Intel ® SoC FPGA All-in-One Bare Metal Application Sample

Ver.22.1

March 2023 Rev. 4

Contents

Before You Read This Manual

The contents of this manual are current as of March 2023.

Some of the software, hardware, and operating procedures described in this manual are common even if they are not specified versions or devices, but some of them may not be common.

Symbols in documents

ⓘ **Note**	Provides supplementary information.
Ⓟ **Point**	Provides important points.
Reference	Provides reference materials and sites for better understanding.
⚠ **Note**	This document contains information that is not discussed in detail, but that is necessary.
⃠ **Prohibited**	Notes and what not to do are described.

Notations in sentences

Underline	Click to jump to another chapter in the manual or to an external site.
Bold italics	Indicates characters displayed in menus, windows, etc. when operating on the screen.
xxxxxxx↵	Indicates the command string to be entered.
Shaded	Indicates the tool to be used.

1. Introduction

This sample can be used as a starting point for building bare metal applications for Intel ® SoC FPGAs.

Because the hardware libraries (HWLib) and other code required for bare metal development are pre-placed and built into the project, users can simply include the required header files and use the APIs without editing the Makefile.

Also, unused APIs are excluded at the time of linking and do not affect the code size.

This document describes the following:

- Applicability Requirements (Supported versions, supported boards)
- Benefits of using this sample
- Sample Directory/File Structure
- Compilation Settings
- Sample Basic Behavior
- How to add commands
- Description of the main routine source code for this sample
- Introduction to useful utility functions
- What is HWLib (Hardware Library)?
- HWLib Examples

2. Advantages of Using this Sample

In a typical bare metal sample application, only the HWLib for the interface is used. To use other HWLibs, the Makefile must be modified to specify additional HWLib sources.

Since it is provided in the Makefile project, any source files added by the user must also be added to the Makefile, which takes time to understand if you are not familiar with the Intel SoC FPGA software development flow.

In this sample, all the sources provided as HWLib are registered and all the APIs can be used by including the HWLib header files you want to use.

Also, all the source files added to the TOP directory of the project are included for compilation, so you can start various evaluations basically without modifying the Makefile.

3. Usage environment

3-1. Development environment

The following table shows the main development environments used in this manual.

Table 1 Main environments used in this manual

Section No.	Item	Description
1	Host PC	64 bit machine with Microsoft® Windows® 10 (64 bit) This document has been tested using Windows® 10.
2	Intel ® Quartus® Prime Standard Edition Development Software (Since Quartus Prime)	A tool for developing SoC FPGA hardware. This document uses Quartus Prime Standard Edition Development Software version 22.1. ■ Quartus Prime Standard Edition version 22.1 ⚠ **Note:** Device data for the SoC FPGA installed on the target board must be installed. For information on installing Quartus Prime, refer to the following site: Quartus® Prime & ModelSim® Installation Instructions ⓘ **Note:** In this document, only the Quartus Prime Programmer is used to download hardware designs "4-6. FPGA Configuration" to the FPGA.
3	Arm® Development Studio Intel® SoC FPGA Edition (Arm DS)/Intel ® SoC FPGA Embedded Development Suite Standard Edition (SoC EDS)	Tools for developing SoC FPGA software. Arm DS replaces Arm® Development Studio 5 Intel® SoC FPGA Edition (hereinafter DS-5) and provides similar functionality. You can use Arm DS to compile and debug application software. Ⓟ **Point:** The final version of SoC EDS was version 20.1. Since then, the software has been unpackaged and the required files, such as the HWLib and gcc compilers, are available separately from the Web. You can use this version of the sample project without installing SoC EDS. ⚠ **Note:** If you do not want to install SoC EDS, set up the Linaro gcc compiler 7.2.0 using the instructions on the following page. Hardware Library (HWLibs) | Documentation | RocketBoards.org This document uses SoC EDS Standard Edition version 20.1. ■ SoC EDS Standard Edition version 20.1 Ⓟ **Point:** For more information on installing SoC EDS and Arm DS: How to install the SoC EDS Embedded Development Suite (SoC EDS) ⚠ **Note:** To debug bare-metal applications using the Intel ® FPGA Download Cable II (USB-Blaster™ II and later), Arm DS/DS-5 Intel® SoC FPGA Edition (Paid) is required. ⚠ **Note:** This version of the sample project assumes the use of Arm DS. If you want to use the DS-5, use an older version of the sample project.
4	Terminal Emulation Software	Serial terminal software is required to use this sample. This document uses freeware software called "Tera Term." ■ Tera Term Download URL ⚠ **Note:** In Tera Term, make the following settings for the valid COM port when connected to the target board's UART. ・Baud rate 115200 bps, 8 bit data, no parity, 1 stop bit, no flow control

3-2. Supported target boards

In this sample, the following target boards can be specified in TARGET_BOARD in the config.mk file.

Table 3 Supported Target Boards for This Sample

ⓘ Note:

This version of the sample project does not contain data for Helio boards.
If you want to use it with Helio, use the previous version of the sample project.

Section No	Target Board
1	Cyclone® V SoC Development Kit
2	Intel ® Arria® 10 SoC Development Kit
3	Helio-Cyclone ® V SoC Kit (no longer available)
4	Sodia-Cyclone ® V ST SoC Evaluation Board
5	DE0-Nano-SoC Kit/Atlas-SoC Kit (no longer available)
6	DE10-Nano Kit

4. How to use the sample

4-1. Connecting the target board

The following is an overview of the target board connections.

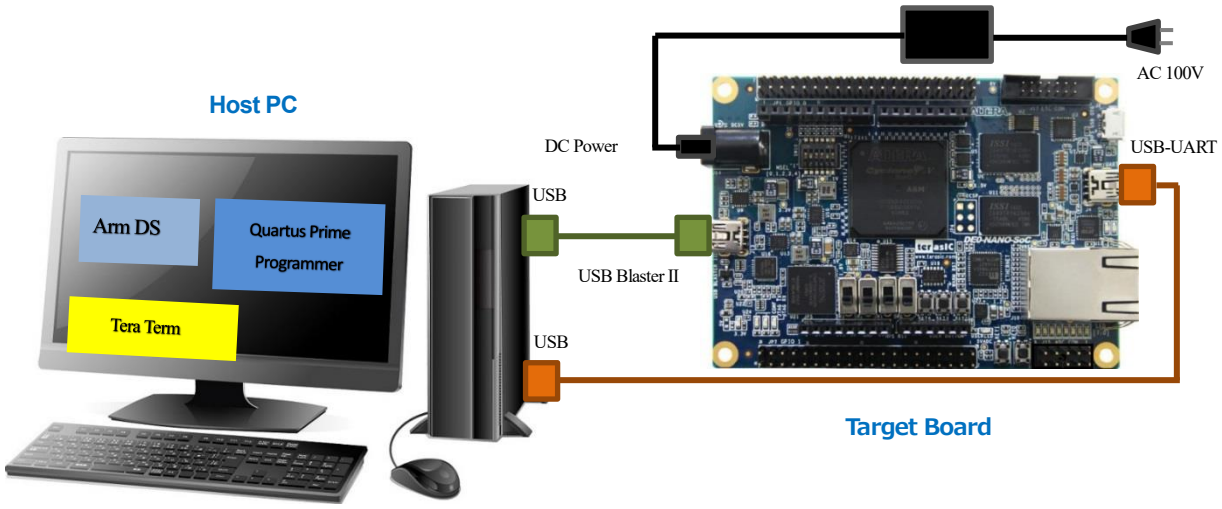


Figure 4 Connecting the target board

Connect the AC adapter and cables as follows:

- Connect the power (AC adapter) to the DC input connector on the target board.
- Use a USB cable to connect the host PC to the on-board USB-BlasterTM II connector on the target board.
- Use a USB cable to connect the host PC to the USB-UART connector on the target board.

> ⚠ **Note:**
>
> When using a Sodia board, a separate USB-Blaster™ II cable is required to configure the FPGA (Writing a .sof File) and debug/run bare metal applications.
> https://www.mouser.jp/ProductDetail/Intel-Altera/PL-USB2-

4-2. Start Arm DS and import sample programs

Start Arm DS and import sample **ALT-HWLib-All-In-One_v22.1_r○.○.tgz**.

To automatically configure the SoC EDS, start Arm DS from the following embedded command shell:

⚠ Note:

If you do not want to install SoC EDS, add the PATH of the compiler tool chain to the environment variable. If you do not want to change the PATH information in the Windows environment variable, you can write it in the .ini file for Arm DS (The .ini setting is not described in the Arm DS manual, so use it at your own risk.).

- Additional PATH information: "C:¥msys64 ¥home¥ ¥intel-socfpga-hwlib¥tools¥gcc¥bin"

The following is an example of adding the PATH information of the compiler tool chain to the .ini file for Arm DS (C: ¥Users¥ ¥AppData¥Roaming¥arm¥ds¥2022.2¥env.ini).

Here is an example of adding PATH information for the compiler toolchain (in bold blue):

```
SET=ARM_PRODUCT_DEF=C:¥Program Files¥Arm¥Development Studio 2022.2¥sw¥mappings¥intel_fpga.elmap
SET=PATH=%PATH%; C:¥msys64¥usr¥bin; C:¥msys64¥home ¥username ¥intel-sofpga-hwlib ¥tools¥gcc¥bin; C:¥Program
Files¥Arm¥Development Studio 2022.2¥sw¥ARMCompiler6.19¥bin;
```

Listing 4: Example of .ini

4-2-1. Launch Embedded Command Shell (SoC EDS)

Run the startup script stored in the Windows Start menu or the SoC EDS installation folder (embedded folder) to start the Embedded Command Shell.

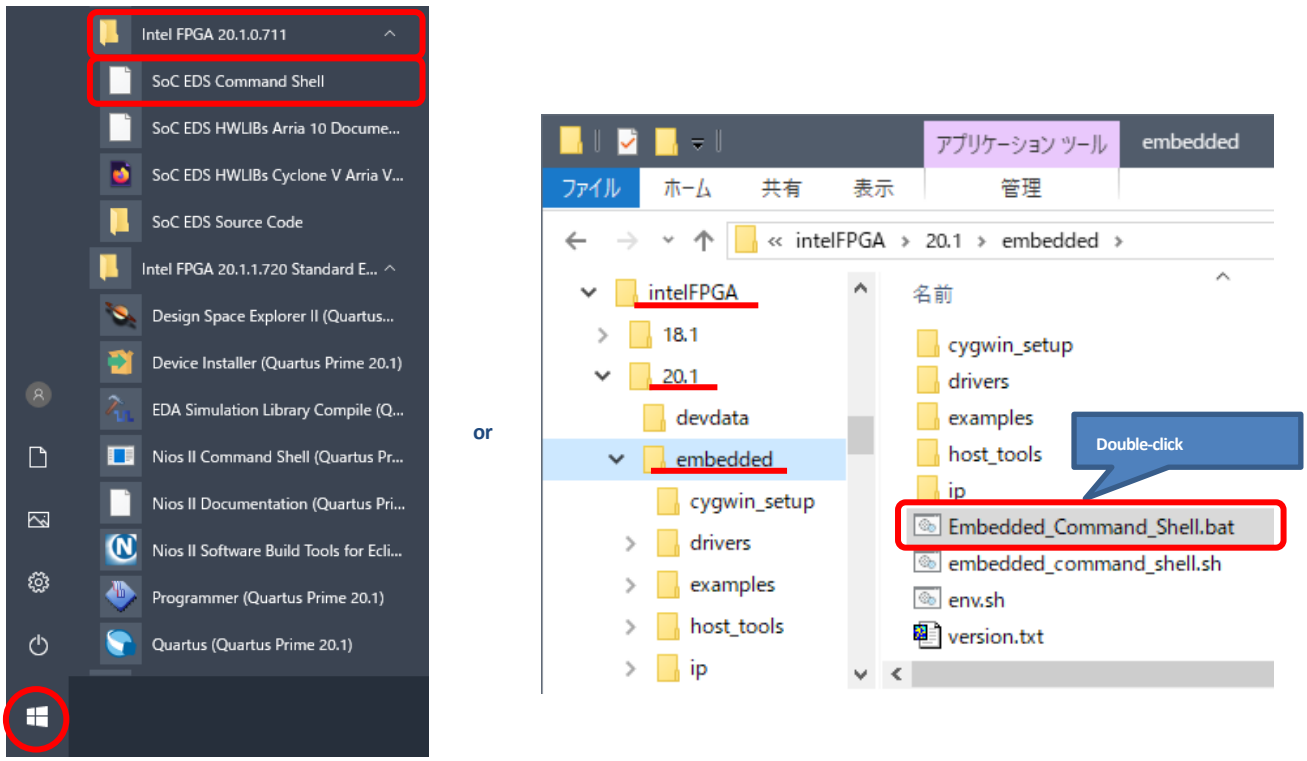


Figure 1 Launch Embedded Command Shell

4-2-2. Starting Arm DS

____ 1. If you are using SoC EDS, when the Embedded Command Shell window opens, enter the following commands (1) - (3) to start Arm DS.

① $ /cygdrive/c/Program ¥Files/Arm/Development ¥Studio ¥2022.2/bin/cmdsuite.exe

*The 2022.2 part varies depending on the version of Arm DS.

② $ bash

③ $ armds_ide &

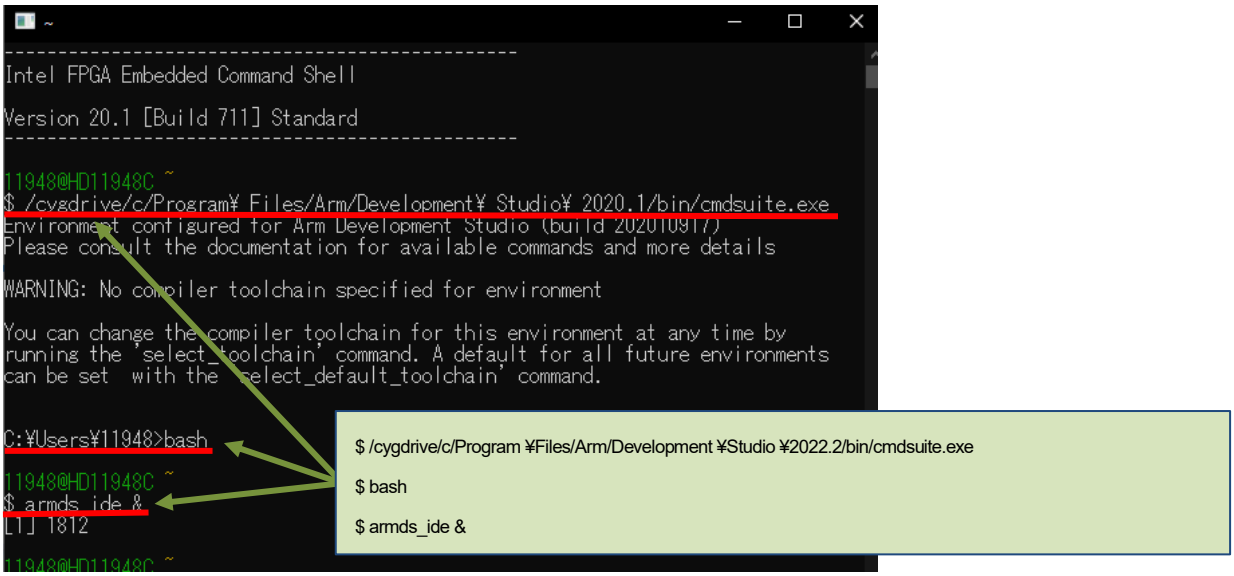


Figure 2: Starting Arm DS

____ 2. If you are not using SoC EDS, start Arm DS IDE 2022.2 from the Windows Start menu.

※ The 2022.2 part varies depending on the version of Arm DS.

