</td><td>以下にサンプル関数の概要を示します。
① sample_clkmgr_init();
→ ターゲットボードの HPS 用 DIPSW, PUSHSW のための GPIO 設定を行います（テストプログラム操作用）。
→ Global Timer の設定を行います（CPU Core の動作クロックで Timer イネーブル）（テストプログラム操作用）。
→ 動作確認用に割り込みコントローラー（GIC）の設定変更を行います。
本関数で設定を行った後、PLL の Lock/Unlock 契機で割り込み処理が動作し、下記のコンソールメッセージを出力します。
″[IRQ#205] CLKMGR_IRQ (0x000000CD,0x000001C7) count=1″
② sample_clkmgr_test_print_pllconfig ();
→ Clock Manager の情報取得系 API を実行し、下記の情報を表示します（情報取得系 API の試行 兼 設定内容の確認）。
③ sample_clkmgr_test_customize_config ();
→ Clock Manager の設定変更系 API を一通り実行します（設定変更系 API の試行）。
④ sample_clkmgr_test_main();
→ テストプログラムを実行します。無限ループ内で以下の処理を行います。
(1) Global Timer カウンター値の下から 28bit 目以上（下位 32bit の bit#31:28）が変化する毎にカウンター値をコンソール出力します。
HPS 用 PUSHSW3 が押されている間は無条件にコンソール出力します（Timer カウンターの進み具合の確認用）。
(2) HPS 用 PUSHSW2 の押下契機で カテゴリー：Clock Manager Status の API を試行します。
PLL Lock/Unlock Status のクリアおよび表示を行います（Clock Manager Status API の試行）。
(3) HPS 用 PUSHSW1 の押下契機で カテゴリー：Safe Mode Options の API を試行します。
Safe Mode ステータスの表示およびクリアを行います（Safe Mode Options API の試行）。
(4) HPS 用 DIPSW1-4 の変更契機で、PLL の Multiplier 設定値の変更および変更後のクロック周波数の表示を試みます。
Main PLL の M(1-4096) を対象に、DIPSW1 を MSB、DIPSW4 を LSB として（4bit 値 × 4）を Multiplier の設定値に適用します。
(5) HPS 用 PUSHSW0 の押下契機で無限ループを終了します（テストプログラムの終了）。</td></tr>
<tr><td>備考</td><td>詳細は sample_clock_manager_readme.txt および sample_clock_manager.c を参照ください。</td></tr>
</table>

10-3.sample_dma_mem.c（DMA 転送サンプルプログラム）

・ 【表 10-3】 sample_dma_mem.c ソースファイル

<table>
<tr><td>ソースファイル</td><td>sample_dma_mem.c</td></tr>
<tr><td>TOP 関数名</td><td>int sample_dma_mem_test_cmd(char* options)</td></tr>
<tr><td>概要</td><td>DMA 転送サンプルプログラム</td></tr>
<tr><td>機能</td><td>
HPS 内臓 DMA (DMA-330) を使って、メモリー to メモリーの DMA 転送を行うサンプルです。
本サンプルでは MMU および L1, L2 キャッシュ、ならびに ACP ポートを有効化した環境下で転送を行う実装としています。
ターゲットボード上の PUSHSW、DIPSW の操作により下記のオプションを選択可能としています。
・ DMA Channel の選択(0～7)
・ 転送経路（通常ポート（ACP 未使用）／ACP ポート経由）
・ キャッシュ・メンテナンス操作を行う／行わない
⚠ 注記： Arria 10 SoC は、AXI トランザクションのキャッシュ属性を見て ACP ポートの利用を自動的に判断するため、ACP 使用/未使用のオプションは意味を持ちません。
上記のオプションを選択した上で DMA テストを実行すると、1 回の実行当たり MMU 設定の異なる以下 8 パターンのバッファー間転送を試行し、DMA 転送の実行にかかった処理時間および転送結果(OK / NG)を表示します。
<table>
<tr><th>テスト</th><th>転送元バッファー</th><th>転送先バッファー</th></tr>
<tr><td>TEST.01</td><td>NonCache Buffer</td><td>NonCache Buffer</td></tr>
<tr><td>TEST.01</td><td>Cacheable [Write-Through (WT)] Buffer</td><td>NonCache Buffer</td></tr>
<tr><td>TEST.01</td><td>Cacheable [Write-Back (WB)] Buffer</td><td>NonCache Buffer</td></tr>
<tr><td>TEST.01</td><td>Cacheable [Write-Back with Allocate (WBA)] Buffer</td><td>NonCache Buffer</td></tr>
<tr><td>TEST.01</td><td>NonCache Buffer</td><td>Cacheable [WBA] Buffer</td></tr>
<tr><td>TEST.01</td><td>Cacheable [WT] Buffer</td><td>Cacheable [WBA] Buffer</td></tr>
<tr><td>TEST.01</td><td>Cacheable [WB] Buffer</td><td>Cacheable [WBA] Buffer</td></tr>
<tr><td>TEST.01</td><td>Cacheable [WBA] Buffer</td><td>Cacheable [WBA] Buffer</td></tr>
</table>
DMA 転送を行う際の処理手順は以下のように実装しています。
実行時に表示される処理時間(※)も以下の手順別に表示されます。
＜DMA 転送テストの処理手順 ＞
1. テストデータ格納(転送元へデータ書き込み／転送先を 0 クリア)
2. キャッシュ・メンテナンス(キャッシュメモリーに格納されたデータを物理メモリーへ反映)
3. DMA-330 用マイクロコード生成
4. DMA 実行(転送開始 ～ DMA 完了割り込み発生)
5. DMA 転送結果をベリファイ .. 結果を OK/NG 表示
⚠ 注記： ACP ポート経由の DMA 転送を利用した場合、L1、 L2 キャッシュにも作用するため、CPU からのアクセスに対するキャッシュ効率が下がることが考えられます。
ACP の利用については、CPU 側の処理性能も十分に検証した上でご検討されるようご注意ください(本サンプルで表示される処理時間は、あくまで参考値としてください)。
プログラム実行開始後、初期設定や各種 API の実行テストが完了すると、下記の表示とともにループ処理を開始します。
"==== Start While(1) loop process!!! (Exit PUSHSW0(SW8) becomes ON.) ===="
ループ中に HPS 用 PUSHSW の操作を検出した場合、それぞれ以下の処理を実行します。
<table>
<tr><th>スイッチ</th><th>実行する処理</th></tr>
<tr><td>PUSHSW0</td><td>ループを抜けてプログラム終了</td></tr>
<tr><td>PUSHSW1</td><td>DMA レジスター表示（Management Thread, Ch Thead のステータスを表示）</td></tr>
<tr><td>PUSHSW2</td><td>DMA 転送テスト実行（オプション選択: 通常ポート（ACP 未使用））</td></tr>
<tr><td>PUSHSW3</td><td>DMA 転送テスト実行（オプション選択: ACP ポート経由）</td></tr>
<tr><td>DIPSW4</td><td>オプション選択: キャッシュ・メンテナンス ON/OFF</td></tr>
<tr><td>DIPSW3</td><td>オプション選択: DMA Channel 0 ～ 7</td></tr>
</table>
⚠ 注記： 基本的に、Running(F8) した状態で、PUSHSW / DIPSW を操作して DMA テストを実行します。転送データの内容を確認する場合には、Break(F9) してメモリービューを参照してください。
</td></tr>
<tr><td>補足</td><td>本サンプルの DMA 転送用 API には、HWLib の alt_dma.c をベースにカスタマイズした、alt_dma_custom.c を利用しています。
詳細については、sample_dma_mem_readme.txt の「補足 4」を参照ください。
config.mk の USED_DMA を 1 に設定しビルドを行ってください。</td></tr>
<tr><td>サンプル関数</td><td>以下にサンプル関数の概要を示します。
① sample_dmac_test_init();
→ DMA-330 の利用に必要な設定(割り込みコールバックの登録など)をこの関数内で行っています。
② sample_dmac_print_manager_status();
→ DMA Manager スレッドステータスを表示します。
③ sample_dmac_print_ch_status(channel, true);
→ DIPSW で選択中の DMA CH ステータスを表示します。
④ sample_dmac_test_main(channel, acp_en, cacheope_en);
→ DMA 転送テストを実行します。8 パターンの転送を行います。</td></tr>
<tr><td>備考</td><td>詳細は sample_dma_mem_readme.txt および sample_dma_mem.c を参照ください。</td></tr>
</table>