⚠ **Note:** To start with this step, you must have previously set the compiler toolchain PATH information. (Review the note on the previous page again).

____ 3. You will be prompted for a workspace folder. Select or create a unique workspace for your software project.

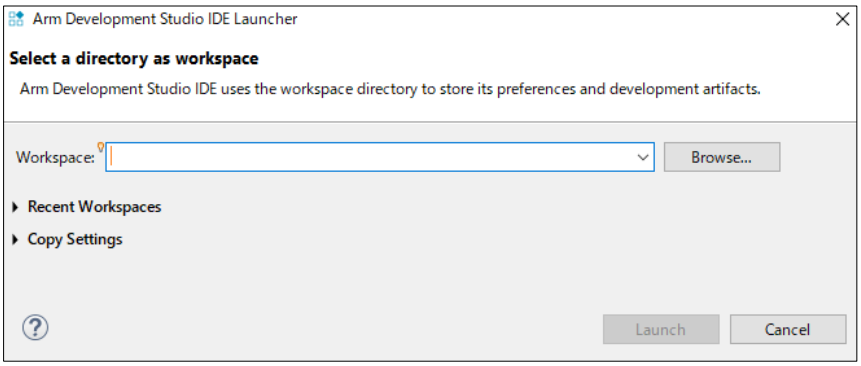


Figure 3 Specifying a Workspace for Arm DS

____ 4. If the Arm DS Welcome screen appears, click **Close** (**X**).

Note that it may take some time to close by pressing the X.

The Welcome screen can be used to access documentation, tutorials, and videos.

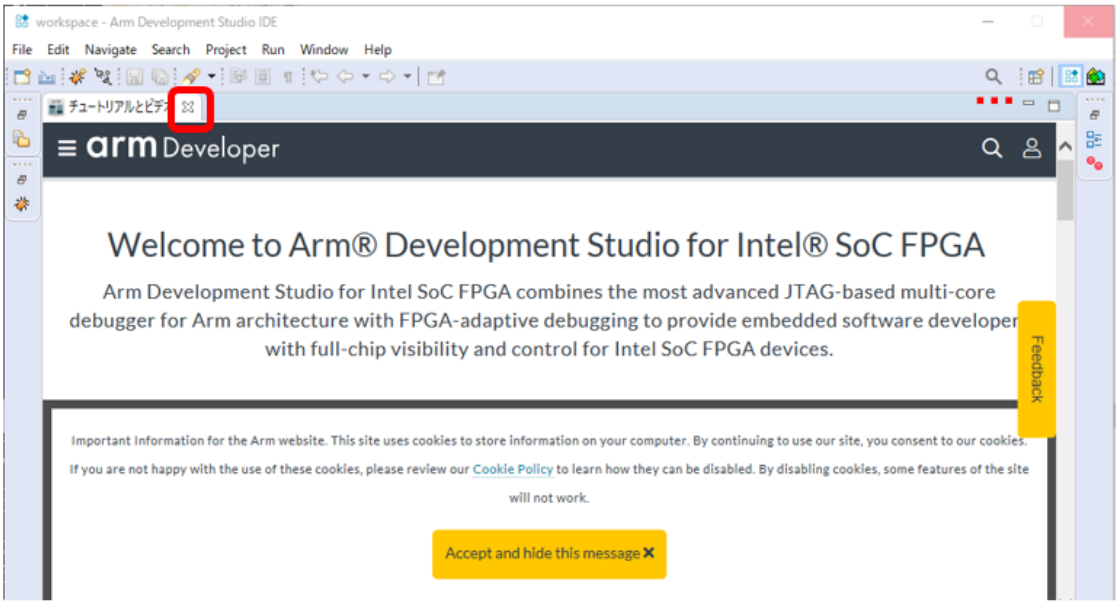


Figure 4: Arm DS Welcome screen

4-3. Import Bare Metal Sample Application

Import the Bare Metal Sample Application **ALT-HWLib-All-In-One_v22.1_ro.o** into Arm DS.

____ 1. From the DS menu, select **File** > **Import....**

____ 2. Select **General** > **Existing Projects into Workspace** and click **Next >**.

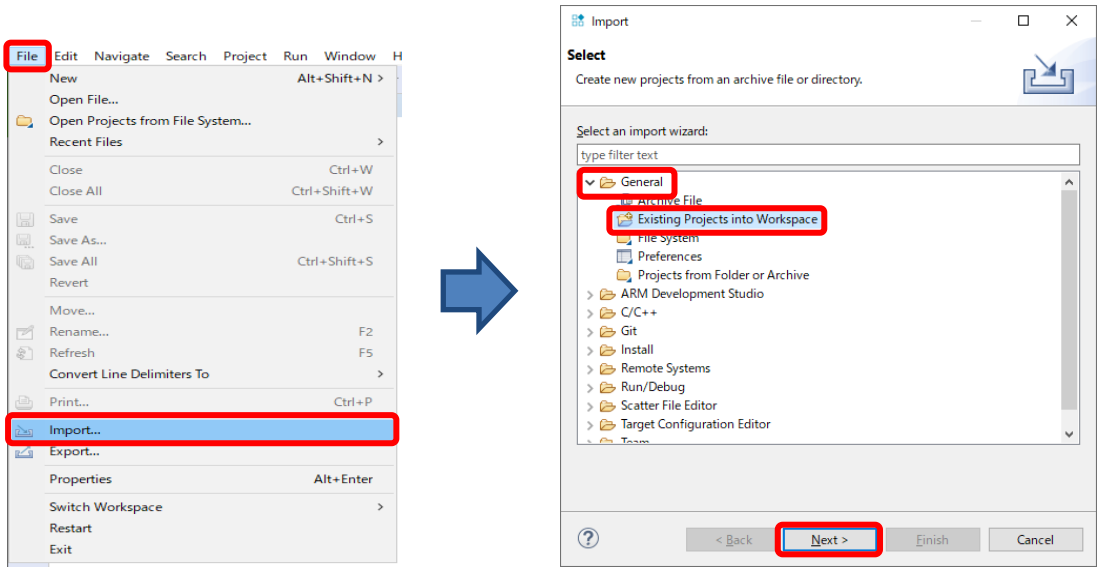


Figure 5 Importing Existing Projects

____ 3. Select the "**Select archive file:**" option. Use the "**Browse ...**" button to locate the following sample projects: **ALT-HWLib-All-In-One_v22.1_ro.o.tgz** Select and press the "**Finish**" button.

____ 4. The imported bare metal sample application project **ALT-HWLib-All-In-One_v22.1_ro.o** has been added to the **Project Explorer** panel on the left side of the Arm DS screen. Expand ALT-HWLib-All-In-One_v22.1_ro.o to see the various files contained in the project.

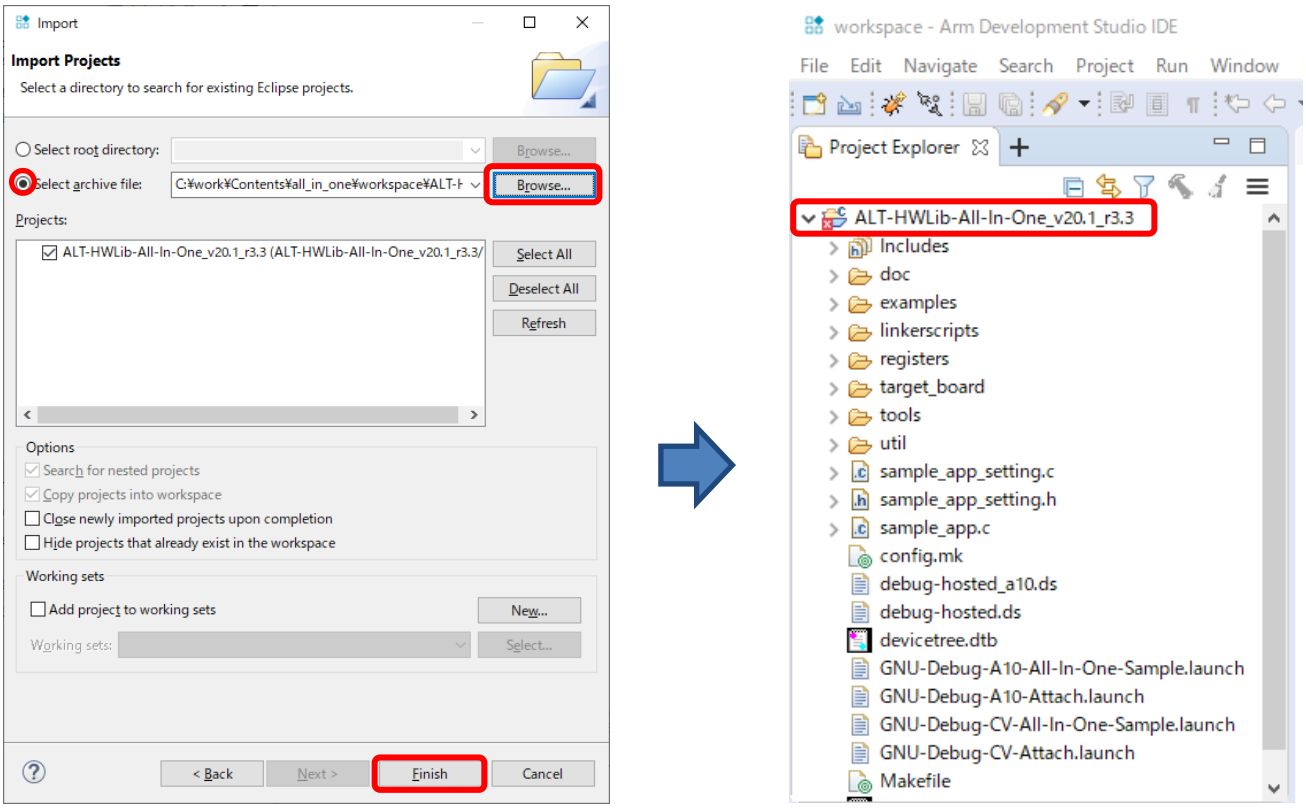


Figure 6 Adding the Sample Application

4-4. Compilation Settings

In this sample, you can specify the compiler to use, the target board, and whether to use semi-hosting in the **config.mk** file.

The default settings are ARM compiler, UART output (non-semi-hosting), generate system header files, and atlas board.

Also, when using DMA-related samples (sample_dma_mem.c/sample_dmac.c), build with USED_DMA set to 1. If not, build with USED_DMA set to 0.

```
#################
# 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

################################################
# Select example code to test (0:Disable/1:Enable) #
################################################
# ENABLE_EXAMPLE_DMA      : sample_dmac.c, sample_dma_mem.c (USED_DMA must be 1)
ENABLE_EXAMPLE_DMA        := 0
# ENABLE_EXAMPLE_CACHE    : sample_cache_manage.c
ENABLE_EXAMPLE_CACHE      := 0
```

Compiler specification: GCC/ARMCC

printf Output specification: 0= semi-hosting/1=UART

System header file generation: 0= do not generate/1= generate

⚠ **Note:** Use pre-generated header files in a Windows environment. (If you want to regenerate it, run the command in the Nios II Command Shell.)

Target board specification:
- atlas
- sodia
- c5socdk
- a10socdk
- de10nano

Use DMA related samples (sample_dma_mem.c/sample_dmac.c):
0 = Do not use
1 = Use

Enabling the Example programs:
0 = Disable
1 = Enable

[Listing 1] Compilation settings in the config.mk file4

4-5. Building the Bare Metal Sample Application

The next step is to build and run the imported Bare Metal Sample Application project.

4-5-1. Building the Project

Highlight the bare metal sample application project **ALT-HWLib-All-In-One_v22.1_r0.0** and right-click to run **Build Project**.

When the build completes, a bare metal application .**axf file** is generated.

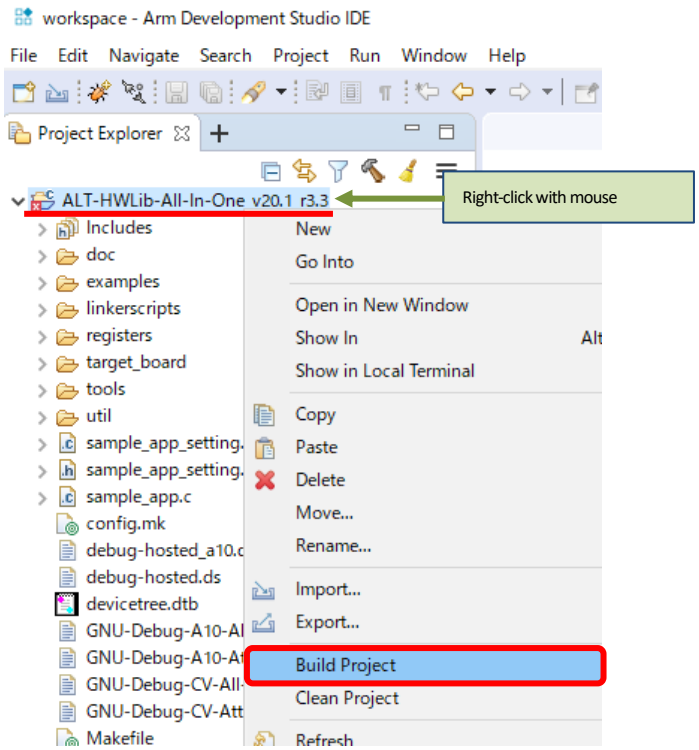
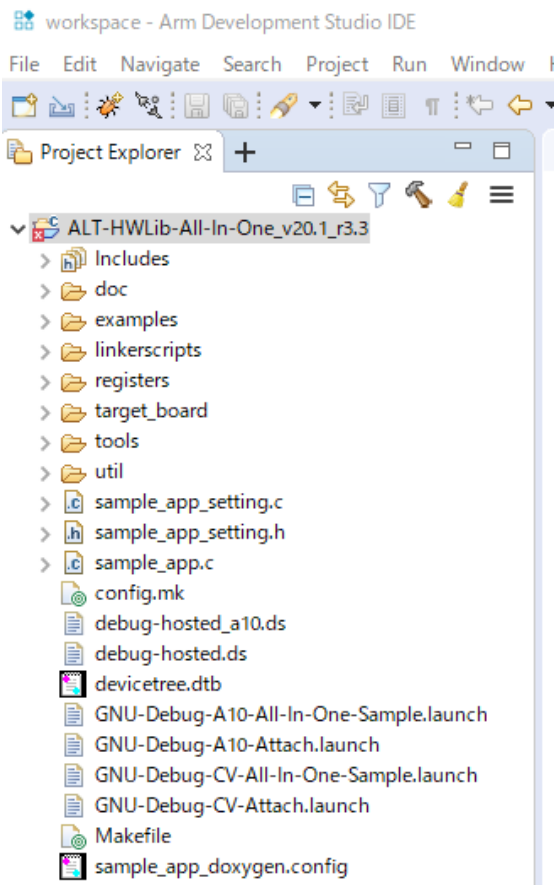


Figure 7 Building the Project

4-5-2. Bare Metal Sample Application Project File Configuration

The following figure shows the file configuration of the sample application.

Before Build



After Build

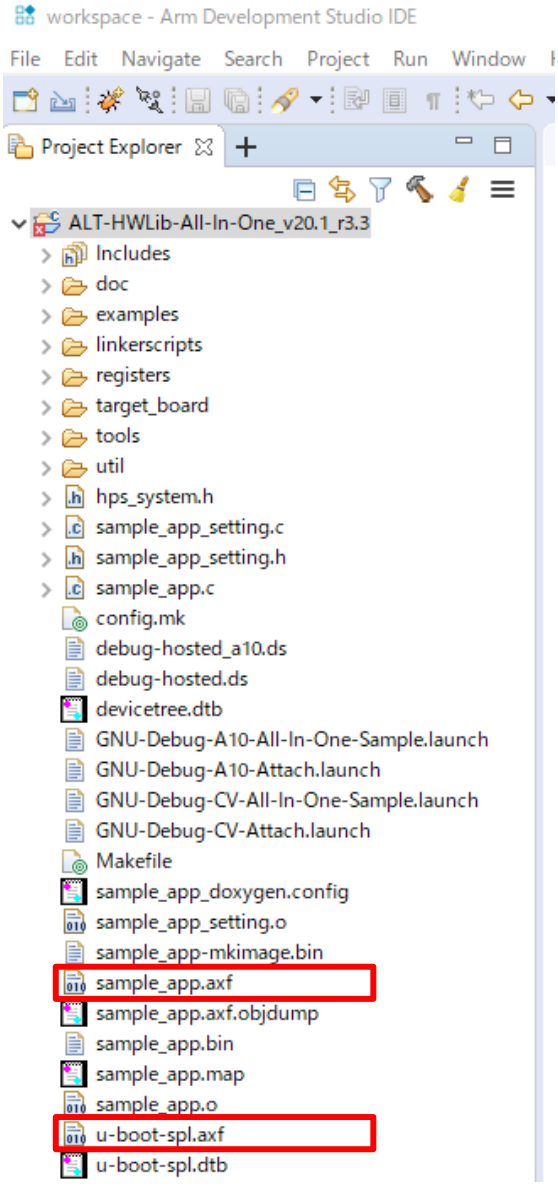


Figure 8 Sample Application File Configuration

- **sample_app.axf** is the sample application executable.
- **u-boot-spl.axf** is the Preloader executable.

4-6. FPGA Configuration

Next, configure the FPGA by programming a hardware design file with the extension **.sof** to the SoC FPGA.

4-6-1. How to Download the FPGA Design File to the Target Board

Download the hardware design (sof file) to the FPGA.

Refer to "4-1Connecting the target board" section to confirm that the board connection is complete. If there are no problems with the setup, connect the AC adapter to the board and turn on the power.

____ 1. From the Quartus Prime menu, click "**Tools**" ⇒ "**Programmer**" or click the **Programmer** icon to start Programmer.

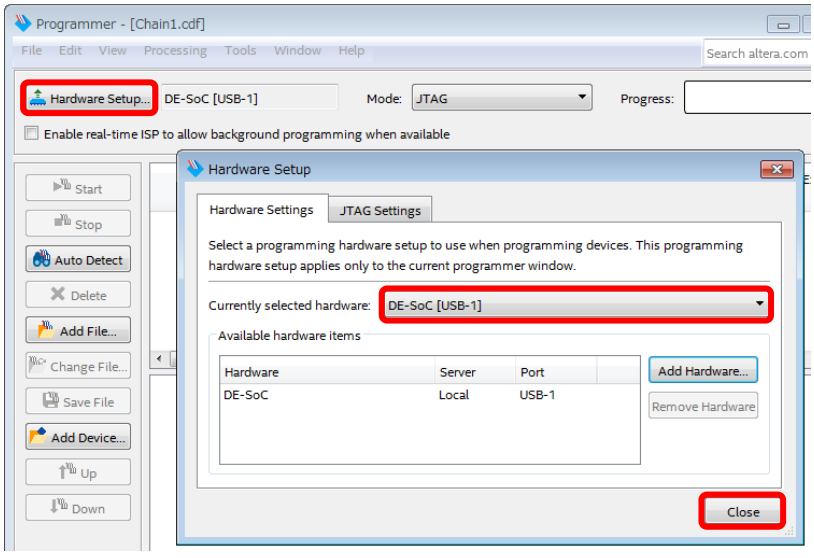
____ 2. Click the [**Hardware Setup**] button in **Programmer,** select the programming hardware from the **Currently selected hardware** pull-down list in the Hardware Setup window, and close the window.

ⓘ Note:

Since the ***Atlas-SoC*** board is used as an example in this manual, ***DE-SoC*** is selected as the programming hardware as shown below.
Select the programming hardware shown in the table below according to your target board.

[Table 4] Programming hardware corresponding to the target board

No.	Target Board	Programming Hardware
1	Cyclone® V SoC Development Kit	USB-BlasterII [USB-x]
2	Intel ® Arria® 10 SoC Development Kit	USB-BlasterII [USB-x]
3	Sodia-Cyclone ® V ST SoC Evaluation Board	USB-BlasterII [USB-x]
4	DE0-Nano-SoC Board/Atlas-SoC Board	DE-SoC [USB-x]
5	DE10-Nano Board	DE-SoC [USB-x]



[Figure 9] Hardware Setup

____ 3. Click the [Auto Detect] button to detect the FPGA connected to the JTAG chain on the board.

____ 4. Select the device installed on the target board from the **Select Device** window and click.

ⓘ Note:

In this manual, the ***Atlas-SoC*** board is used as an example, so ***5CSEMA4*** is selected as the device as shown below.

Select the device shown in the table below according to your target board.

[Table 4] Devices corresponding to the target board

Section No.	Target board	Device
1	Cyclone® V SoC Development Kit	5CSXFC6
2	Intel ® Arria® 10 SoC Development Kit	10AS066N3
3	Sodia-Cyclone ® V ST SoC Evaluation Board	5CSTFD6
4	DE0-Nano-SoC Board/Atlas-SoC Board	5CSEMA4
5	DE10-Nano Board	5CSEBA6

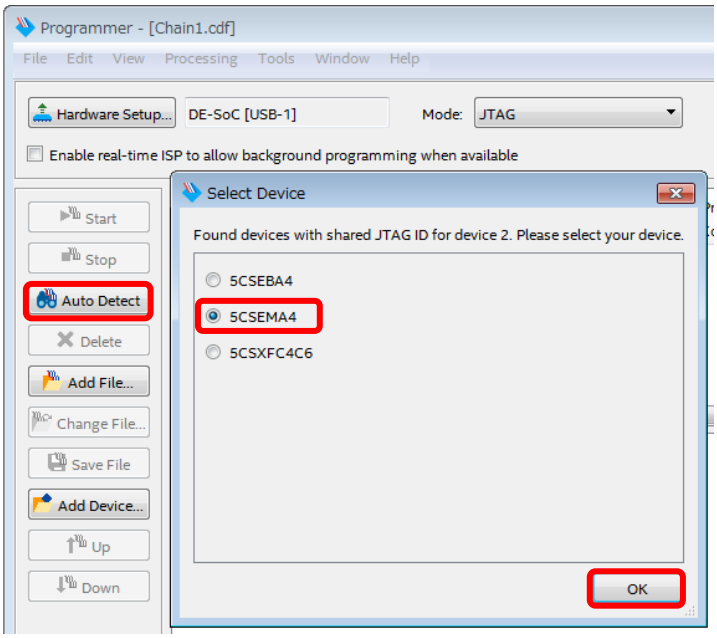


Figure 410 Device Selection (Atlas-SoC Board Example)

____ 5. If the following dialog box appears, select **Yes**.

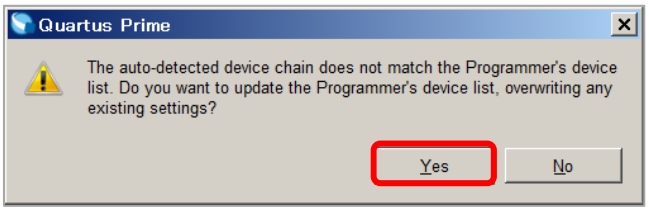


Figure 411 Dialog Box

This displays SOCVHPS and 5CSMA4 on the JTAG chain. SOCVHPS indicates that the HPS side has been recognized, and 5CSMA4 indicates that the FPGA side has been recognized.

____ 6. Select the file you want to download.

5 Right-click on CSEMA4 in the **Device** field and click **Change File**.

In the **Select New Programming File** dialog box, select the **.sof** file corresponding to the target board.

ⓘ Note:

The .sof file corresponding to each target board is located under the ***target_board*** directory of this sample project.

Select the .sof file according to the target board you are using.

Table 4 .sof file corresponding to the target board

Section No.	Target board	.sof file
1	Cyclone® V SoC Development Kit	c5socdk¥**soc_system.sof**
2	Intel ® Arria® 10 SoC Development Kit	a10socdk¥**ghrd_10as066n2.sof**
3	Sodia-Cyclone ® V ST SoC Evaluation Board	sodia¥**soc_system.sof**
4	DE0-Nano-SoC board/Atlas-SoC board	atlas¥**soc_system.sof**
5	DE10-Nano board	de10nano¥**soc_system.sof**

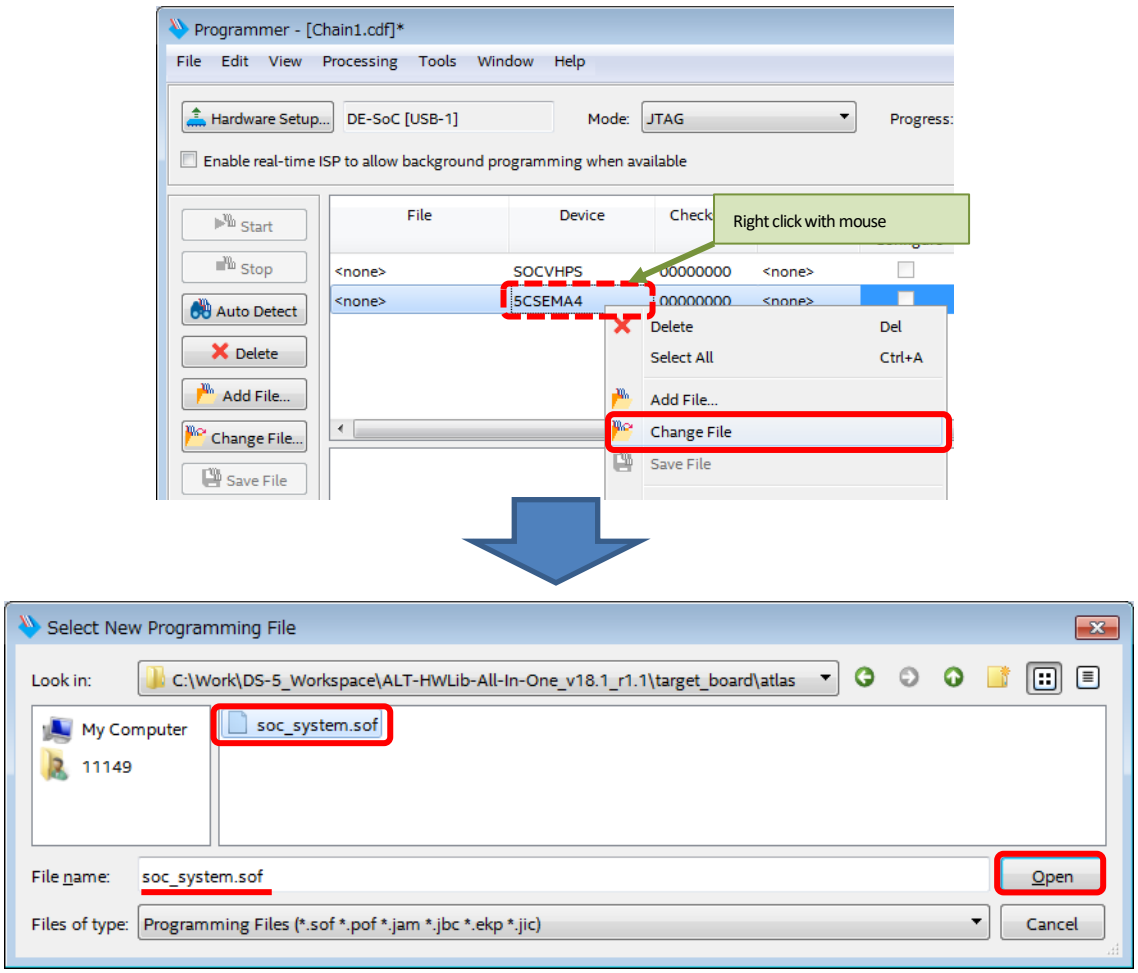


Figure 412 Selecting the sof file (Atlas-SoC board example)

____ 7. Check "**Program/Configure**" and click the [**Start**] button to complete the configuration. This operation causes the operational image to be written to the FPGA.

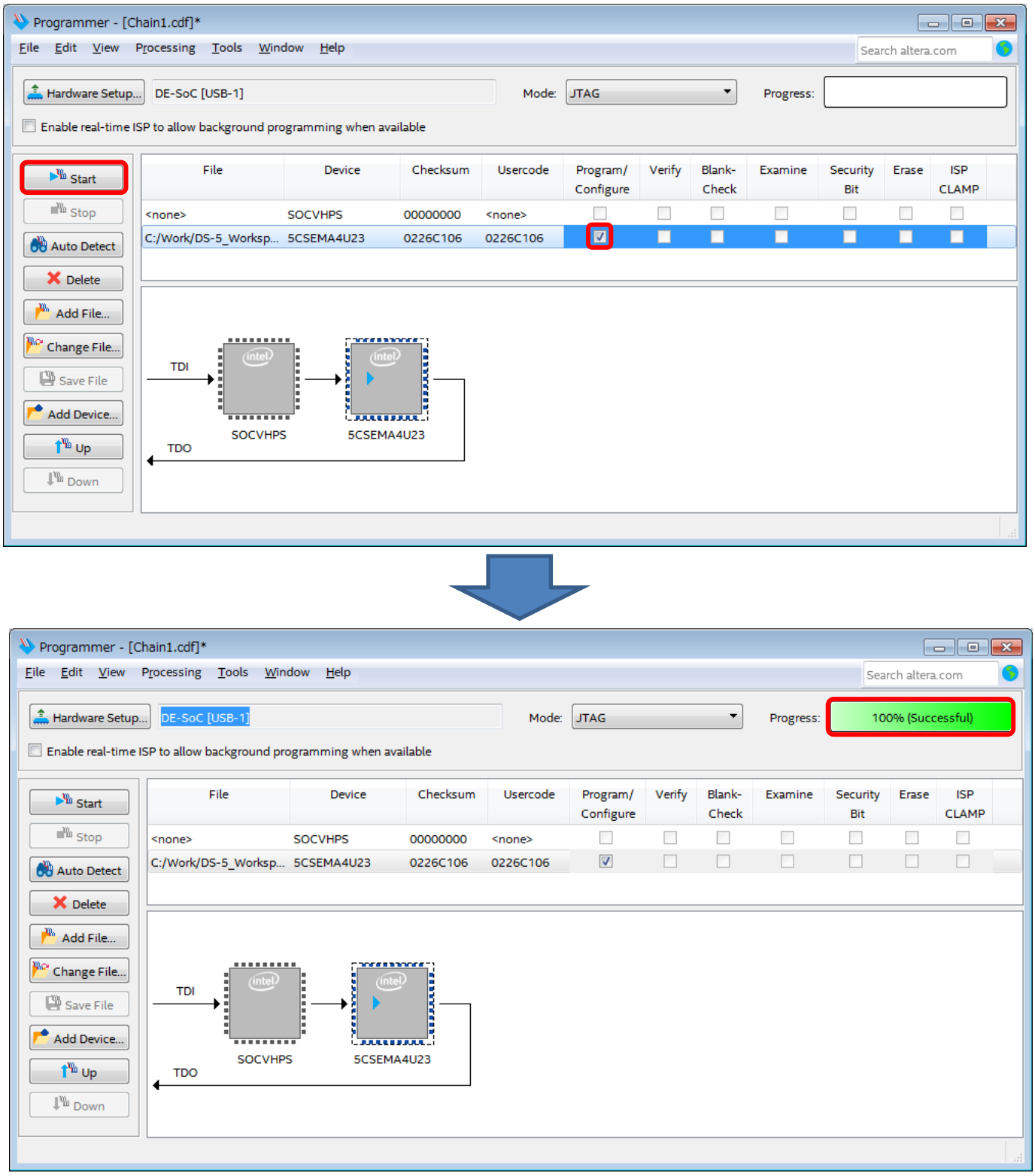


Figure 413: Download sof (Atlas-SoC board example)

4-7. Debugging the Bare Metal Sample Application

The next step is to debug the Bare Metal sample application that you built.