10-4.sample_dmac.c（HPS DMA（DMA-330）を使用したサンプルプログラム）

・ 【表 10-4】 sample_dmac.c ソースファイル

ソースファイル	sample_dmac.c
TOP 関数名	int sample_dmac_test_cmd(char* options)
概要	HPS DMA（DMA-330）を使用したサンプルプログラム
機能	HPS 内臓 DMA（DMA-330）を使って、DMA 転送を行うサンプルです。 このプログラムを実行する際は、コンソールから以下のパラメーターを入力します。 〈DMA 転送元アドレス〉〈DMA 転送先アドレス〉〈DMA 転送サイズ〉〈転送バイト幅〉 例えば、 転送元アドレス = 0x10000 転送先アドレス = 0x12000 転送サイズ = 0x100 転送バイト幅 = 8 の場合は次のように 10000 12000 100 8 と入力します。 Command: dmatest 10000 12000 100 8 dmatest 10000 12000 100 8 dma_test ==== DMA Test Parameters ==== - DMA CH Select ..... ALT_DMA_CHANNEL_0 - Source Address ..... 0x00010000 - Destination Address ..... 0x00012000 - Transport size ..... 0x00000100 - Transport via ACP Port ... False - Execute cache operation .. True ==== Time Mesurement Results (0 ~ 4) ==== [TIME# 0] - Elapsed Seconds (nsec): 0.000001 (1280) [TIME# 1] - Elapsed Seconds (nsec): 0.000002 (2720) [TIME# 2] - Elapsed Seconds (nsec): 0.000003 (3840) [TIME# 3] - Elapsed Seconds (nsec): 0.000003 (3040) [TIME# 4] - Elapsed Seconds (nsec): 0.000008 (8800) DMA Result ... OK [IRQ#136] DMA_IRQ0 (0x00000088,0x00000003) count=10
補足	本サンプルの DMA 転送用 API には、HWLib の alt_dma.c をベースにカスタマイズした、alt_dma_custom.c を利用しています。 詳細については、sample_dmac_readme.txt の「補足」を参照ください。 config.mk の USED_DMA を 1 に設定しビルドを行ってください。
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_dma_m2m_setting(bytes)； → 引数 bytes の値に従って DMA(DMA-330)の設定を行います。bytes には 1、2、4、8 のいずれかを設定します。 ② sample_dmac_test_main(ALT_DMA_CHANNEL_0, (void*)srcaddr, (void*)dstaddr, (size_t)size)； → DMA 転送実行サンプル関数 sample_dmac_test_execute() を呼び出します。 ③ sample_dmac_test_execute()； → alt_dma_channel_exec ()DMA 転送実行 API 関数を呼び出します。
備考	詳細は sample_dmac_readme.txt および sample_dmac.c を参照ください。

10-5.sample_ecc.c（ECC 管理サンプルプログラム）

・ 【表 10-5】 sample_ecc.c ソースファイル

ソースファイル	sample_ecc.c
TOP 関数名	int sample_ecc_test_cmd(char* options)
概要	ECC 管理サンプルプログラム
機能	HWLib にて下記のカテゴリーに分類されている API を一通り動作させます。 Error Correcting Code (ECC) Management On-Chip RAM の ECC を利用して以下の動作を確認します。 ・ ECC エラー・インジェクション ・ ECC 割り込みの発生 ・ ECC エラー時の読み出しデータ(メモリーチェック) ・ キャッシュ有効／無効による ECC 動作の違い プログラム実行開始後、初期設定や各種 API の実行テストが完了すると、下記の表示とともにループ処理を開始します。 "==== Start While(1) loop process!!! (Exit PUSHSW0(SW8) becomes ON.) ===="
	ループ中に HPS 用 PUSHSW の操作を検出した場合、それぞれ以下の処理を実行します。 スイッチ / 実行する処理 PUSHSW0 / ループを抜けてプログラム終了 PUSHSW1 / キャッシュクリーン を行った上でテスト領域へのリードアクセスを行います(ECC 割り込みの発生確認) PUSHSW2 / キャッシュパージ(クリーンと無効化)を行った上でテスト領域へのリードアクセスを行います(ECC 割り込みの発生確認) PUSHSW3 / DIPSW1、DIPSW2 の設定に応じて ECC エラーのインジェクションおよびメモリーチェックを行います HPS 用 DIPSW は以下の動作選択に利用します。 スイッチ / 動作選択 DIPSW1 / ECC Double Bit Error (uncorredtable) インジェクション設定 {OFF:無効／ON:有効} DIPSW2 / ECC Single Bit Error (correctable) インジェクション設定 {OFF:無効／ON:有効} DIPSW3 / L2C-310 Debug Mode 設定 {OFF:無効／ON:有効} (ON: 有効にするとキャッシュが「強制ライトスルー」かつ「ラインフィル無効」で動作するモードに入ります) DIPSW4 / L1 / L2 キャッシュ設定 {OFF:無効／ON:有効}
	● DIPSW1、 DIPSW2 のいずれかを ON にした状態で、PUSHSW3 を押すとメモリーチェックの書き込みアクセス時に ECC エラーがインジェクションされます。 メモリーチェック NG を検出する毎に、NG の番地／期待値／読み出し結果を表示します。 On-Chip RAM の先頭から 128byte(0x80)をテスト領域として利用します。 キャッシュ有効 (DIPSW4 = ON) の場合、書き込みアクセスの設定をライトバックにしているので On-Chip RAM への書き込みが動作せず、エラー・インジェクションが機能しません。 DIPSW3 を ON にすることで、強制的にライトスルーで動作するモードに入るためキャッシュ有効でもエラー・インジェクションが動作する状態に設定できます。 ● PUSHSW1、 PUSHSW2 のいずれかを押すと、メモリーチェックした領域に対してリードアクセスを行います。 メモリーチェック時にエラー・インジェクションした後に、本操作を行うことで ECC エラーの検出動作が確認できます。ECC 割り込みを検出する毎にメッセージを表示します。 キャッシュ有効(DIPSW4 = ON)の場合は、事前にキャッシュ・メンテナンス処理を行います。 (PUSHSW1 はキャッシュクリーン、 PUCHSW2 はキャッシュパージ(クリーンと無効化)を実行) ● メモリーチェックおよびリードアクセスは、動作検証のために様々なアクセス方法で実行するようになっています。 – On-Chip RAM アクセス用の番地: 0x00000000～ / 0xFFFF0000～ – ビット幅: 8bit / 16bit / 32bit / 64bit
サンプル関数	以下にサンプル関数の概要を示します。 ① util_time_init(); → 処理時間計測処理の初期化を行います。下記の処理を行います。 ・ 処理時間測定用にグローバルタイマーの設定を行います (測定を行う場合に必要な設定です。現状、測定処理の呼び出しは未実装です)。 ・ Clock Manager の設定および設定情報の表示を行います(各種 Clock 周波数など動作環境のパラメーター確認用です)。 ② sample_ecc_test_init(); → 以下の処理を行います。 ・ remap レジスターを設定します (先頭番地 0x00000000 から On-Chip RAM へのアクセスが可能となるように設定)。 ・ ターゲットボードの GPIO を設定します (HPS 用 PUSHSW、 DIPSW の設定)。 ・ On-Chip RAM の ECC 設定および有効化を行います (ECC 有効化および割り込み許可設定)。 ・ MMU 設定および有効化を行います (On-Chip RAM 領域は、L1 / L2 キャッシュ共にライトバックに設定)。 ③ sample_ecc_test_main(); → テストプログラムを実行します。 無限ループ内で上記「機能」欄に記載したスイッチ検出処理を実行します。 ④ sample_ecc_test_uninit(); → 本サンプルでは何も処理を行いません。 ⑤ util_time_uninit(); → 処理時間計測処理の事後処理を行います。 計測結果をコンソールに表示します。
備考	詳細は sample_ecc_readme.txt および sample_ecc.c を参照ください。