Before debugging, refer to section 4-1 Connecting the target board and make sure that the cabling between the host PC and the target board and the power on the target board are complete.

Also, start the serial terminal (Tera Term is used in this document) and make the following settings for the valid COM port that is connected to the UART on the target board to enable terminal I/O.

- Baud rate 115200 bps
- 8 bit data
- No parity
- 1 stop bit
- No flow control

4-7-1. Run debugging

____ 1. Highlight the bare metal sample application project **ALT-HWLib-All-In-One_v22.1_r0.0**, right click and select **Debug As** > **Debug Configurations....**

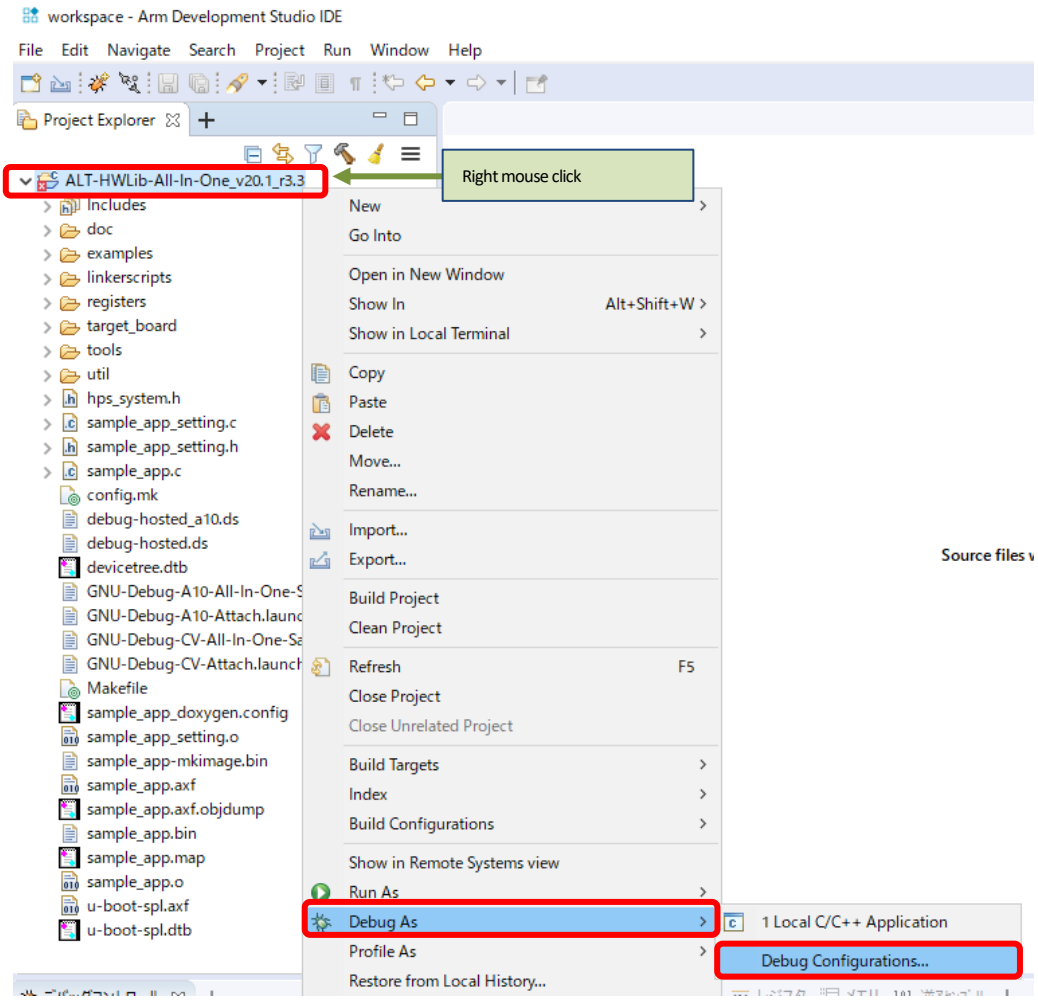


Figure 414 Debug As > Debug Configurations...

____ 2. From the left panel of the Debug Configuration window, click Generic ARM C/C++ Application > **GNU-Debug-<device>-All-In-One-Sample** (If not, click the (+) next to Generic ARM C/C++ Application).

ⓘ Note:

In this manual, the ***Atlas-SoC*** board is used as an example. The debug configuration is set to "*General ARM C/C++ Application*" ⇒ "*GNU-Debug-CV-All-In-One-Sample*" as shown below.

The debug configuration is set to "*Intel SoC FPGA*" ⇒ "*Cyclone V SoC (Dual Core)*" ⇒ "*Bare Metal Debug*" ⇒ "*Debug Cortex-A9_0*" using a USB-BlasterTM II as the target connection.

Select the debug configuration shown in the table below according to your target board.

[Table 2] Debug configuration according to the target board

No.	Target board	Debug configuration
1	Intel ® Arria® 10 SoC Development Kit	GNU-Debug-A10-All-In-One-Sample
2	Cyclone® V SoC Development Kit	GNU-Debug-CV-All-In-One-Sample
3	Sodia-Cyclone ® V ST SoC Evaluation Board	
4	DE0-Nano-SoC Board/Atlas-SoC Board	
5	DE10-Nano Board	

____ 3. Press the **Browse ...** button on the right side of the connection section to display the selection screen for the USB-BlasterTM connection.

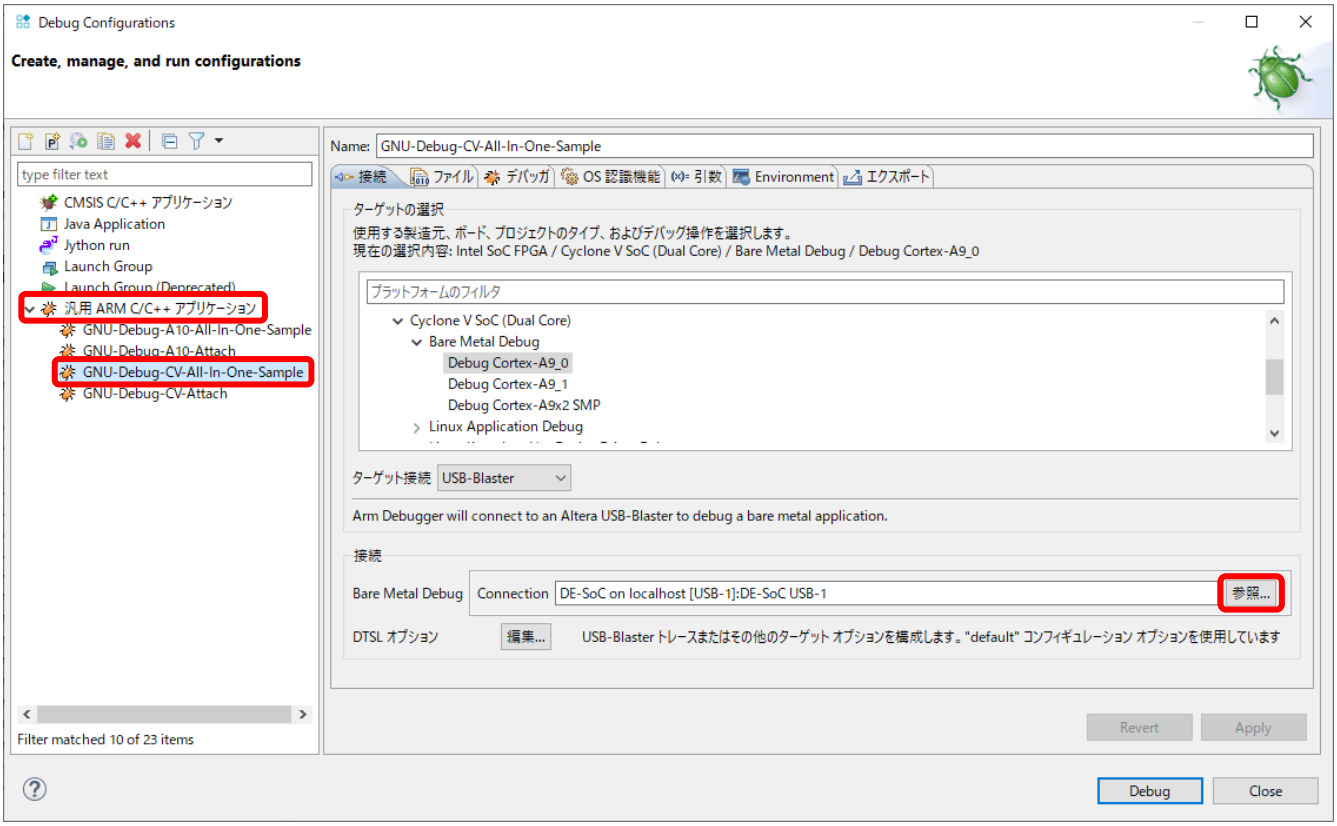


Figure 415: Debug Configuration

____ 4. In the connection browser window, highlight the desired target connection and click **Select**.

ⓘ Note:

In this document, the ***Atlas-SoC*** board is used as an example, so ***DE-SoC on localhost*** is selected as the target connection as shown below.

Select the target connection from the table below according to your target board.

[Table 3] Target connection corresponding to the target board

No.	Target Board	Target Connection
1	Cyclone® V SoC Development Kit	USB-BlasterII on localhost
2	Intel ® Arria® 10 SoC Development Kit	USB-BlasterII on localhost
3	Sodia-Cyclone ® V ST SoC Evaluation Board	USB-BlasterII on localhost
4	DE0-Nano-SoC board/Atlas-SoC board	DE-SoC on localhost
5	DE10-Nano board	DE-SoC on localhost

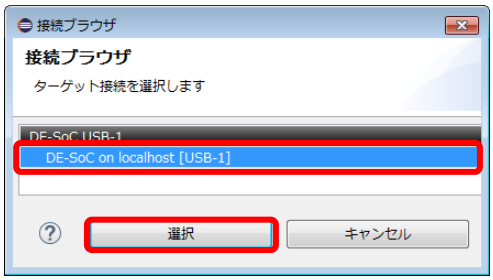


Figure 416 Selecting a Debug Cable

____ 5. Click the Debug button at the bottom right of the **Debug Configurations** window.

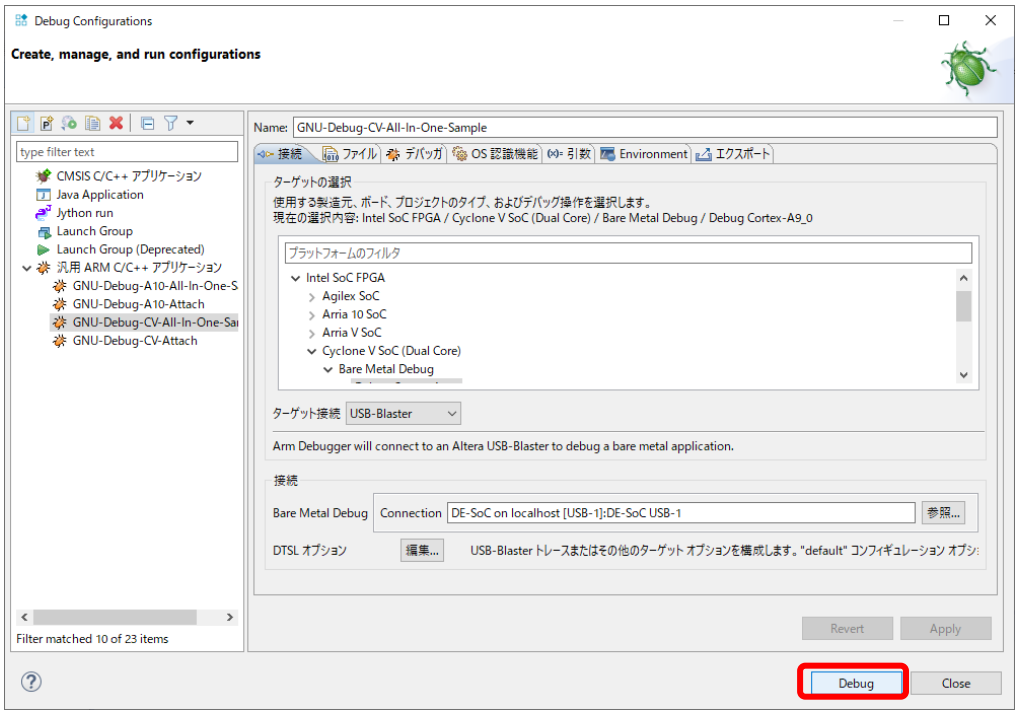


Figure 417 Running Debugging

____ 6. If prompted to confirm the perspective switch, click **Yes** to accept it.

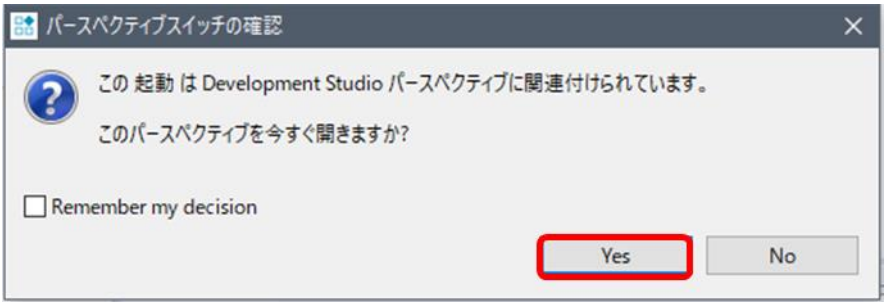


Figure 418 Checking the Perspective Switch

If you receive a Windows Defender Firewall warning, click **Allow Access**.

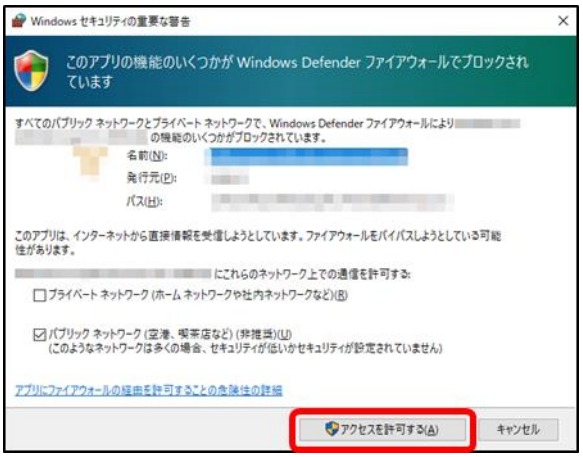


Figure 19 Security Warning

ⓘ **Note:**

If you receive an error during download, check the following:

(1) Make sure the network interface (e.g. USB-Ethernet Interface Adapter) to which the Arm DS license is associated is enabled.

(2) Make sure that the evaluation board is powered off and the PC is rebooted. If the evaluation board is powered off, remember to download the FPGA data again.

The debugger follows the instructions in the startup script to enable the semi-hosting feature and then downloads the application to the board via JTAG. When the program counter reaches the main function, it is broken and ready to start debugging. At this stage, you can use all the debugging features of Arm DS (View and edit registers and variables, reference disassembly code, etc.).

____ 7. Green **Continue** button (or press F8) to run the application.

This will display the execution of the Bare Metal Sample application to a valid COM port on the Terminal (Tera Term) connected to the UART on the target board.

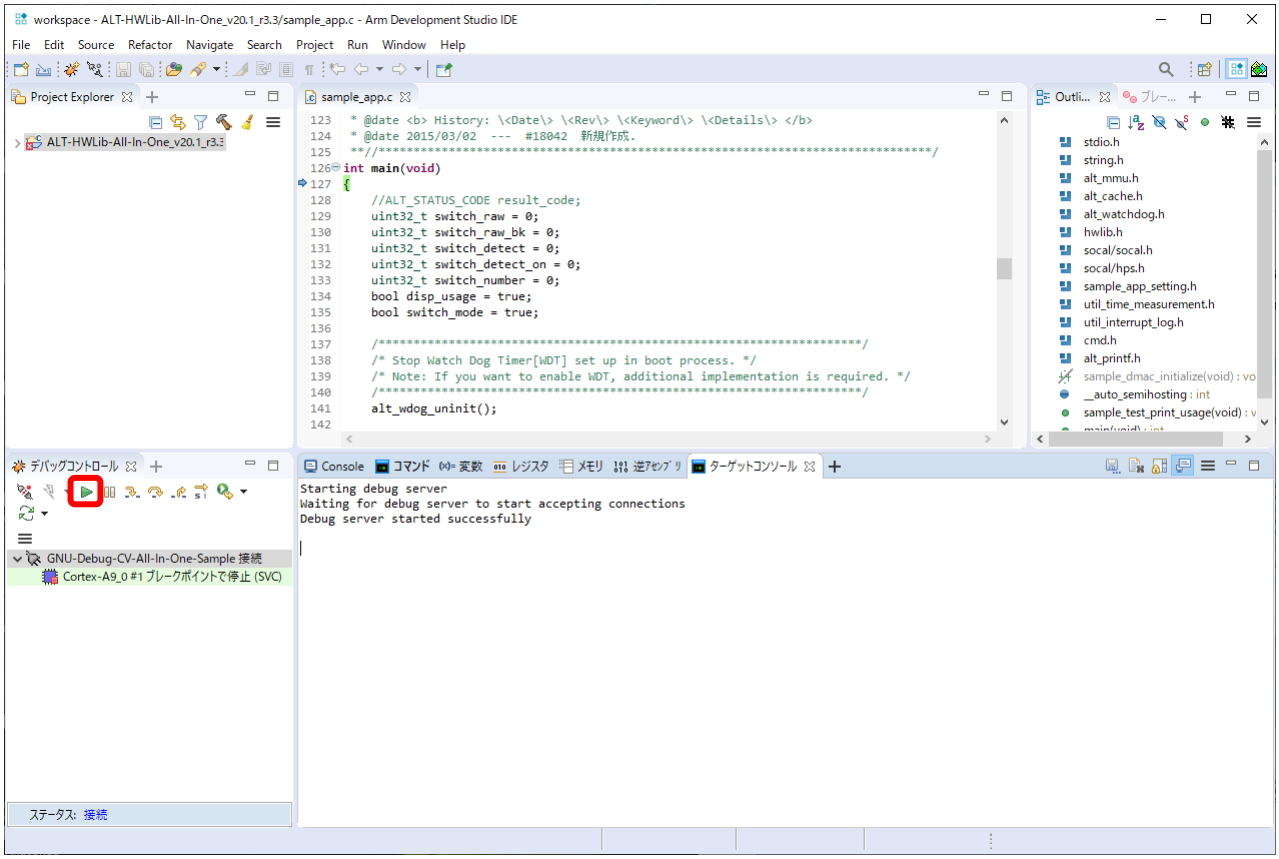


Figure 420 Running/Debugging the Application

____ 8. If the DIP switch of the target board is set to "Command mode" (described later), the command menu will be displayed in the terminal (Tera Term) as shown below, and the command will be waiting for input.

Figure 421: Bare Metal Sample Application Execution in Command Mode

____ 9. To finish debugging, click the **Disconnect from Target** button to disconnect from the CPU, and then click the Remove All Connections button to remove the target.  Click the shortcut button on the menu bar to switch back to the original C/C + + perspective.  Click the Delete button to delete the target.

____ 10. Menu bar shortcut button  to switch back to the original C/C++ perspective.

5. Basic Behavior of This Sample

Depending on the bit 0 state of the DIP switch on the target board, it is possible to switch the operation to one of the following two modes for verification:

① ON: **Switch mode** (check DIP switch/PUSH switch operation)

② OFF: **Command mode** (execution of various tests by command input)

5-1. Switch mode

Confirms the settings of the DIP switch and PUSH switch, and outputs the setting status to a message.

5-2. Command mode

Commands registered in COMMANDS_LIST commands[] can be executed. The following commands are registered by default.

User-created processes can also be registered as commands and executed.

Table 5. Commands Registered by Default

Command	Command Line	Execution function
Menu display command	**menu**	cmd_menu()
Memory read command	**mr** <type:8/16/32> <addr(HEX)>	cmd_mem_read()
Memory write command	**mw** <type:8/16/32> <addr(HEX)> <data(HEX)>	cmd_mem_write()
Memory dump command	**md** <type:8/16/32> <addr(HEX)> <size(HEX)>	cmd_mem_dump()
Memory fill command	**mf** <type(0:inc/1:fixed> <data(HEX)> <addr(HEX)> <size(HEX)>	cmd_mem_fill()
Exit command	**exit**	cmd_exit()

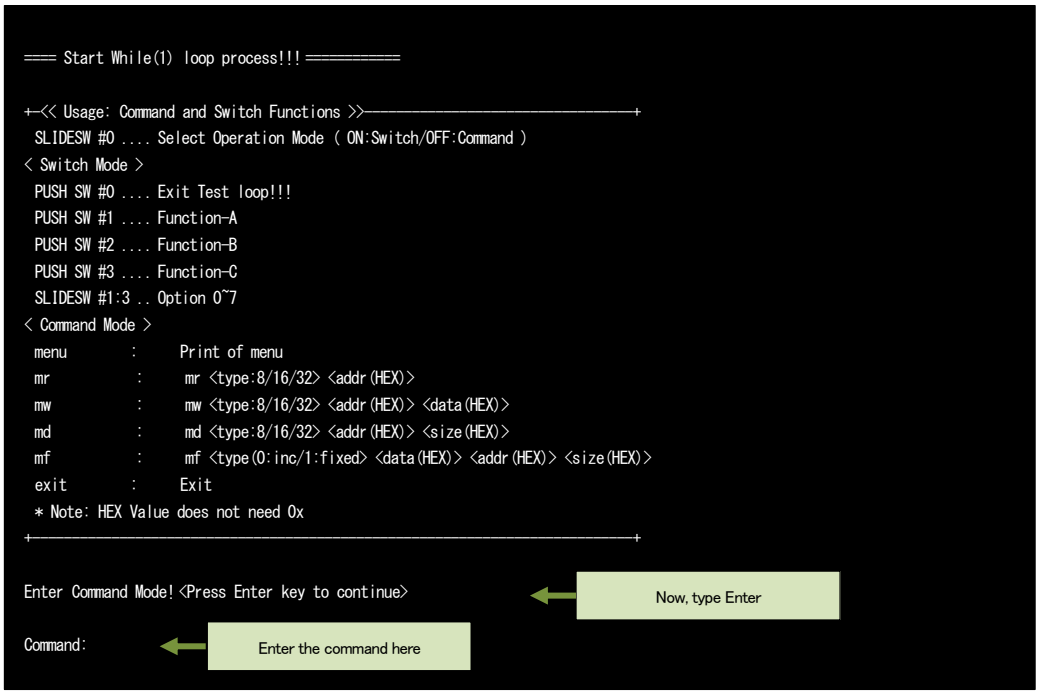


Figure 5: Command mode menu display by default

6. Description of the main routine source code for this sample

The following is an excerpt of the main routine for this sample (in sample_app.c).

Describes the main routine source code for this sample.

```
/*****************************************************************************
 *  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()
Function to show usage of this sample test

cmd_menu()
Display command menus registered in COMMANDS_LIST commands[]

main()
Main function for this sample

util_intlog_init() Initialization function for interrupt logging utility

cpu0_init() Initialization function for CPU0

Continued on the next page

```
/* ########################################## */
/* ## 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() Utility function for interrupt logging

sample_detect_switch() Function to get switch state

sample_test_print_usage() Function to show usage of this sample

sample_detect_switch() Function to get the switch state

If push switch 0 is ON,
exit the while loop of this test

If DIP switch bit 0 is ON,
put the operation into Switch mode

If DIP switch bit 0 is OFF,
put the operation into Command mode
and wait for the Enter key.

In Switch mode,

If DIP switch bit 1 is ON, a message will be displayed.
Display a message

If DIP switch bit 2 is ON,
a message will be displayed.

If DIP switch bit 3 is ON,
a message will be displayed.

Continued on the next page

```
            // 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;
}
```

If push switch 1 is turned ON, a message is displayed.

If push switch 2 is turned ON, a message is displayed.

If push switch 3 is turned ON, a message is displayed.

In the Command mode,

cmd_execute() Executes command input processing.
Exits the while loop of this test when the "exit" command is entered.

util_intlog_print() Interrupt logging utility function

util_intlog_print() Interrupt logging utility function

End

[Listing 1] Main routine source code for this sample6

7. Introduction to useful utility functions

The util directory contains useful utility functions.

Some typical utility functions are listed below.

[Table 1] MMU configuration utility functions7

File	MMU configuration utility functions	Description
l2mmu_setting.c l2mmu_setting.h	void **sample_mmu_init_and_enable** (void)	Initialize and enable the MMU by doing the following • Initialize the MMU • Create the MMU table • Enable MMU
	int **cpu0_l2mmu_init** (void)	Do the following: • Enable SIMD and VFP • ACTLR.SMP = 1/NSACR.NS_SMP = 1 • Initialize SCU (SCU is a shared resource of MPCore) • Set AXI transaction signal (AxUSER [0] = 1). This setting is required for coherent forwarding with ACP • Initialize GIC distributor register • Set and enable MMU • Enable Cache • Initializes and enables CPU interrupts. • Enables GIC global interrupts (Core0 only).

Table 7 Memory Access Utility Functions

File	Memory Access Utility Functions	Description
mem_util.c mem_util.h	void **sample_memset_address32** (uint32_t* start, size_t size)	Sets the 32 bit address value in memory.
	void **sample_memset_incrementdata** (uint32_t* start, uint32_t testdata, size_t size)	Sets the increment data of 1 in memory.
	void **sample_memset_incrementdata_4byte** (uint32_t* start, uint32_t testdata, size_t size)	Sets the increment data of 4 in memory.
	void **sample_memdmp_word** (const uint32_t* start, size_t size)	Dump memory in 32 bit size
	void **sample_memdmp_halfword** (const uint16_t* start, size_t size)	Dump memory in 16 bit size
	void **sample_memdmp_byte** (const uint8_t* start, size_t size)	Dump memory in 8 bit size

Table 2 usleep utility functions7

File	usleep utility functions	Description
usleep_soc.c	void **usleep** (uint32_t us)	Inserts a specified microsecond sleep using a global timer.

Table 3 Interrupt logging utility functions7

File	Interrupt logging utility functions	Description
util_interrupt_log.c util_interrupt_log.h	void **util_intlog_init** (void)	Interrupt log initialization processing: Always called first
	void **util_intlog_record** (ALT_INT_INTERRUPT_t kind, int opt1, int opt2)	Interrupt log recording processing: Called in an interrupt routine
	void **util_intlog_print** (void)	Interrupt log output processing: Called periodically in a normal routine

Table 4 Timing utility functions7

File	Timing utility functions	Description
util_time_measurement.c util_time_measurement.h	void **util_time_init** (void)	Initializes the time measurement program ・ Prints clock setting information ・ Sets the global timer for measurement ・ Initializes measurement recording information
	void **util_time_uninit** (void)	Uninitializes the time measurement program ・ Uninitializes the global timer for measurement ・ Prints all measurement results and clears the measurement record information
	void **util_time_record_start_point** (uint32_t index)	Records the start point of the time measurement
	void **util_time_record_end_point** (uint32_t index)	Records the end point of the time measurement
	void **util_time_print_result_by_counter** (uint32_t index)	Prints the measurement results on the counter (prints the heading)
	void **util_time_print_result_by_seconds** (uint32_t index)	Prints measurement results in seconds (print headings)
	void **util_time_print_result_all** (UtilTimePrintTarget_et printby)	Prints all measurement results
	void **util_time_print_result_partial** (int startid, int endid, UtilTimePrintTarget_et printby)	Prints partial measurement results with the specified content
	void **util_time_print_result_all_and_clear** (UtilTimePrintTarget_et printby)	Print all measurement results and clear all records

8. What is HWLib (Hardware Library)?

Used for bare metal applications, HWLib

- reduces the complexity of writing low-level SoC software (no need to write your own SoC register definitions, etc.)
- It abstracts all system registers
- A usable layer for bare metal applications, OS drivers, OS kernels, etc.
- Contains tested functionality for basic system operation (For example, changes in clock speed, cache settings, FPGA configuration, etc.)

8-1. HWLib Components

HWLib consists of two components.

- SoC abstraction layer (SoCAL) - (low-level HAL)
  - Macro-based abstraction layer (header file) for accessing hardware IP registers
  - Separates software and hardware
- Hardware Manager (HWMgr)
  - A collection of C and assembly APIs for high-level access to SoC hardware.
  - `Include SoCAL header files with #include`

The util/hwlib directory for this sample project contains all the sources provided by Intel as HWLib, and you can use all the APIs by including the HWLib header files you want to use.

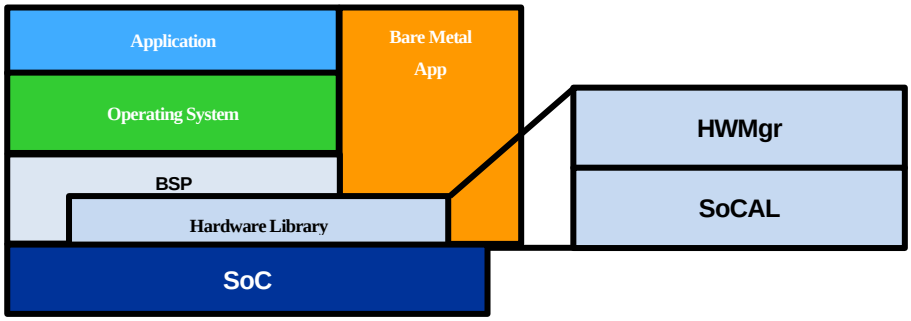


Figure 8: HWLib Components

8-2. HWLib Configuration (Functions with API)

The following HWLib API is provided.