10-6.sample_globaltmr.c（グローバルタイマー・サンプルプログラム）

・ 【表 10-6】 sample_globaltmr.c ソースファイル

ソースファイル	sample_globaltmr.c
TOP 関数名	`int sample_globaltmr_test_cmd(char* options)`
概要	グローバルタイマー・サンプルプログラム
機能	HWLib にて下記のカテゴリーに分類されている API を一通り動作させます。 The Global Timer Manager API グローバルタイマーを動作させ以下の機能の動作を確認します。 ・ プリスケーラーの設定に応じてタイマーカウンターの周期が変わることを確認します。 ・ コンパレーター有効（Global Timer Control Register が Comp Enable = 1）な場合に以下の機能が動作することを確認します。 – オート・インクリメント機能（コンパレーターの比較結果が一致する毎に自動加算されます） – グローバルタイマーからの割り込み発生（コンパレーターの比較結果が一致する付近で発生します）
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_globaltmr_alldisable(); → グローバルタイマーの全機能を disable/stop に設定します（disable/stop 設定用 API の試行）。 ② sample_globaltmr_setting_gic(); → 動作確認用に割り込みコントローラー（GIC）の設定変更を行います。 ・ 本関数で設定を行った後、下記の条件で割り込みが動作します。 – グローバルタイマー割り込み契機で sample_globaltimer_callback() が起動します。 – HPS 用 DIPSW、PUSHSW の 8 ポートは割り込みトリガーとしません（設定変更せずデフォルトのまま）。 ③ sample_globaltmr_paraminit(); → グローバルタイマーの動作パラメーター設定用 API を試します（パラメーター設定用 API の試行 兼 初期設定）。 ・ 下記の 3 つのパラメーター設定が行われます。 – プリスケーラー設定値（タイマークロックの分周比）を 0 に設定します（入力クロック［CPU コアの動作クロック 800MHz］をそのままタイマークロックに適用。設定値を 1 にすると 400MHz、2 なら 200MHz という具合に、“設定値 + 1” の値で入力クロックが分周されます）。 – オート・インクリメント機能の加算レジスター値（32bit）に 0x40000000 を設定します。コンパレーターの比較結果が一致する毎にレジスター値がコンパレーターに自動加算されます。 – コンパレーター（64bit カウンター比較用レジスター）にグローバルタイマーの 現在値 + 0x0000000040000000 を設定します。 ④ sample_globaltmr_allenable(); → グローバルタイマーの全機能を enable/start に設定します（enable/start 設定用 API の試行）。 ⑤ sample_globaltmr_print_get_result(); → 各種レジスター値およびステータスの取得 API を試します（情報取得系 API の試行 兼 設定内容の確認）。 ⑥ sample_globaltmr_test_main(); → テストプログラムを実行します。 ・ 無限ループ内で以下の処理を行います。 – タイマーカウンター値の下から 26bit 目以上（下位 32bit の bit#31:26）が変化する毎にカウンター値をコンソールに出力します。HPS 用 PUSHSW3 が押されている間は無条件にコンソール出力します（タイマーカウンターの進み具合確認用）。 – HPS 用 PUSHSW1 の押下契機でコンパレーターのレジスター値（64bit）を変更します。上位 32bit は“カウンターの上位 32bit 値 + 1”、下位 32bit はタイマーカウンターの下位 32bit 値に設定し、変更後の値をコンソール出力します（コンパレーターの動作確認用にコンパレーターのレジスター値を少し進めます）。 – HPS 用 PUSHSW2 の押下契機でコンパレーターのレジスター値（64bit）をコンソールに出力します（オート・インクリメント機能の確認用）。 – HPS 用 DIPSW1 – 4 の変更契機でプリスケーラーの設定値を変更します。DIPSW1 を MSB、DIPSW4 を LSB として 4bit の値をそのままプリスケーラーに設定し、設定内容をコンソール出力します（プリスケーラーの設定値に連動して、タイマーカウンターの動作速度が変化する事が確認できます）。 – HPS 用 PUSHSW0 の押下契機で無限ループを終了します（テストプログラムを終了します）。 ⑦ sample_globaltmr_callback(); → 割り込みが発生したことをコンソールへ出力し、割り込みステータスをクリアします。
備考	詳細は sample_globaltmr_readme.txt および sample_globaltmr.c を参照ください。

10-7.sample_gpio.c（GPIO サンプルプログラム）

・ 【表 10-7】 sample_gpio.c ソースファイル

ソースファイル	sample_gpio.c
TOP 関数名	int sample_gpio_test_cmd(char* options)
概要	GPIO サンプルプログラム
機能	⚠ **注記：** 本サンプルプログラムは インテル® Arria® 10 SoC 開発キット（a10socdk）には未対応です（GPIO 接続の HPS 用ユーザースイッチが用意されていないため）。 HWLib にて下記のカテゴリーに分類されている API を一通り動作させます。 The General Purpose Input/Output Manager API ＋ General-Purpose IO Configuration Functions ＋ General-Purpose IO Interrupt Functions ＋ General-Purpose IO via Bit Index ＋ General-Purpose IO Utility Functions ターゲットボードの HPS 用 DIPSW、PUSHSW を入力信号として利用可能となるように GPIO のコンフィグレートを行い、各スイッチの操作に連動する形でデバッガのコンソール上に GPIO 入力レジスターの値の変化を表示します。
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_gpio_utility(); → カテゴリー：General-Purpose IO Utility Functions の API を試します（GPIO ユーティリティー API の試行）。 ② sample_gpio_utility(); → カテゴリー：General-Purpose IO Configuration Functions、および General-Purpose IO via Bit Index の API を試します（GPIO コンフィグレート用 API の試行）。 ③ sample_gpio_iconfig(); → カテゴリー：General-Purpose IO Interrupt Functions の API を試します（GPIO 割り込みコンフィグレート用 API の試行）。 ④ sample_gpio_iointerrupt(); → 動作確認用に割り込みコントローラー（GIC）の設定変更を行います。 ・ 本関数で設定を行った後、下記の条件で割り込みが動作します。 － GPIO 割り込み契機で sample_gpio_callback() が起動します。 － HPS 用 DIPSW1:2 を利用して割り込みトリガーを選択します。 DIPSW1:2 = 0 … Rising-Edge DIPSW1:2 = 1 … Falling-Edge DIPSW1:2 = 2 … Rising-Edge DIPSW1:2 = 3 … Falling-Edge － HPS 用 PUSHSW3 を契機に GPIO 割り込みを発生させます。 ⑤ sample_gpio_iopolling(); → 無限ループ内で GPIO の入力レジスターを監視し、値の変化を検出する毎にデバッガコンソールへ変化内容を出力します。 ※ HPS 用 PUSHSW0 の押下契機で無限ループを終了します（テストプログラムを終了します）。 ⑥ sample_gpio_callback(); → 割り込みの発生および割り込みステータスの値をログバッファーへ記録し、割り込みステータスをクリアします。ログバッファーへ記録した情報を元に、通常ルーチンに復帰した後でコンソールに出力します。
備考	詳細は sample_gpio_readme.txt および sample_gpio.c を参照ください。

10-8.sample_gptmr.c（General-Purpose タイマー・サンプルプログラム）

・ 【表 10-8】 sample_gptmr.c ソースファイル

ソースファイル	sample_gptmr.c
TOP 関数名	int sample_gptmr_test_cmd(char* options)
概要	General-Purpose タイマー・サンプルプログラム
機能	HWLib にて下記のカテゴリーに分類されている API を一通り動作させます。 The General Purpose Timer Manager API ＋ Enable, Disable, and Status ＋ Counters Interface ＋ Interrupts ＋ Mode Control HWLib から参照できる General Purpose Timer（以下、GPT）関連レジスターの初期値を全て表示した上で、全ての GPT（下記の 4 個）を全て起動します。 ・ OSC1 timer 0 … osc1_clk で動作する 32bit タイマー（動作クロック固定） ・ OSC1 timer 1 … osc1_clk で動作する 32bit タイマー（動作クロック固定） ・ SP timer 0 ….. l4_sp_clk で動作する 32bit タイマー ・ SP timer 1 ….. l4_sp_clk で動作する 32bit タイマー ⓘ **Note**: OSC1 timer は、外部クロック（osc1_clk）をそのまま動作クロックとします（固定）。 ⓘ **Note**: SP timer は、Main PLL（C1:main_base_clk）or Peripheral PLL（C4:periph_base_clk）をクロックソースとするため可変になります（※ 動作クロックを変更する際にはタイマー停止する旨の注意書きがマニュアルに記載されています）。 また、各 GPT のタイムアウトを契機とする割り込み（4 系統）を全て有効に設定し、発生契機でコンソール表示を行うように対応しています。
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_gptmr_test_init(); → テストプログラムの初期設定を行います。 ・ 下記の処理を行います。 – GPT 用 HWLib の初期化（alt_gpt_all_tmr_init）を実行。 – GPT の初期設定値を全て表示。 – テスト用に GPT の設定を変更。 ✓ OSC1 timer 0 … 20 秒毎にタイムアウト割り込み発生 (mode=ALT_GPT_RESTART_MODE_PERIODIC (User-defined count mode), resetcount=500000000 (@25MHz)) ✓ OSC1 timer 1 … 40 秒毎にタイムアウト割り込み発生 (mode=ALT_GPT_RESTART_MODE_PERIODIC (User-defined count mode), resetcount=1000000000 (@25MHz)) ✓ SP timer 0 ….. 5 秒毎にタイムアウト割り込み発生 (mode=ALT_GPT_RESTART_MODE_PERIODIC (User-defined count mode), resetcount=500000000 (@100MHz)) ✓ SP timer 1 ….. 10 秒毎にタイムアウト割り込み発生 (mode=ALT_GPT_RESTART_MODE_PERIODIC (User-defined count mode), resetcount=1000000000 (@100MHz)) ※ 詳細は sample_gptmr_test_init のコードをご確認ください。 – 変更後の GPT の設定値を全て表示。 – 割り込みコントローラーの設定および設定値を表示。 ② sample_gptmr_test_main(); → テストプログラムを実行します。 ・ 無限ループ内で以下の処理を行います。 – HPS 用 DIPSW1 – 4 の変更契機で、各 GPT の動作モードを変更します。また、モード変更を行った場合はタイマーを再起動します。 各 DIPSW の対応は以下の通りです。 DIPSW1 .. OSC1 timer 0 DIPSW2 .. OSC1 timer 1 DIPSW3 .. SP timer 0 DIPSW4 .. SP timer 1 ※ 各 DIPSW は、ON で PERIODIC（User-defined count mode）、OFF で ONESHOT（Free-running mode）になります。 – HPS 用 PUSHSW1 の押下契機で 4 個の GPT のカウンター値およびタイムアウトまでの残時間（ミリ秒）を表示します。 – HPS 用 PUSHSW2 の押下契機で 4 個の GPT の動作情報（モード、タイマーイネーブル、割り込みイネーブル、割り込みペンディング）を表示します。 – HPS 用 PUSHSW3 の押下契機で 4 個の GPT 全てを再起動（リセット）します。 – HPS 用 PUSHSW0 の押下契機で無限ループを終了します（テストプログラムを終了）。 ③ sample_gptmr_test_uninit(); → GPT 用 HWLib の事後処理（alt_gpt_all_tmr_uninit）を実行します。
備考	詳細は sample_gptmr_readme.txt および sample_gptmr.c を参照ください。

10-9.sample_interruptctrlSGI.c（割り込みコントローラー（主に SGI）サンプルプログラム）

・ 【表 10-9】 sample_interruptctrlSGI.c ソースファイル

ソースファイル	sample_interruptctrlSGI.c
TOP 関数名	int sample_intctrl_test_cmd(char* options)
概要	割り込みコントローラー（主に SGI）サンプルプログラム
機能	HWLib にて下記のカテゴリーに分類されている API を一通り動作させます。 Interrupt Controller Low-Level API [Secure] ＋ Interrupt Controller Global Interface [Secure] ＋ Interrupt Controller Distributor Interface [Secure] ＋ Software Generated Interrupts [Secure] ＋ Interrupt Controller CPU Interface [Secure] ＋ Interrupt Service Routine [Secure] ＋ Interrupt Utility Functions [Secure] 但し、他のサンプルにおいて割り込み設定に利用している API については他のサンプルにて検証済みのため割愛しています。
	本サンプルでは Software Generated Interrupt（以降、SGI）を動かすための設定を行い、HPS 用 PUSHSW の操作を契機に SGI を発行します。 尚、本サンプルはシングルコア構成のプロジェクトとして作成しているため、マルチコアでコア間の割り込み通知の確認は行えません（他のコア宛に SGI を発行する処理は本サンプルでも動作させています）。 プログラムの初期設定処理にて、全ての SGI 割り込み（下記の 16 個）に対して、コールバック関数の登録および割り込み設定（Distributer の有効化およびプライオリティー設定）を行います。 ALT_INT_INTERRUPT_SGI0 0x00 ALT_INT_INTERRUPT_SGI1 0x10 ALT_INT_INTERRUPT_SGI2 0x20 ALT_INT_INTERRUPT_SGI3 0x30 ALT_INT_INTERRUPT_SGI4 0x40 ALT_INT_INTERRUPT_SGI5 0x50 ALT_INT_INTERRUPT_SGI6 0x60 ALT_INT_INTERRUPT_SGI7 0x70 ALT_INT_INTERRUPT_SGI8 0x80 ALT_INT_INTERRUPT_SGI9 0x90 ALT_INT_INTERRUPT_SGI10 0xA0 ALT_INT_INTERRUPT_SGI11 0xB0 ALT_INT_INTERRUPT_SGI12 0xC0 ALT_INT_INTERRUPT_SGI13 0xD0 ALT_INT_INTERRUPT_SGI14 0xE0 ALT_INT_INTERRUPT_SGI15 0xF0 ⓘ **Note**: 上記リストの右側の数値は、プライオリティー設定値です。
	HPS 用 DIPSW（4bit）の値で、対象とする SGI（ALT_INT_INTERRUPT_SGI0 ～ ALT_INT_INTERRUPT_SGI15）を選択し、HPS 用 PUSHSW1 ～ 3 のいずれかを押下することで SGI 割り込みを発行します。 スイッチ / SGI 割り込みの発行 PUSHSW1 / SGI 発行元のコアへ割り込みを通知します（Core#0 宛）→ Core#0 にて SGI のコールバックが動作します PUSHSW2 / SGI 発行元のコアを除く全コアへ割り込みを通知します（Core#1 宛）→ Core#1 を動かさないので何も反応しません PUSHSW3 / SGI 発行元のコアを含む全コアへ割り込みを通知します（Core#0、 Core#1 宛）→ Core#0 のみコールバックが動作します。Core#1 は無反応です SGI のコールバック関数が動作すると、DS-5 のアプリケーション・コンソール上に下記形式にてメッセージが表示されます（括弧内“(xxx,yyy)”の数値は、 1 個目（xxx）が icciar の値、 2 個目（yyy）が当該 IRQ の検出回数カウンター、 count=* の値は全ての IRQ の総検出回数を示します）。 "[IRQ#0] SGI0 (0x00000000,0x0000000A) count=11"
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_intctrl_test_init(); → テストプログラムの初期設定を行います。下記の処理を行います。 ・ 割り込みコントローラーのイネーブル設定用 API の試行（alt_int_cpu_enable_ns/all、 alt_int_global_enable_ns/all）。 ・ 割り込みコントローラーの初期化 API 実行。 ・ 割り込みコントローラーの設定および設定値を表示（SGI の設定およびコールバックの登録など）。 ・ 割り込みコントローラーの CPU インターフェースのパラメーター取得 API の試行。 – alt_int_cpu_config_get() – alt_int_cpu_priority_mask_get() – alt_int_cpu_binary_point_get() – alt_int_cpu_binary_point_get_ns() ・ 割り込みコントローラーの CPU インターフェースのパラメーター設定 API の試行。 – alt_int_cpu_config_set() – alt_int_cpu_priority_mask_set() – alt_int_cpu_binary_point_set() – alt_int_cpu_binary_point_set_ns() ・ 割り込みコントローラーのイネーブル設定 API 実行（alt_int_cpu_enable、 alt_int_global_enable）。 ② sample_intctrl_test_main(); → テストプログラムを実行します。 ・ 無限ループ内で以下の処理を行います。