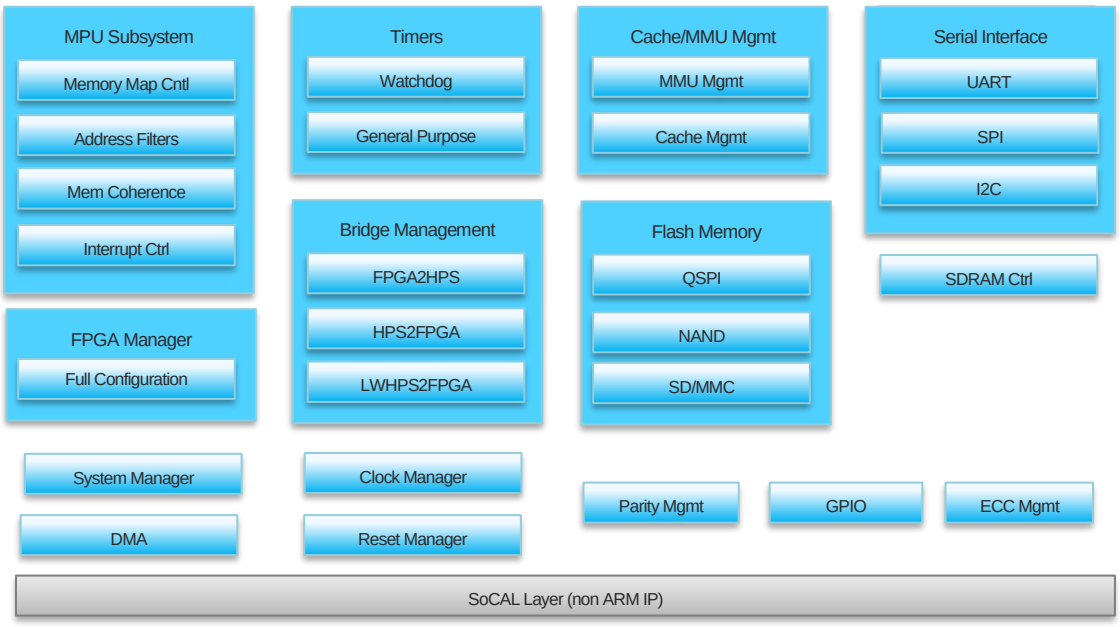


Figure 1 HWLib API8

8-3. HWLib Documentation

- Location for SoCAL related documentation
  - *<SoC EDS installation directory>/ip/altera/hps/altera_hps/doc/***<device_name>/socal/html/index.html**
    - **<device_name>** For Cyclone V/Arria V: **soc_cv_av**
    - For Arria10: **soc_a10**
- Location for HW Manager related documentation
  - *<SoC EDS installation directory>/ip/*altera/hps/altera_hps/doc/hwmgr/**<device name>**/index.html
- Also accessible from the Windows Start menu.

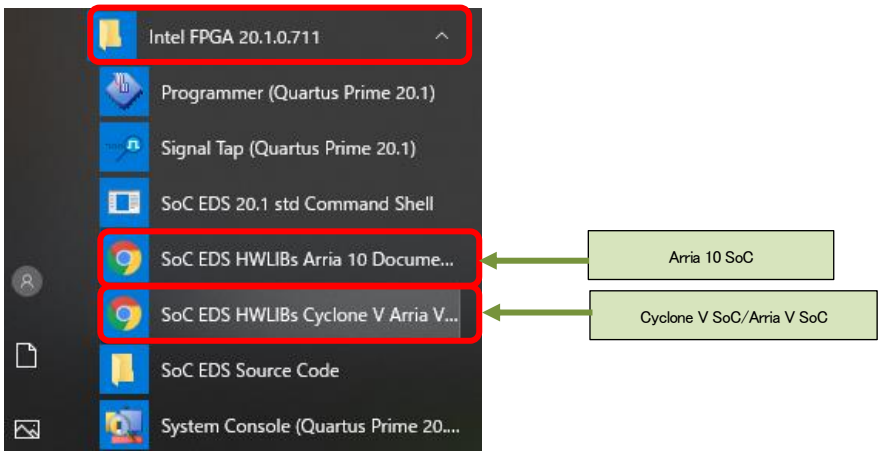


Figure 2 Accessing the HWLib Documentation from the Windows Start Menu8

9. HWLib Examples

The examples directory contains various HWLib software source codes.

- sample_cache_manage.c (cache management sample program)
- sample_clock_manager.c (clock manager sample program)
- sample_dma_mem.c (DMA transfer sample program)
- sample_dmac.c (HPS DMA-330 sample program)
- sample_ecc.c (ECC management sample program)
- sample_globaltmr.c (global timer sample program)
- sample_gpio.c (GPIO sample program)
- sample_gptmr.c (general-purpose timer sample program)
- sample_interruptctrlSGI.c (interrupt controller (mainly SGI) sample program)
- sample_time_measurement.c (time measurement implementation sample program)
- sample_watchdog.c (watchdog timer sample program)

If you use these Example source code files, you can set the corresponding Example to 1: Enable in the config.mk file in the TOP directory of your project to build it as a compilation target for Arm DS.

```
#####################################################
# Select example code to test (0:Disable/1:Enable) #
#####################################################
# ENABLE_EXAMPLE_DMA      : sample_dmac.c,sample_dma_mem.c (USED_DMA must be 1)
ENABLE_EXAMPLE_DMA        := 0
# ENABLE_EXAMPLE_CACHE    : sample_cache_manage.c
ENABLE_EXAMPLE_CACHE      := 0
# ENABLE_EXAMPLE_CLK      : sample_clock_manager.c
ENABLE_EXAMPLE_CLK        := 0
# ENABLE_EXAMPLE_ECC      : sample_ecc.c
ENABLE_EXAMPLE_ECC        := 0
# ENABLE_EXAMPLE_GLTMR    : sample_globaltmr.c
ENABLE_EXAMPLE_GLTMR      := 0
# ENABLE_EXAMPLE_GPIO     : sample_gpio.c (Arria 10 is not supported)
ENABLE_EXAMPLE_GPIO       := 0
# ENABLE_EXAMPLE_GPTMR    : sample_gptmr.c
ENABLE_EXAMPLE_GPTMR      := 0
# ENABLE_EXAMPLE_INTCTRL : sample_interruptctrlSGI.c
ENABLE_EXAMPLE_INTCTRL    := 0
# ENABLE_EXAMPLE_TIME     : sample_time_measurement.c
ENABLE_EXAMPLE_TIME       := 0
# ENABLE_EXAMPLE_WDOG     : sample_watchdog.c
ENABLE_EXAMPLE_WDOG       := 0
```

[Listing 9] Compilation settings in the config.mk file

The following pages give an overview of the various samples in the examples directory (See each source code file and readme.txt for more information).

⚠ Note:

In this HWLib Example, software operation is switched by operating 4 PUSH switches (PUSHSW below) and 4 SLIDE switches (DIPSW below) on the HPS side.

However, if the ***Atlas-SoC/DE0-Nano-SoC/DE10 Nano*** development board is selected as the target board, the switches mentioned above will be insufficient on the HPS side. Therefore, the following implementation is required.

- PUSHSW 0 ... Simultaneously pressing PUSHSW (KEY0, KEY1) on the FPGA side indicates PUSHSW0.
- PUSHSW 1 ... Simply pressing PUSHSW (KEY0) on the FPGA side indicates PUSHSW1.
- PUSHSW 2 ... Simply pressing the FPGA PUSHSW (KEY1) indicates PUSHSW 2.
- PUSHSW 3 ... Simply pressing the HPS USER PUSHSW (KEY2) indicates PUSHSW 3.
- DIPSW 0:3 ... FPGA DIPSW (SW0, SW1, SW2, SW3)

If the ***Arria 10 SoC*** development board is selected, all switches (PUSHSW x 4, DIPSW x 4) will be used on the FPGA side instead of the HPS side.

⚠ Note:

sample_gpio.c *(GPIO sample program)* does not support the ***Arria® 10 SoC*** *Development Kit (a10socdk) because GPIO-connected HPS user switches are not available.*

9-1. sample_cache_manage.c (Cache Management Sample Program)

・ Table 9 sample_cache_manage.c Source File

<table>
<tr><td>Source File</td><td>sample_cache_manage.c</td></tr>
<tr><td>TOP Function Name</td><td>int sample_cache_manage_test_cmd(char* options)</td></tr>
<tr><td>Overview</td><td>Cache Management Sample Program</td></tr>
<tr><td>Function</td><td>This program tries all the APIs in the following categories in HWLib.
Cache Management API
+ System Level Cache Management API
+ L1 Cache Management API
+ L2 Cache Management API</td></tr>
<tr><td>Sample Functions</td><td>The following is an overview of the sample functions.
① sample_cache_manage_init();
⟶ GPIO settings for DIPSW and PUSHSW for HPS on the target board (for operating the test program)
⟶ Execute HWLib API to enable all L1 and L2 Caches.
⟶ Change interrupt controller (GIC) settings to check operation.
Enable L2 Cache Combined IRQ "ALT_INT_INTERRUPT_L2_COMBINED_IRQ."
The above interrupt is issued under the condition that all 3 types of abnormal notifications from L2 Cache Controller are combined (OR).
This sample does not implement a mechanism that causes Cache abnormality.
If a Cache abnormality is reproduced by some method, an interrupt occurs and the following console message is output (not verified).
"[INTERRUPT]L2 Cache Combined Interrupt is occurred!! status=0x0000****"
⟶ Configure the L2 Cache Controller to enable the interrupt notification function (Enable using HWLib).

② sample_cache_manage_test_main();
⟶ Run the test program. Perform the following processing in the infinite loop.
<table><tr><th>DISPSW
[4321]</th><th>PUSHSW0
Press</th><th>PUSHSW1
Press</th><th>PUSHSW2
Press</th><th>PUSHSW3
Press</th></tr><tr><td>xxx1</td><td>End the infinite loop
(End test program)</td><td>–</td><td>–</td><td rowspan="9">To verify the cache effect
Start the function mul_f32_test_function to measure and display the processing time

⚠ Note:
In this sample, MMU is not set, so the effect of caching cannot be confirmed. If necessary, add MMU settings and check.</td></tr><tr><td>xxx0</td><td>Execute API to enable/disable all L1 and L2 Cache functions
(Enable and Disable are executed alternately every time SW is pressed)</td><td>–</td><td>–</td></tr><tr><td>0010</td><td>–</td><td>Run alt_cache_system_invalidate</td><td>Run alt_cache_l1_data_invalidate_all</td></tr><tr><td>0100</td><td>–</td><td>alt_cache_system_clean
Run</td><td>alt_cache_l1_data_clean_all
Run</td></tr><tr><td>1000</td><td>–</td><td>alt_cache_system_purge
Run</td><td>alt_cache_l1_data_purge_all
Run</td></tr><tr><td>0011</td><td>–</td><td>–</td><td>Run alt_cache_l2_data_invalidate_all</td></tr><tr><td>0101</td><td>–</td><td>–</td><td>alt_cache_l2_data_clean_all
Run</td></tr><tr><td>1001</td><td>–</td><td>–</td><td>alt_cache_l2_data_purge_all
Execute</td></tr></table></td></tr>
<tr><td>Remarks</td><td>For details, see sample_cache_manage_readme.txt and sample_cache_manage.c.</td></tr>
</table>

9-2. sample_clock_manager.c (clock manager sample program)

・ Table 1: sample_clock_manager.c Source File9

<table>
<tr><td>Source File</td><td>sample_clock_manager.c</td></tr>
<tr><td>TOP Function Name</td><td>int sample_clkmgr_test_cmd(char* options)</td></tr>
<tr><td>Overview</td><td>Clock manager sample program</td></tr>
<tr><td>Function</td><td>This sample changes the M (1 −4096) of the Main PLL at the switch timing of the DIPSW 1 −4 for HPS (to test the HPS main clock frequency switching). The Main clock is changed according to the value of the DIPSW 1 −4 for HPS as follows.
<table>
<tr><th>DISPSW
[4321]</th><th>M of the Main PLL
(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>
Try all APIs in the following categories in HWLib:
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
All APIs in the above categories are tested, and the configuration information of the three PLLs (Main PLL, Peripheral PLL, SDRAM PLL) is displayed and visible.
Interrupts generated by the Clock Manager (Lock/Unlock of the three PLLs) are also displayed on the console when they occur.</td></tr>
<tr><td>Sample functions</td><td>The following is an overview of the sample functions.
① sample_clkmgr_init();
⟶ Makes GPIO settings for DIPSW and PUSHSW for HPS on the target board (for test program operation).
⟶ Makes Global Timer settings (Timer enabled for CPU Core operation clock) (for test program operation).
⟶ Changes the interrupt controller (GIC) settings to check operation.
After setting with this function, interrupt processing is executed when PLL Lock/Unlock triggers, and the following console message is output.
″[IRQ#205] CLKMGR_IRQ (0x000000CD,0x000001C7) count=1″
② sample_clkmgr_test_print_pllconfig ();
⟶ Executes the information acquisition API of Clock Manager, and displays the following information (both trying the information acquisition API and checking the settings).
③ sample_clkmgr_test_customize_config ();
⟶ Executes the Clock Manager configuration change API (trial of the configuration change API).
④ sample_clkmgr_test_main();
⟶ Executes the test program. Performs the following processing in an infinite loop.
(1) Outputs the counter value to the console every time the last 28 bits or more of the Global Timer counter value (bit #31:28 of the lower 32 bits) change.
Outputs the counter value to the console unconditionally while PUSHSW3 for HPS is pressed (for checking the progress of the Timer counter).
(2) Attempts the API of Category: Clock Manager Status when PUSHSW2 for HPS is pressed.
Clears and displays PLL Lock/Unlock Status (attempts the Clock Manager Status API).
(3) Attempts the API of Category: Safe Mode Options when PUSHSW1 for HPS is pressed.
Displays and clears the Safe Mode status (try Safe Mode Options API).
(4) Attempts to change the Multiplier setting value of the PLL and display the clock frequency after the change at the time of change of DIPSW1−4 for HPS.
For M (1−4096) of the Main PLL, applies (4bit value × 4) to the Multiplier setting value with DIPSW1 as MSB and DIPSW4 as LSB.
(5) Ends the infinite loop at the time of press of PUSHSW0 for HPS (end of test program).</td></tr>
<tr><td>Remarks</td><td>For details, refer to sample_clock_manager_readme.txt and sample_clock_manager.c.</td></tr>
</table>

9-3. sample_dma_mem.c (DMA transfer sample program)

・ Table 2: sample_dma_mem.c Source File9

<table>
<tr><td>Source File</td><td>sample_dma_mem.c</td></tr>
<tr><td>TOP Function Name</td><td>int sample_dma_mem_test_cmd(char* options)</td></tr>
<tr><td>Overview</td><td>DMA Transfer Sample Program</td></tr>
<tr><td>Function</td><td>
This sample performs memory-to-memory DMA transfer using HPS built-in DMA (DMA-330).
This sample is implemented in an environment where MMU, L1 and L2 caches, and ACP ports are enabled.
The following options can be selected by operating PUSHSW and DIPSW on the target board.
・DMA Channel selection (0~7)
・Transfer path (via normal port (ACP not used)/ACP port)
・Enable/disable cache maintenance operations
⚠ Note: Because the Arria 10 SoC automatically determines the use of ACP ports by looking at the cache attributes of AXI transactions, the ACP Use/Not Use option has no meaning.
When the DMA test is run with the above options selected, the following 8 patterns of buffer-to-buffer transfers with different MMU settings are attempted per execution, and the processing time and result (OK/NG) of the DMA transfers are displayed.
<table>
<tr><th>Test</th><th>Source buffer</th><th>Destination buffer</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>
</td></tr>
<tr><td></td><td>
The processing procedure for DMA transfer is implemented as follows.
The processing time (*) displayed during execution is also displayed for each of the following steps.
＜DMA transfer test processing procedure＞
1. Test data storage (writing data to the transfer source/clearing 0 of the transfer destination)
2. Cache maintenance (reflecting data stored in cache memory to physical memory)
3. Microcode generation for DMA-330
4. DMA execution (start of transfer ~ DMA completion interrupt occurs)
5. Verify DMA transfer results... OK/NG display of results
⚠ Note: If DMA transfer via ACP port is used, the L1 and L2 caches are also affected, which may reduce the cache efficiency for CPU access.
When using ACP, be sure to thoroughly verify the CPU's processing performance (The processing time displayed in this sample is only a reference value.).
</td></tr>
<tr><td></td><td>
After program execution starts, when initial settings and execution tests of various APIs are completed, loop processing starts with the following display.
"==== Start While(1) loop process!!! (Exit PUSHSW0(SW8) becomes ON.) ===="
If an operation of PUSHSW for HPS is detected during the loop, the following processing is executed.
<table>
<tr><th>Switch</th><th>Processing to be executed</th></tr>
<tr><td>PUSHSW0</td><td>Exits the loop and ends the program.</td></tr>
<tr><td>PUSHSW1</td><td>DMA Register Display (View the status of Management Thread, Ch Thead)</td></tr>
<tr><td>PUSHSW2</td><td>DMA Transfer Test Run (Option Selection: Normal Port (ACP Not Used)</td></tr>
<tr><td>PUSHSW3</td><td>DMA Transfer Test Run (Option Selection: ACP Port)</td></tr>
<tr><td>DIPSW4</td><td>Option Selection: Cache Maintenance ON/OFF</td></tr>
<tr><td>DIPSW3</td><td>Option Selection: DMA Channel 0 ~ 7</td></tr>
</table>
⚠ Note: Basically, the DMA test is run by operating the PUSHSW/DIPSW while running (F8). To check the contents of the transfer data, break (F9) and refer to the memory view.
</td></tr>
<tr><td>Supplement</td><td>
The DMA transfer API in this sample uses alt_dma_custom.c, which is customized based on alt_dma.c of HWLib.
For details, see Supplement 4 on sample_dma_mem_readme.txt.
Set USED_DMA to 1 in config.mk and build.
</td></tr>
<tr><td>Sample function</td><td>
The following is an overview of the sample function.
① sample_dmac_test_init();
⟶ The configuration required to use DMA-330 (such as registering interrupt callbacks) is done in this function.
② sample_dmac_print_manager_status();
⟶ Displays the DMA Manager thread status.
③ sample_dmac_print_ch_status(channel, true);
⟶ Displays the DMA CH status currently selected in DIPSW.
④ sample_dmac_test_main(channel, acp_en, cacheope_en);
⟶ Runs the DMA transfer test. Transfers 8 patterns.
</td></tr>
<tr><td>Remarks</td><td>For details, see sample_dma_mem_readme.txt and sample_dma_mem.c.</td></tr>
</table>