- HPS 用 PUSHSW1 の押下契機で SGI 発行 API を実行します。下記のパラメーターを指定することで、Core#0 宛で SGI を発行します。
  int_id（1st Argument）= HPS 用 DIPSW（4bit）に応じて、
  0（ALT_INT_INTERRUPT_SGI0）～ 15（ALT_INT_INTERRUPT_SGI15）を指定します。
  target_filter（2nd Argument）= ALT_INT_SGI_TARGET_SENDER_ONLY（発行元のみを指定）
- HPS 用 PUSHSW2 の押下契機で SGI 発行 API を実行します。下記のパラメーターを指定することで、Core#1 宛で SGI を発行します。
  int_id（1st Argument）= HPS 用 DIPSW(4bit)に応じて、
  0（ALT_INT_INTERRUPT_SGI0）～ 15（ALT_INT_INTERRUPT_SGI15）を指定します。
  target_filter（2nd Argument）= ALT_INT_SGI_TARGET_ALL_EXCL_SENDER（発行元を除く全てを指定）
- HPS 用 PUSHSW3 の押下契機で SGI 発行 API を実行します。下記のパラメーターを指定することで、Core#0、Core#1 宛で SGI を発行します。
  int_id（1st Argument）= HPS 用 DIPSW（4bit）に応じて、
  0（ALT_INT_INTERRUPT_SGI0）～ 15（ALT_INT_INTERRUPT_SGI15）を指定します。
  target_filter（2nd Argument）= ALT_INT_SGI_TARGET_LIST（target_list で宛先を指定）
  target_list（3rd Argument）= 3（bit#0[Core#0]=1 && bit#1[Core#1]=1）
- HPS 用 PUSHSW0 の押下契機で無限ループを終了します（テストプログラムを終了します）。

③ sample_intctrl_test_uninit();
- 割り込みコントローラーのディセーブル設定用 API の試行（alt_int_cpu_disable_ns/all, alt_int_global_disable_ns/all）。
- 割り込みコントローラーのディセーブル設定 API 実行（alt_int_cpu_disable, alt_int_global_disable）。

ⓘ **Note**: sample_interruptcontrollerSGI.c の下記の定義を書き換えることで、本サンプルとは異なる設定を試すこともできます。

```
/*! TEST Parameters */
#define SAMPLE_PARAM_USE_SECURE_BINARY_POINT   (false)   /*!< If true then use the Secure Binary Point Register for both secure and non-secure interrupts. If false then use Secure Binary Point Register for secure interrupts and non-secure Binary Point Register for non-secure interrupts. */
#define SAMPLE_PARAM_USE_FIQ_FOR_SECURE_INT    (false)  /*!< If true then signal secure interrupts using the FIQ signal. If false then signal secure interrupts using the IRQ signal. */
#define SAMPLE_PARAM_ALLOW_SECURE_ACK_FOR_ALL  (true)   /*!< Controls whether a secure acknowledgement of an interrupt, when the highest priority pending interrupt is non-secure, causes the CPU interface to acknowledge the interrupt. If true then a secure acknowledgement of the interrupt is not completed and the Interrupt ID of the Non-secure interrupt is returned. If false then a secure acknowledgement of the interrupt is not completed and the Interrupt ID of 1022 is returned. */
#define SAMPLE_PARAM_PRIORITY_MASK    (255)   /*!< The interrupt priority mask is the group priority needed to instruct the GIC to preempt lower priority interrupt. The valid range for this value is 0 - 255. */
#define SAMPLE_PARAM_BINARY_POINT     (0)     /*!< The binary point to use. The valid range for the value is 0 - 7. */
#define SAMPLE_PARAM_BINARY_POINT_NS  (0)     /*!< The binary point to use. The valid range for the value is 0 - 7. */
```

備考	詳細は sample_interruptctrlSGI_readme.txt および sample_interruptctrlSGI.c を参照ください。

10-10. sample_time_measurement.c（時間測定を実装するサンプルプログラム）

・ 【表 10-10】 sample_time_measurement.c ソースファイル

ソースファイル	sample_time_measurement.c
TOP 関数名	int sample_time_measurement_test_cmd(char* options)
概要	時間測定を実装するサンプルプログラム

機能	HWLib の The Global Timer Manager API を用いて、処理時間計測の仕組みを実装したサンプルプログラムです。
	下記のソース/ヘッダーファイルを別のプロジェクトに追加することで同様の仕組みで処理時間を計測することができます。
	– util_time_measurement.c
	– util_time_measurement.h
	● 処理時間計測用関数の説明
	void util_time_init(void);
	– 初期化処理。本機能を利用する際に必ず最初に呼び出してください。
	void util_time_record_start_point(uint32_t index);
	– 計測開始したいポイントに関数 CALL を実装してください。
	<uint32_t index>
	index は同時に複数個所の計測を行う際の計測ポイント識別用番号となります。 0～31 の適当な番号を指定してください。
	同時計測ポイントを増やしたい場合は、ヘッダーファイル内の #define UTIL_TIME_MAX_TRACE の定義値を増やすことで対応可能です。
	※ 引数の index 値は、全ての関数で同じ仕様として実装しています。
	void util_time_record_end_point(uint32_t index);
	– 計測終了するポイントにこの関数コールを実装してください（引数の index には、対応する開始ポイントと同じ値を指定してください）。
	void util_time_print_result_by_counter(uint32_t index);
	– index で指定した計測結果(1 件)をグローバルタイマーの「カウンター値」として表示します。
	void util_time_print_result_by_seconds(uint32_t index);
	– index で指定した計測結果(1 件)を「時間」として表示します。
	void util_time_print_result_all(void);
	– 全件の計測結果を「カウンター値」と「時間」の両方で表示します。
	void util_time_print_result_all_and_clear(void);
	– 全件の計測結果を「カウンター値」と「時間」の両方で表示します。また全ての計測結果情報をクリアします。
	⚠ **注記：**
	● 計測結果の 時間 は、HPS のメインクロック(mpu_clk)を 800MHz の設定とした場合に正しい時間で表示されることを前提にしています。
	異なるクロックの環境で計測を行う場合には、ヘッダーファイル内の下記定義を変更することで対応可能です。
	#define UTIL_TIME_GLOBALTIMER_PRESCALE 1
	#define UTIL_TIME_NSEC_PER_COUNT 10
	例えば、600MHz の環境で計測する場合には下記の値に変更します。
	#define UTIL_TIME_GLOBALTIMER_PRESCALE 2
	#define UTIL_TIME_NSEC_PER_COUNT 20
	UTIL_TIME_GLOBALTIMER_PRESCALE は、グローバルタイマーのプリスケーラー設定値となります。
	UTIL_TIME_NSEC_PER_COUNT は、時間を計算する際にグローバルタイマー・カウンターの差分値に乗算されます。
	● グローバルタイマーのカウント周期のナノ秒未満の値は全て切り捨て誤差となるため、プリスケーラーの値をナノ秒未満の誤差が少なくなる設定値として利用してください。
	グローバルタイマーを別用途で利用している場合などで、プリスケーラー値を変更できない場合には、カウンター値の計測結果をご利用ください(カウント周期の掛け算は別で行う)。
	● 上記パラメーター(UTIL_TIME_GLOBALTIMER_PRESCALE、 UTIL_TIME_NSEC_PER_COUNT)を自動計算するための Excel シートを用意しました。→ プロジェクト内のファイル： ParameterSettings_for_TimeMeasurement.xlsx をご利用ください。

サンプル関数	以下にサンプル関数の概要を示します。 ① sample_time_measurement_init(); → ターゲットボードの HPS 用 DIPSW、 PUSHSW のための GPIO 設定を行います(テストプログラム操作用の設定)。 → 時間計測処理の初期化 (void util_time_init) を呼び出します。 ② sample_time_measurement_test(); → 無限ループを行い、HPS 用 PUSHSW、 DIPSW の操作を契機に下記の処理を呼び出します。 – DIPSW1:4 .. 計測対象識別用の index 値として利用します。 – PUSHSW0 .. 計測結果の全件表示を行い無限ループから抜けます(util_time_print_result_all)(テストを終了します)。 – PUSHSW1 .. 計測結果の全件表示とクリアを呼び出します(util_time_print_result_all_and_clear)。 – PUSHSW2 .. DIPSW を index 値として計測開始を記録(util_time_record_start_point)。 – PUSHSW3 .. DIPSW を index 値として計測終了を記録し、1 件分の計測結果を表示します(util_time_record_end_point、util_time_print_result_by_counter、 util_time_print_result_by_seconds)。 Ⓟ **Point：** 無限ループ処理が走っている状態で、PUSHSW2 を押した後、任意の秒数待機した上で PUSHSW3 を押してみてください。 計測結果として待機した時間に応じた時間が表示されます。
備考	詳細は sample_time_measurement_readme.txt および sample_time_measurement.c を参照ください。

10-11. sample_watchdog.c（ウォッチドッグ・タイマー・サンプルプログラム）

・ 【表 10-11】 sample_watchdog.c ソースファイル

ソースファイル	sample_watchdog.c
TOP 関数名	int sample_wdog_test_cmd(char* options)
概要	ウォッチドッグ・タイマー(およびリセット・マネージャー)サンプルプログラム
機能	HWLib にて下記のカテゴリーに分類されている API を一通り動作させます。 The Watchdog Timer Manager API ＋ Watchdog Timer Enable, Disable, Restart, Status ＋ Watchdog Timer Counter Configuration ＋ Watchdog Timer Interrupt Management ＋ Watchdog Timer Miscellaneous Configuration The Reset Manager ＋ Reset Status ＋ Reset Control ● HWLib から参照できるウォッチドッグ・タイマー関連レジスターの初期値を全て表示した上で、下記の 3 種類のウォッチドッグ・タイマーを全て起動します。 – CPU Private Watchdog Timer(ALT_WDOG_CPU) – L4 Watchdog 0 (ALT_WDOG0) – L4 Watchdog 1 (ALT_WDOG1) ● また、ウォッチドッグ・タイマータイムアウトを契機とする割り込み(3 種類)を全て有効に設定し、発生契機でコンソール表示を行うように対応しています。 また、各ウォッチドッグ・タイマーの割り込みコールバック・ルーチン上で以下の処理を行います。 – CPU Private Watchdog Timer … ペンディング・クリア & コンソール表示のみ – L4 Watchdog 0 … ペンディング・クリア & コンソール表示 & COLD リセット実行。 – L4 Watchdog 1 … ペンディング・クリア & コンソール表示 & WARM リセット実行。
サンプル関数	以下にサンプル関数の概要を示します。 ① sample_wdog_test_init(); → 以下の処理を行います。 – リセット・マネージャーの要因レジスター値を表示します。 – ウォッチドッグ・タイマー用 HWLib の初期化 (alt_wdog_init) を実行します。 – ウォッチドッグ・タイマーの初期設定値を全て表示します。 – テスト用にウォッチドッグ・タイマーの設定を変更します。 ・ CPU Private Watchdog Timer (ALT_WDOG_CPU) → FREERUN モード (タイムアウト発生後もタイマー動作を継続する) ・ L4 Watchdog 0 (ALT_WDOG0) → INT THEN RESET モード (初回タイムアウトで割込み発生、2 回目で WARM リセットするモード) ・ L4 Watchdog 1 (ALT_WDOG1) → INT THEN RESET モード (初回タイムアウトで割込み発生、2 回目で WARM リセットするモード) (※ 詳細は sample_wdog_test_init のコードを直接ご確認ください) – 変更後のウォッチドッグ・タイマーの設定値を全て表示します。 – 割り込みコントローラーの設定および設定値を表示します。 ② sample_wdog_test_main(); → テストプログラム実行します。無限ループ内で以下の処理を行います – HPS 用 DIPSW1 – 4 の変更契機で、CPU Private Watchdog Timer 用のプリスケーラー設定値を変更します。 DIPSW1 を MSB、DIPSW4 を LSB として 4bit の値をそのまま設定値に適用します。 – HPS 用 PUSHSW1 の押下契機で CPU Private Watchdog Timer をリセットします。 – HPS 用 PUSHSW2 の押下契機で L4 Watchdog 0 をリセットします。 – HPS 用 PUSHSW3 の押下契機で L4 Watchdog 1 をリセットします。 – HPS 用 PUSHSW0 の押下契機で無限ループを終了します (テストプログラムの終了)。 ③ sample_wdog_test_uninit(); → ウォッチドッグ・タイマー非初期化 API 関数 (alt_wdog_uninit()) を呼び出します。
備考	詳細は sample_watchdog_readme.txt および sample_watchdog.c を参照ください。

11. 補足

11-1.Command モード時に実行するユーザーコマンドの追加方法

「6-2. Command モード」で説明しているように、COMMANDS_LIST commands[] にユーザーが作成した処理をコマンドとして登録し実行させることが可能です。

① 実行させたいユーザーのソースコード（xxx.c）を本プロジェクトの Top ディレクトリーに置きます。

② util/cmd.c ファイル内に以下の内容を追加してセーブします。

(1) ユーザーコマンド関数の extern 宣言を追記します。

```
extern int sample_dmac_test_cmd(char* options);
extern int sample_dma_mem_test_cmd(char* options);
extern int sample_cache_manage_test_cmd(char* options);
extern int sample_clkmgr_test_cmd(char* options);
extern int sample_ecc_test_cmd(char* options);
extern int sample_globaltmr_test_cmd(char* options);
extern int sample_gptmr_test_cmd(char* options);
extern int sample_intctrl_test_cmd(char* options);
extern int sample_time_measurement_test_cmd(char* options);
extern int sample_wdog_test_cmd(char* options);
```