9-4. sample_dmac.c (sample program using HPS DMA (DMA -330))

・ [Table 3] sample_dmac.c Source File9

Source file	sample_dmac.c
TOP Function name	int sample_dmac_test_cmd(char* options)
Overview	Sample program using HPS DMA (DMA-330)
Function	This sample performs DMA transfer using HPS built-in DMA (DMA-330). To execute this program, input the following parameters from the console. <DMA transfer source address > <DMA transfer destination address > <DMA transfer size > < transfer byte width > For example: Source address = 0x 10,000 Destination address = 0x12000 Transfer size = 0x100 Transfer byte width = 8 For, enter 10,000 12000 100 8 as follows: 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
Addendum	The DMA transfer API in this sample uses alt_dma_custom.c, which is customized based on HWLib's alt_dma.c. For more information, see Addendum on sample_dmac_readme.txt. **Set USED_DMA to 1 in config.mk and build.**
Sample Functions	The following is an overview of the sample functions. ① sample_dma_m2m_setting(bytes); ⟶ Sets DMA (DMA-330) according to the value of the bytes argument. Set bytes to 1, 2, 4, or 8. ② sample_dmac_test_main(ALT_DMA_CHANNEL_0, (void*)srcaddr, (void*)dstaddr, (size_t)size); ⟶ Call the sample DMA transfer execution function sample_dmac_test_execute(). ③ sample_dmac_test_execute(); ⟶ Call the alt_dma_channel_exec () DMA transfer execution API function.
Remarks	For details, see sample_dmac_readme.txt and sample_dmac.c.

9-5. sample_ecc.c (ECC administration sample program)

・ Table 9: sample_ecc.c Source File

<table>
<tr><td>Source File</td><td>sample_ecc.c</td></tr>
<tr><td>TOP Function Name</td><td>int sample_ecc_test_cmd(char* options)</td></tr>
<tr><td>Overview</td><td>CC Management Sample Program</td></tr>
<tr><td>Function</td><td>
Use HWLib to operate all of the APIs classified in the following categories.
Error Correcting Code (ECC) Management
Use ECC of On-Chip RAM to check the following operations.
・ECC error/injection
・Occurrence of ECC interrupt
・Read data in case of ECC error (memory check)
・ECC operation difference depending on cache enable/disable
After program execution is started, when initialization and execution tests of various APIs are completed, loop processing starts with the following display.
"==== Start While(1) loop process!!! (Exit PUSHSW0(SW8) becomes ON.) ===="

If an HPS PUSHSW operation is detected during a loop, the following processes are executed.
<table>
<tr><th>Switch</th><th>Processing to be executed</th></tr>
<tr><td>PUSHSW0</td><td>Exit the loop and end the program</td></tr>
<tr><td>PUSHSW1</td><td>Perform cache cleaning and read access to the test area (check for occurrence of ECC interrupt)</td></tr>
<tr><td>PUSHSW2</td><td>Perform cache purge (clean and disable) and read access to the test area (check for occurrence of ECC interrupt)</td></tr>
<tr><td>PUSHSW3</td><td>ECC error injection and memory check are performed according to the settings of DIPSW1 and DIPSW2.</td></tr>
</table>
DIPSW for HPS is used to select the following operations.
<table>
<tr><th>Switch</th><th>Operation selection</th></tr>
<tr><td>DIPSW1</td><td>ECC Double Bit Error (uncorrectable) Injection setting [OFF: disabled/ON: enabled]</td></tr>
<tr><td>DIPSW2</td><td>ECC Single Bit Error (correctable) Injection setting [OFF: disabled/ON: enabled]</td></tr>
<tr><td>DIPSW3</td><td>L2C-310 Debug Mode setting [OFF: disabled/ON: enabled]
(ON: When enabled, the cache enters a mode in which it operates with "forced write-through" and "line fill disabled")</td></tr>
<tr><td>DIPSW4</td><td>L1/L2 cache setting [OFF: Disabled/ON: Enabled]</td></tr>
</table>
● If you press PUSHSW3 with either DIPSW1 or DIPSW2 set to ON, ECC errors will be injected during memory check write access.
Each time a memory check NG is detected, the address, expected value, and read result of the NG will be displayed.
The first 128 bytes (0x80) of On-Chip RAM will be used as the test area.
When cache is enabled (DIPSW4 = ON), write access is set to write-back, so writing to On-Chip RAM does not work and error injection does not work.
By turning DIPSW3 ON, the write-through mode is forcibly entered, and error injection can be set to work even when cache is enabled.
● Press either PUSHSW1 or PUSHSW2 to perform read access to the memory checked area.
By performing this operation after error injection during memory check, you can check the ECC error detection operation. Displays a message each time an ECC interrupt is detected.
When cache is enabled (DIPSW4 = ON), cache maintenance processing is performed beforehand.
(PUSHSW1 performs cache clean, PUCHSW2 performs cache purge (clean and disable))
● Memory check and read access are performed by various access methods to verify operation.
– On-Chip RAM access address: 0x00000000~/0xFFFF0000~
– Bit width: 8bit/16bit/32bit/64bit</td></tr>
<tr><td>Sample function</td><td>
The following is an overview of the sample function.
① util_time_init();
⟶ Initializes processing time measurement. Performs the following operations.
・ Sets the global timer for processing time measurement (These settings are required for measurement. Currently, the measure process call is not implemented).
・ Sets the Clock Manager and displays the setting information (for checking the operating environment parameters such as various clock frequencies).
② sample_ecc_test_init();
⟶ Performs the following operations.
・ Sets the remap register (Configure On-Chip RAM to be Accessible from First Address 0x00000000).
・ Sets the GPIO of the target board (Setting PUSHSW and DIPSW for HPS).
・ Configures and enables on-chip RAM ECC (ECC Enabled and Interrupt Allowed).
・ Configures and enables MMU (On-Chip RAM space is set to write-back for both L1 and L2 caches).
③ sample_ecc_test_main();
⟶ Runs the test program. Runs the switch detection process described in "Function" above in an infinite loop.
④ sample_ecc_test_uninit();
⟶ This sample does not perform any processing.
⑤ util_time_uninit();
⟶ Performs post-processing of processing time measurement processing. Displays the measurement result on the console.
</td></tr>
<tr><td>Remarks</td><td>For details, see sample_ecc_readme.txt and sample_ecc.c.</td></tr>
</table>

9-6. sample_globaltmr.c (global timer sample program)

・ Table 4: sample_globaltmr.c Source File9

Source File	sample_globaltmr.c
TOP Function Name	`int sample_globaltmr_test_cmd(char* options)`
Overview	Global Timer Sample Program
Function	This program allows you to run all of the APIs in the following categories in HWLib. The Global Timer Manager API Run the global timer and check the operation of the following functions. ・ Check that the timer counter cycle changes according to the prescaler setting. ・ Check that the following functions operate when comparator is enabled (Global Timer Control Register is set to Comp Enable = 1). – Auto-increment function (Each time the comparator comparison result matches, it is added automatically.) – Interrupts from the global timer (This occurs when the comparator comparison result matches.)
Sample function	The following is an overview of the sample function. ① sample_globaltmr_alldisable(); ⟶ Sets all functions of the global timer to disable/stop (try the API for setting disable/stop). ② sample_globaltmr_setting_gic(); ⟶ Change the interrupt controller (GIC) settings to check operation. ・ After setting with this function, the interrupt operates under the following conditions. – sample_globaltimer_callback() starts at the globaltimer interrupt trigger. – The 8 ports of DIPSW and PUSHSW for HPS are not used as interrupt triggers (leave the default without changing the settings). ③ sample_globaltmr_paraminit(); ⟶ Global timer operation Try the API for parameter setting (trial and initial setting of the API for parameter setting). ・ The following 3 parameters are set. – Set the pre-scaler setting value (timer clock division ratio) to 0 (The input clock [CPU core operating clock 800MHz] is applied as it is to the tie mark lock. When set to 1, the input clock is divided into 400 MHz, when set to 2, the input clock is divided into 200 MHz, and so on.). – Set the auto-increment function addition register value (32 bits) to 0x40 million. Each time the comparator comparison results match, the register value is automatically added to the comparator. – Set the current value of the global timer + 0x0000000040000000 in the comparator (register for comparing 64bit counters). ④ sample_globaltmr_allenable(); ⟶ Set all functions of the global timer to enable/start (try enable/start setting API). ⑤ sample_globaltmr_print_get_result(); ⟶ Try various register values and status acquisition APIs (try information acquisition API and check settings). ⑥ sample_globaltmr_test_main(); ⟶ Execute the test program. ・ Perform the following processing in an infinite loop. – The counter value is output to the console every time the lower 26 bits or more of the timer counter value (bit #31:26 of the lower 32 bits) change. The counter value is output to the console unconditionally while PUSHSW3 for HPS is pressed (for checking the progress of the timer counter). – Changes the comparator register value (64 bits) when PUSHSW1 for HPS is pressed. Sets the upper 32bits to the upper 32bit value of the counter + 1, and the lower 32bits to the lower 32bit value of the timer counter. Outputs the changed value to the console (slightly advances the comparator register value to check comparator operation). – Outputs the comparator register value (64 bits) to the console when PUSHSW2 for HPS is pressed (to check the auto-increment function). – Changes the prescaler setting value when DIPSW1 -4 for HPS is pressed. Sets the 4bit value as MSB for DIPSW1 and LSB for DIPSW4 to the prescaler, and outputs the setting to the console (You can see that the operating speed of the timer counter changes in relation to the set value of the prescaler.). – Ends the infinite loop when PUSHSW0 for HPS is pressed (terminates the test program). ⑦ sample_globaltmr_callback(); ⟶ Outputs to the console that an interrupt has occurred, and clears the interrupt status.
Remarks	For details, refer to sample_globaltmr_readme.txt and sample_globaltmr.c.

9-7. sample_gpio.c (GPIO sample program)

・ Table 5 sample_gpio.c Source File9

Source	sample_gpio.c
TOP Function Name	`int sample_gpio_test_cmd(char* options)`
Summary	GPIO Sample Program
Function	⚠ Note: This sample program does not support the Intel ® Arria® 10 SoC Development Kit (a10socdk) because a GPIO-connected HPS user switch is not available. The API classified into the following categories in HWLib is operated as a whole. 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 GPIO is configured so that the target board's HPS DIPSW and PUSHSW can be used as input signals, and changes in the GPIO input register values are displayed on the debugger console in conjunction with each switch operation.
Sample Functions	The following is an overview of the sample functions. ① sample_gpio_utility(); ⟶ Category: General-Purpose IO Utility Functions API (GPIO Utility API Trial). ② sample_gpio_utility(); ⟶ Category: General-Purpose IO Configuration Functions and General-Purpose IO via Bit Index API (GPIO Configuration API Trial). ③ sample_gpio_iconfig(); ⟶ Category: General-Purpose IO Interrupt Functions API (GPIO Interrupt Configuration API Trial). ④ sample_gpio_iointerrupt(); ⟶ Change the interrupt controller (GIC) settings to check operation. ・ After setting with this function, the interrupt operates under the following conditions: – sample_gpio_callback() starts at the GPIO interrupt trigger. – Select the interrupt trigger using DIPSW1:2 for HPS. DIPSW1:2 = 0 ... Rising-Edge DIPSW1:2 = 1 ... Falling-Edge DIPSW1:2 = 2 ... Rising-Edge DIPSW1:2 = 3 ... Falling-Edge – GPIO interrupt is generated when PUSHSW3 for HPS is pressed. ⑤ sample_gpio_iopolling(); ⟶ The GPIO input register is monitored in the infinite loop, and whenever a change in the value is detected, the change is output to the debugger console. ※ The infinite loop is terminated (the test program is terminated) when PUSHSW0 for HPS is pressed. ⑥ sample_gpio_callback(); ⟶ The interrupt occurrence and interrupt status value are recorded in the log buffer, and the interrupt status is cleared. Based on the information recorded in the log buffer, it is output to the console after the normal routine returns.
Remarks	For details, see sample_gpio_readme.txt and sample_gpio.c.

9-8. sample_gptmr.c (General-Purpose Timer Sample Program)

・ Table 9: sample_gptmr.c Source File

Source File	sample_gptmr.c
TOP Function Name	`int sample_gptmr_test_cmd(char* options)`
Overview	General-Purpose Timer Sample Program
Function	Use HWLib to run all of the APIs classified into the following categories. The General Purpose Timer Manager API + Enable, Disable, and Status + Counters Interface + Interrupts + Mode Control Display all of the initial values of General Purpose Timer (Hereinafter, GPT) related registers that can be referenced from HWLib, and start all of the GPTs (4 below). ・ OSC1 timer 0 … 32bit timer running with osc1_clk (fixed operation clock) ・ OSC1 timer 1 … 32bit timer operated by osc1_clk (fixed operating clock) ・ SP timer 0 ….. 32bit timer operated by l4_sp_clk ・ SP timer 1 ….. 32bit timer operated by l4_sp_clk ⓘ **Note:** The OSC1 timer uses the external clock (osc1_clk) as its operating clock (fixed). ⓘ **Note:** The SP timer is variable because it uses the Main PLL (C1: main_base_clk) or Peripheral PLL (C4: peripheral _base_clk) as its clock source (* When the operation clock is changed, a note to stop the timer is written in the manual.). In addition, all interrupts (4 lines) triggered by each GPT timeout are enabled, and the console is displayed when they occur.
Sample Functions	The following is an overview of the sample functions. ① sample_gptmr_test_init(); ⟶ Initialize the test program. ・ Perform the following processing. – Initialize HWLib for GPT (alt_gpt_all_tmr_init). – Display all GPT default values. – Change GPT settings for testing. ✓ OSC1 timer 0 … Timeout interrupt occurs every 20 seconds. (mode=ALT_GPT_RESTART_MODE_PERIODIC(User-defined count mode), resetcount=500000000(@25MHz)) ✓ OSC1 timer 1 … Timeout interrupt occurs every 40 seconds (mode=ALT_GPT_RESTART_MODE_PERIODIC(User-defined count mode), resetcount=1000000000(@25MHz)) ✓ SP timer 0 ….. Timeout interrupt occurs every 5 seconds (mode=ALT_GPT_RESTART_MODE_PERIODIC(User-defined count mode), resetcount=500000000(@100MHz)) ✓ SP timer 1 ….. Timeout interrupt occurs every 10 seconds (mode=ALT_GPT_RESTART_MODE_PERIODIC(User-defined count mode), resetcount=1000000000(@100MHz)) *For details, see the sample_gptmr_test_init code. – Displays all GPT settings after the change. – Displays interrupt controller settings and set values. ② sample_gptmr_test_main(); ⟶ Runs the test program. ・ The following processing is performed in an infinite loop. – The operation mode of each GPT is changed when the HPS DIPSW1 -4 is changed. When the mode is changed, the timer is restarted. The correspondence of each DIPSW is as follows. DIPSW1 .. OSC1 timer 0 DIPSW2 .. OSC1 timer 1 DIPSW3 .. SP timer 0 DIPSW4 .. SP timer 1 ※ Each DIPSW is PERIODIC (User-defined count mode) when ON, or ONESHOT (Free-running mode) when OFF. – When PUSHSW1 for HPS is pressed, the counter values of the four GPTs and the remaining time until timeout (milliseconds) are displayed. – When PUSHSW2 for HPS is pressed, the operation information (mode, timer enable, interrupt enable, interrupt pending) of the four GPTs is displayed. – When PUSHSW3 for HPS is pressed, all four GPTs are restarted (reset). – When PUSHSW0 for HPS is pressed, the infinite loop is terminated (the test program is terminated). ③ sample_gptmr_test_uninit(); ⟶ Performs HWLib postprocessing for GPT (alt_gpt_all_tmr_uninit).
Remarks	For details, refer to sample_gptmr_readme.txt and sample_gptmr.c.