ユーザーコマンド(関数)の extern 宣言を追加します。

※ この例では、examples ディレクトリー内のサンプルプログラムを追加しています。

【リスト 10-1】 ユーザーコマンド関数の extern 宣言を追記

(2) COMMANDS_LIST commands[] にユーザーコマンド関数に関する記述を追加します。

```
COMMANDS_LIST commands[] = {
    {"menu",     "Print of menu",                                                    cmd_menu},
    {"mr",       "mr <type:8/16/32> <addr(HEX)> ",                                   cmd_mem_read},
    {"mw",       "mw <type:8/16/32> <addr(HEX)> <data(HEX)>",                        cmd_mem_write},
    {"md",       "md <type:8/16/32> <addr(HEX)> <size(HEX)>",                        cmd_mem_dump},
    {"mf",       "mf <type(0:inc/1:fixed> <data(HEX)> <addr(HEX)> <size(HEX)>",      cmd_mem_fill},
    {"dma",      "dma <src(HEX)> <dst(HEX)> <size(HEX)> <bytes(1/2/4/8)>",           sample_dmac_test_cmd},
    {"dmamem",   "HPS internal DMA (DMA-330) example program",                       sample_dma_mem_test_cmd},
    {"cache",    "Cache Management example program",                                 sample_cache_manage_test_cmd},
    {"clk",      "Clock Manager example program",                                    sample_clkmgr_test_cmd},
    {"ecc",      "ECC Management example program",                                   sample_ecc_test_cmd},
    {"gltmr",    "Global Timer example program",                                     sample_globaltmr_test_cmd},
    {"gptmr",    "General-Purpose Timer example program",                            sample_gptmr_test_cmd},
    {"intctrl",  "Interrupt Controller (mainly SGI) example program",                sample_intctrl_test_cmd},
    {"time",     "Time measurement example program",                                 sample_time_measurement_test_cmd},
    {"wdog",     "Watchdog timer (and reset manager) example program",               sample_wdog_test_cmd},
    {"exit",     "exit"                                                               ,cmd_exit},
    {0,0,cmd_dummy}
};
```

ユーザーコマンド(関数)に関する記述を追加します

【リスト 10-2】 COMMANDS_LIST commands[] にユーザーコマンド関数を追加

ALT-HWLib-All-In-One_v20.1_r〇.〇 プロジェクトを再度ビルドして、ご使用のターゲットボードの DIP スイッチの bit 0 を OFF にして、 Command モードでプログラムを実行することにより、下図のように追加したユーザーコマンドがメニューに表示され、コマンド入力によりプログラムが実行できるようになります。

```
+-<< Usage: Command and Switch Functions >>-----------------------------------+
 SLIDESW #0 .... Select Operation Mode ( ON:Switch / OFF:Command )
< Switch Mode >
 PUSH SW #0 .... Exit Test loop!!!
 PUSH SW #1 .... Function-A
 PUSH SW #2 .... Function-B
 PUSH SW #3 .... Function-C
 SLIDESW #1:3 .. Option 0~7
< Command Mode >
 menu       :    Print of menu
 mr         :    mr <type:8/16/32> <addr(HEX)>
 mw         :    mw <type:8/16/32> <addr(HEX)> <data(HEX)>
 md         :    md <type:8/16/32> <addr(HEX)> <size(HEX)>
 mf         :    mf <type(0:inc/1:fixed> <data(HEX)> <addr(HEX)> <size(HEX)>
 dma        :    dma <src(HEX)> <dst(HEX)> <size(HEX)> <bytes(1/2/4/8)>
 dmamem     :    HPS internal DMA (DMA-330) example program
 cache      :    Cache Management example program
 clk        :    Clock Manager example program
 ecc        :    ECC Management example program
 gltmr      :    Global Timer example program
 gptmr      :    General-Purpose Timer example program
 intctrl    :    Interrupt Controller (mainly SGI) example program
 time       :    Time measurement example program
 wdog       :    Watchdog timer (and reset manager) example program
 exit       :    Exit
 * Note: HEX Value does not need 0x
+-----------------------------------------------+

Enter Command Mode! <Press Enter key to continue>


Command:
```

追加されたユーザーコマンド

【図 11-1】 追加したユーザーコマンドがメニューに表示される

11-2.本サンプルのディレクトリー／ファイル構成

11-2-1. ALT-HWLib-All-In-One_v20.1_r○.○ ディレクトリー（プロジェクトの TOP ディレクトリー）

【表 11-2】 ALT-HWLib-All-In-One_v20.1_r○.○ ディレクトリーのファイル構成

ALT-HWLib-All-In-One_v20.1_r○.○ ディレクトリー	説明
examples	examples ディレクトリー
linkerscripts	linkerscripts ディレクトリー
registers	registers ディレクトリー
target_board	target_board ディレクトリー
util	util ディレクトリー
config.mk	コンパイルに関する指示が含まれています
debug-hosted.ds	Cyclone V / Arria V 向けデバッグ・スクリプトファイル このファイルの記述によりデバッグ時の設定や自動化を行うことができます
debug-hosted_a10.ds	Arria10 向けデバッグ・スクリプトファイル このファイルの記述によりデバッグ時の設定や自動化を行うことができます
GNU-Debug-A10-All-In-One-Sample.launch	Arria10 向けサンプル・ランチャーファイル ARM DS デバッガの起動コンフィグレーション用ファイルです
GNU-Debug-A10-Attach.launch	Arria10 向けアタッチ・ランチャーファイル
GNU-Debug-CV-All-In-One-Sample.launch	Cyclone V / Arria V 向けサンプル・ランチャーファイル ARM DS デバッガの起動コンフィグレーション用ファイルです
GNU-Debug-CV-Attach.launch	Cyclone V / Arria V 向けアタッチ・ランチャーファイル
Makefile	本ベアメタル・サンプル・プロジェクトをビルドする Makefile です
sample_app.c	本ベアメタル・サンプル・アプリケーションのメイン C ソースコード・ファイル
sample_app_setting.c	本ベアメタル・サンプル・アプリケーションの設定 C ソースコード・ファイル
sample_app_setting.h	本ベアメタル・サンプル・アプリケーションの設定ヘッダーファイル
sample_dmac.c	本ベアメタル・サンプル・アプリケーションの DMA テスト C ソースコード・ファイル

11-2-2. examples ディレクトリー

【表 11-3】 examples ディレクトリーのファイル構成

examples ディレクトリー	説明
readme.txt	Examples の使用方法が書かれているテキストファイル
sample_cache_manage.c	HWLib を使用するキャッシュ管理のサンプルプログラム
sample_cache_manage_readme.txt	sample_cache_manage.c サンプルの readme テキストファイル
sample_clock_manager.c	HWLib を使用するクロック・マネージャーのサンプルプログラム
sample_clock_manager_readme.txt	sample_clock_manager.c サンプルの readme テキストファイル
sample_dma_mem.c	HWLib を使用する DMA 転送サンプルプログラム
sample_dma_mem_readme.txt	sample_dma_mem.c サンプルの readme テキストファイル
sample_dmac.c	HWLib を使用する HPS DMA（DMA-330）を使用したサンプルプログラム
sample_dmac_readme.txt	sample_dmac.c サンプルの readme テキストファイル
sample_ecc.c	HWLib を使用する Error Correcting Code（以降、ECC ）管理サンプルプログラム
sample_ecc_readme.txt	sample_ecc.c サンプルの readme テキストファイル
sample_globaltmr.c	HWLib を使用するグローバルタイマーのサンプルプログラム
sample_globaltmr_readme.txt	sample_globaltmr.c サンプルの readme テキストファイル
sample_gpio.c	HWLib を使用する General Purpose I/O（以降、GPIO ）サンプルプログラム
sample_gpio_readme.txt	sample_gpio.c サンプルの readme テキストファイル
sample_gptmr.c	HWLib を使用する General-Purpose タイマーのサンプルプログラム
sample_gptmr_readme.txt	sample_gptmr.c サンプルの readme テキストファイル
sample_interruptctrlSGI.c	HWLib を使用する割り込みコントローラー（主に SGI）のサンプルプログラム
sample_interruptctrlSGI_readme.txt	sample_interruptctrlSGI.c サンプルの readme テキストファイル
sample_time_measurement.c	HWLib を使用した時間測定を実装するサンプルプログラム
sample_time_measurement_readme.txt	sample_time_measurement.c サンプルの readme テキストファイル
sample_watchdog.c	HWLib を使用するウォッチドッグ・タイマー（およびリセット・マネージャー）のサンプルプログラム
sample_watchdog_readme.txt	sample_watchdog.c サンプルの readme テキストファイル

11-2-3. linkerscripts ディレクトリー

【表 11-4】 linkerscripts ディレクトリーのファイル構成

linkerscripts ディレクトリー	説明
arria10-dk-ram.ld	GNU C Compiler（以降、GCC ）用 Arria 10 SoC 向けリンカースクリプト・ファイル 本ベアメタル・サンプルにおけるプログラムのリンク（メモリー配置）に使用されます
cycloneV-dk-ram.ld	GCC 用 Cyclone V / Arria V SoC 向けリンカースクリプト・ファイル 本ベアメタル・サンプルにおけるプログラムのリンク（メモリー配置）に使用されます
soc_a10-scatter.scat	ARM C Compiler（以降、ARMCC ）用 Arria 10 SoC 向けリンカースクリプト・ファイル 本ベアメタル・サンプルにおけるプログラムのリンク（メモリー配置）に使用されます
soc_cv_av-scatter.scat	ARMCC 用 Cyclone V / Arria V SoC 向けリンカースクリプト・ファイル 本ベアメタル・サンプルにおけるプログラムのリンク（メモリー配置）に使用されます

11-2-4. registers / soc_a10 ディレクトリー

【表 11-5】 registers / soc_a10 ディレクトリーのファイル構成

registers / soc_a10 ディレクトリー	説明
soc_a10_hps_addon_dma330.tcf	Arria 10 SoC 向け DMA Controller（DMA-330）用のレジスター定義 DS-5 のレジスタービューに表示項目を追加するための設定ファイルです
soc_a10_hps_addon_mpul2_l2c310.tcf	Arria 10 SoC 向け L2 Cache Controller（L2C-310）用のレジスター定義 DS-5 のレジスタービューに表示項目を追加するための設定ファイルです
soc_a10_hps_addon_mpuscu.tcf	Arria 10 SoC 向け Cortex-A9 MPCore 内蔵ペリフェラル用のレジスター定義 DS-5 のレジスタービューに表示項目を追加するための設定ファイルです

11-2-5. registers / soc_cv_av ディレクトリー

【表 11-6】 registers / soc_cv_av ディレクトリーのファイル構成

registers / soc_cv_av ディレクトリー	説明
soc_cv_av_hps_addon_dma330.tcf	Cyclone V / Arria V SoC 向け DMA Controller（DMA-330）用のレジスター定義 DS-5 のレジスタービューに表示項目を追加するための設定ファイルです
soc_cv_av_hps_addon_mpul2_l2c310.tcf	Cyclone V / Arria V SoC 向け L2 Cache Controller（L2C-310）用のレジスター定義 DS-5 のレジスタービューに表示項目を追加するための設定ファイルです
soc_cv_av_hps_addon_mpuscu.tcf	Cyclone V / Arria V SoC 向け Cortex-A9 MPCore 内蔵ペリフェラル用のレジスター定義 DS-5 のレジスタービューに表示項目を追加するための設定ファイルです

11-2-6. target_board / a10socdk ディレクトリー

【表 11-7】 target_board / a10socdk ディレクトリーのファイル構成

target_board / a10socdk ディレクトリー	説明
ghrd_10as066n2.sof	Arria 10 SoC 開発ボード向け .sof ファイル
ghrd_10as066n2.sopcinfo	Arria 10 SoC 開発ボード向け .sopcinfo ファイル
u-boot-spl	Arria 10 SoC 開発ボード向け preloader ファイル
u-boot-spl.dtb	Arria 10 SoC 開発ボード向け preloader デバイスツリー
u-boot-with-spl.sfp	Arria 10 SoC 開発ボード向け Flash 書き込み用ブートローダー （Preloader x4 と Uboot x1 で構成）

11-2-7. target_board / atlas ディレクトリー

【表 11-8】 target_board / atlas ディレクトリーのファイル構成

target_board / atlas ディレクトリー	説明
soc_system.sof	Atlas SoC 開発ボード向け .sof ファイル
soc_system.sopcinfo	Atlas SoC 開発ボード向け .sopcinfo ファイル
u-boot-spl	Atlas SoC 開発ボード向け preloader ファイル
u-boot-spl.dtb	Atlas SoC 開発ボード向け preloader デバイスツリー
u-boot-with-spl.sfp	Atlas SoC 開発ボード向け Flash 書き込み用ブートローダー （Preloader x4 と Uboot x1 で構成）

11-2-8. target_board / c5socdk ディレクトリー

【表 11-9】 target_board / c5socdk ディレクトリーのファイル構成

target_board / c5socdk ディレクトリー	説明
soc_system.sof	Cyclone V SoC 開発ボード向け .sof ファイル
soc_system.sopcinfo	Cyclone V SoC 開発ボード向け .sopcinfo ファイル
u-boot-spl	Cyclone V SoC 開発ボード向け preloader ファイル
u-boot-spl.dtb	Cyclone V SoC 開発ボード向け preloader デバイスツリー
u-boot-with-spl.sfp	Cyclone V SoC 開発ボード向け Flash 書き込み用ブートローダー （Preloader x4 と Uboot x1 で構成）

11-2-9. target_board / de10nano ディレクトリー

【表 11-10】 target_board / de10nano ディレクトリーのファイル構成

target_board / de10nano ディレクトリー	説明
soc_system.sof	DE10-Nano 開発ボード向け .sof ファイル
soc_system.sopcinfo	DE10-Nano 開発ボード向け .sopcinfo ファイル
u-boot-spl	DE10-Nano 開発ボード向け preloader ファイル
u-boot-spl.dtb	DE10-Nano 開発ボード向け preloader ファイルのデバイスツリー
u-boot-with-spl.sfp	DE10-Nano 開発ボード向け Flash 書き込み用ブートローダー （Preloader x4 と Uboot x1 で構成）

11-2-10. target_board / sodia ディレクトリー

【表 11-11】 target_board / sodia ディレクトリーのファイル構成

target_board / sodia ディレクトリー	説明
soc_system.sof	Sodia 開発ボード向け .sof ファイル
soc_system.sopcinfo	Sodia 開発ボード向け .sopcinfo ファイル
u-boot-spl	Sodia 開発ボード向け preloader ファイル
u-boot-spl.dtb	Sodia 開発ボード向け preloader ファイルのデバイスツリー
u-boot-with-spl.sfp	Sodia 開発ボード向け Flash 書き込み用ブートローダー （Preloader x4 と Uboot x1 で構成）

11-2-11. util ディレクトリー

【表 11-12】 util ディレクトリーのファイル構成

util ディレクトリー	説明
hwlib	HWLib ディレクトリー
nios_hal	Nios® II Hardware Abstraction Layer（以降、Nios HAL ）ディレクトリー
cmd.c	コマンド・ユーティリティー C ソースコード・ファイル 実行させたい処理を COMMANDS_LIST commands[] に追加登録することで、コマンドによる実行が可能になります。
cmd.h	コマンド・ユーティリティー・ヘッダーファイル
l2mmu_setting.c	L2 MMU 設定ユーティリティー C ソースコード・ファイル
l2mmu_setting.h	L2 MMU 設定ユーティリティー・ヘッダーファイル
mem_util.c	メモリー・ユーティリティー C ソースコード・ファイル
mem_util.h	メモリー・ユーティリティー・ヘッダーファイル
usleep_soc.c	usleep ユーティリティー C ソースコード・ファイル
util_interrupt_log.c	割り込みログ・ユーティリティー C ソースコード・ファイル
util_interrupt_log.h	割り込みログ・ユーティリティー・ヘッダーファイル
util_time_measurement.c	時間計測ユーティリティー C ソースコード・ファイル
util_time_measurement.h	時間計測ユーティリティー・ヘッダーファイル

改版履歴

Revision	年月	概要
1	2019 年 3 月	初版
3	2021 年 5 月	v20.1 用に修正（ドキュメントテンプレートも更新） - ツール環境の変更（DS-5 から Arm DS）に対応 - MMU サンプル（sample_mmu.c）を削除 （全サンプル共通の処理で MMU のセットアップを実装済みのため） - Helio ボードの対応を削除