9-9. sample_interruptctrlSGI.c (interrupt controller (mainly SGI) sample program)

・ Table 6: sample_interruptctrlSGI.c Source File9

Source	sample_interruptctrlSGI.c
TOP Function Name	`int sample_intctrl_test_cmd(char* options)`
Overview	Interrupt controller (mainly SGI) sample program
Function	This program performs a complete set of HWLib APIs classified into the following categories. 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] However, the API used for interrupt setting in other samples has already been verified in other samples, so it is omitted.
	In this sample, settings are made to run Software Generated Interrupt(hereafter, SGI), and SGI is issued when PUSHSW for HPS is operated. Note that this sample is created as a single-core project, so it is not possible to check interrupt notifications between cores in multi-cores. (The processing to issue SGI to other cores is also executed in this sample.). During the program initialization process, the callback function is registered and the interrupt setting (Distributor activation and priority setting) is performed for all SGI interrupts (16 as shown below). 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:** The values to the right of the above list are priority settings.
	Select the target SGI(ALT_INT_INTERRUPT_SGI0 to ALT_INT_INTERRUPT_SGI15) with the HPS DIPSW (4bit) value, and press any of the HPS PUSHSW1 ~ 3 to issue an SGI interrupt. Switch: Issue an SGI interrupt PUSHSW1: Notifies the SGI source core of the interrupt (addressed to Core# 0) → SGI callback works in Core#0. PUSHSW2: Notify all cores except the SGI source core (to Core#1) → No response since Core#1 is not moved PUSHSW3: Notify all cores including the SGI source core (To Core#0, Core#1) → The callback works only on Core#0. Core#1 does not respond When the SGI callback function works, a message in the following format is displayed on the DS-5 application console (In parentheses "(xxx, yyy)," the first value (xxx) is the value of icciar, the second value (yyy) is the detection count counter for that IRQ, and the value of count=* is the total detection count for all IRQs.). `"[IRQ#0] SGI0 (0x00000000,0x0000000A) count=11"`
Sample function	The following is an overview of the sample function. ① sample_intctrl_test_init(); → Initialize the test program. Perform the following processing: ・ Attempt API for interrupt controller enable setting (alt_int_cpu_enable_ns/all, alt_int_global_enable_ns/all). ・ Execution of interrupt controller initialization API. ・ Display interrupt controller settings and values, such as SGI settings and callback registration. ・ Attempt API for getting interrupt controller CPU interface parameters. – alt_int_cpu_config_get() – alt_int_cpu_priority_mask_get() – alt_int_cpu_binary_point_get() – alt_int_cpu_binary_point_get_ns() ・ Attempt parameter setting API of interrupt controller CPU interface. – alt_int_cpu_config_set() – alt_int_cpu_priority_mask_set() – alt_int_cpu_binary_point_set() – alt_int_cpu_binary_point_set_ns() ・ Execution of interrupt controller enable setting API (alt_int_cpu_enable, alt_int_global_enable). ② sample_intctrl_test_main(); → Execute test program. ・ The following processing is performed in an infinite loop. – The SGI issue API is executed when PUSHSW1 for HPS is pressed. Specify the following parameters to issue SGI to Core#0. int_id (1st Argument) = DIPSW for HPS (4 bits) Specify 0 (ALT_INT_INTERRUPT_SGI0) to 15 (ALT_INT_INTERRUPT_SGI15). target_filter (2nd Argument) = ALT_INT_SGI_TARGET_SENDER_ONLY (specify the source only)

	– Execute the SGI issue API when PUSHSW2 for HPS is pressed. Specify the following parameters to issue SGI to Core#1. int_id (1st Argument) = DIPSW for HPS (4 bits) Specify 0 (ALT_INT_INTERRUPT_SGI0) to 15 (ALT_INT_INTERRUPT_SGI15). target_filter (2nd Argument) = ALT_INT_SGI_TARGET_ALL_EXCL_SENDER (specify all except the issuer) – Execute the SGI issue API when PUSHSW3 for HPS is pressed. SGI is issued to Core#0 and Core#1 by specifying the following parameters. int_id (1st Argument) = DIPSW for HPS (4 bits) Specify 0 (ALT_INT_INTERRUPT_SGI0) to 15 (ALT_INT_INTERRUPT_SGI15). target_filter (2nd Argument) = ALT_INT_SGI_TARGET_LIST (target_list specifies the destination) target_list (3rd Argument) = 3 (bit#0[Core#0]=1 && bit#1[Core#1]=1) – Ends the infinite loop when PUSHSW0 for HPS is pressed (terminates the test program). ③ sample_intctrl_test_uninit(); • Attempts to disable the interrupt controller API (alt_int_cpu_disable_ns/all, alt_int_global_disable_ns/all). • Attempts to disable the interrupt controller API (alt_int_cpu_disable, alt_int_global_disable). ⓘ **Note:** You can try different settings from this sample by rewriting the following definitions in sample_interruptcontrollerSGI.c. /*! 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. */
Remarks	For details, see sample_interruptctrlSGI_readme.txt and sample_interruptctrlSGI.c.

9-10. sample_time_measurement.c (sample program to implement time measurement)

Table 97: sample_time_measurement.c Source File

Source	sample_time_measurement.c
TOP Function Name	`int sample_time_measurement_test_cmd(char* options)`
Overview	Sample program to implement time measurement
Function	This is a sample program that implements the processing time measurement mechanism using The Global Timer Manager API of HWLib. By adding the following source/header files to another project, you can measure the processing time with the same mechanism. – util_time_measurement.c – util_time_measurement.h

	● Description of the processing time measurement function
	void util_time_init(void);
	– Initialization processing. This function must be called first when using this function.
	void util_time_record_start_point(uint32_t index);
	– Implement the function CALL at the point where you want to start measurement.
	<uint32_t index>
	index is the number used to identify measurement points when measuring multiple points simultaneously. Specify an appropriate number in the range of 0~31.
	If you want to increase the number of simultaneous measurement points, you can do so by increasing the definition value of #define UTIL_TIME_MAX_TRACE in the header file.
	*The index value of the argument is implemented as the same specification for all functions.
	void util_time_record_end_point(uint32_t index);
	– Implement this function call at the end point of the measurement (The index argument must be equal to the corresponding start point).
	void util_time_print_result_by_counter(uint32_t index);
	– Displays one measurement result specified by index as a global timer counter value.
	void util_time_print_result_by_seconds(uint32_t index);
	– Displays one measurement result specified by index as a time.
	void util_time_print_result_all(void);
	– Displays all measurement results in both counter value and time.
	void util_time_print_result_all_and_clear(void);
	– Displays all measurement results in both counter value and time. Also clears all measurement result information.

	⚠ **Note:** ● The measurement results are based on the assumption that the correct time is displayed when the HPS main clock (mpu_clk) is set to 800 MHz. If you are using a different clock, you can change the following definitions in the header file. #define UTIL_TIME_GLOBALTIMER_PRESCALE 1 #define UTIL_TIME_NSEC_PER_COUNT 10 For example, if you are using a 600 MHz clock, change the following values. #define UTIL_TIME_GLOBALTIMER_PRESCALE 2 #define UTIL_TIME_NSEC_PER_COUNT 20 UTIL_TIME_GLOBALTIMER_PRESCALE is the global timer prescaler setting value. UTIL_TIME_NSEC_PER_COUNT is multiplied by the difference value of the global timer counter when calculating the time. ● Any value less than nanosecond in the global timer count period is a truncation error, so use the prescaler value as a setting value that reduces the error less than nanosecond. If you cannot change the prescaler value because you are using the global timer for another purpose, use the counter value measurement result (multiply the count period separately). ● An Excel sheet has been prepared to automatically calculate the above parameters (UTIL_TIME_GLOBALTIMER_PRESCALE, UTIL_TIME_NSEC_PER_COUNT). ->Use the file ParameterSettings_for_TimeMeasurement.xlsx in the project.
Sample function	The following is an overview of the sample function. ① sample_time_measurement_init(); ⟶ GPIO settings for DIPSW and PUSHSW for HPS on the target board (settings for test program operation). ⟶ Call initialization of time measurement processing (void util_time_init). ② sample_time_measurement_test(); ⟶ An infinite loop is performed, and the following processing is called when PUSHSW and DIPSW for HPS are operated. – DIPSW1:4 .. Used as an index value to identify the measurement target. – PUSHSW0 .. Displays all measurement results and exits the infinite loop (util_time_print_result_all) (ends the test). – PUSHSW1 .. Displays and clears all measurement results (util_time_print_result_all_and_clear). – PUSHSW2 .. Records the start of measurement with DIPSW as the index value (util_time_record_start_point). – PUSHSW3 .. Records the end of measurement with DIPSW as the index value, and displays one measurement result (util_time_record_end_point, util_time_print_result_by_counter, util_time_print_result_by_seconds). Ⓟ Point: While an infinite loop is running, press PUSHSW2, wait for any number of seconds, and then press PUSHSW3. The time corresponding to the wait time is displayed as the measurement result.
Remarks	For details, refer to sample_time_measurement_readme.txt and sample_time_measurement.c.

9-11. sample_watchdog.c (watchdog timer sample program)

・ Table 8 sample_watchdog.c Source File9

Source File	sample_watchdog.c
TOP Function Name	`int sample_wdog_test_cmd(char* options)`
Overview	Watchdog Timer (and Reset Manager) Sample Program
Function	This program performs all of the APIs classified in the following categories in HWLib. 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
	● After displaying all the initial values of watchdog timer-related registers that can be referenced from HWLib, all three watchdog timers listed below are started. – CPU Private Watchdog Timer(ALT_WDOG_CPU) – L4 Watchdog 0 (ALT_WDOG0) – L4 Watchdog 1 (ALT_WDOG1)
	● In addition, all three types of interrupts triggered by watchdog timer timeout are enabled, and a console display is performed when these interrupts occur. In addition, the following processing is performed in the interrupt callback routine of each watchdog timer. – CPU Private Watchdog Timer ... Pending Clear & Console Display Only – L4 Watchdog 0 ... Pending Clear & Console Display & COLD Reset Execution. – L4 Watchdog 1 ... Pending Clear & Console Display & WARM Reset Execution.
Sample Function	The following is an overview of the sample function. ① sample_wdog_test_init(); ⟶ It performs the following actions: – Displays the reset manager's cause register value. – Initialize the watchdog timer HWLib (alt_wdog_init). – Display all initial watchdog timer settings. – Change the watchdog timer settings for testing. ・ CPU Private Watchdog Timer (ALT_WDOG_CPU) ->FREERUN mode (the timer continues to operate even after a timeout occurs) ・ L4 Watchdog 0 (ALT_WDOG0) ->INT THEN RESET mode (Mode in which an interrupt occurs at the first timeout and WARM is reset at the second timeout) ・ L4 Watchdog 1 (ALT_WDOG1) ->INT THEN RESET mode (Mode in which an interrupt occurs at the first timeout and WARM is reset at the second timeout) (* For details, please check the code of sample_wdog_test_init directly.) – Displays all changed watchdog timer settings. – Displays interrupt controller settings and set values. ② sample_wdog_test_main(); ⟶ Runs the test program. Performs the following processing in an infinite loop. – Changes the prescaler set value for the CPU Private Watchdog Timer at the time of change of DIPSW1 -4 for HPS. Makes DIPSW1 MSB and DIPSW4 LSB, and applies the 4-bit value as it is to the set value. – Resets the CPU Private Watchdog Timer at the time of press of PUSHSW1 for HPS. – Resets L4 Watchdog 0 at the time of press of PUSHSW2 for HPS. – Resets L4 Watchdog 1 at the time of press of PUSHSW3 for HPS. – When PUSHSW0 for HPS is pressed, the infinite loop is ended (end of test program). ③ sample_wdog_test_uninit(); ⟶ The watchdog timer non-initialization API function (alt_wdog_uninit()) is called.
Remarks	For details, refer to sample_watchdog_readme.txt and sample_watchdog.c.

10. Supplementary Information

10-1. How to add user commands to be executed in the Command mode

As explained in "5-2. Command mode," you can register user-created processes as commands in COMMANDS_LIST commands[] and execute them.

① Place the source code (xxx.c) of the user you want to execute in the Top directory of this project.

② Add the following contents to the util/cmd.c file and save it.

(1) Add extern declarations for user command functions.

```
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);
```

Add extern declarations for user commands (functions).

*This example adds the sample programs in the examples directory.

[Listing 10 -1] Add extern declarations for user command functions

(2) Add descriptions of user command functions to COMMANDS_LIST commands[].

```
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}
};
```

Adds a description of a user command (function)

[Listing 10 -2] User command functions are added to COMMANDS_LIST commands[].

ALT-HWLib-All-In-One_v22.1_r○.○ Build the project again, turn off bit 0 of the DIP switch of your target board, and run the program in Command mode. As shown below, the added user commands will be displayed in the menu and the program can be executed by entering the command.

```
+-<< 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:
```

User Commands Added

[Figure 1] Added user commands are displayed in the menu10

10-2. Directory/file structure of this sample

10-2-1. ALT-HWLib-All-In-One_v22.1_r○.○ directory (TOP directory of the project)

Table 10: File structure of the ALT-HWLib-All-In-One_v22.1_r○.○ directory

ALT-HWLib-All-In-One_v22.1_r○.○ Directory	Description
examples	examples directory
linkerscripts	linkerscripts directory
registers	registers directory
target_board	target_board directory
util	util directory
config.mk	Contains instructions for compiling
debug-hosted.ds	Debug script file for Cyclone V/Arria V This file can be written to configure and automate debugging
debug-hosted_a10.ds	Debug script file for Arria10 This file can be used to configure and automate debugging.
GNU-Debug-A10-All-In-One-Sample.launch	Sample launcher file for Arria10 File for the startup configuration of the ARM DS debugger
GNU-Debug-A10-Attach.launch	Attach launcher file for Arria10
GNU-Debug-CV-All-In-One-Sample.launch	Sample launcher file for Cyclone V/Arria V File for the startup configuration of the ARM DS debugger
GNU-Debug-CV-Attach.launch	Attach launcher files for Cyclone V/Arria V
Makefile	Makefile used to build this Bare Metal sample project
sample_app.c	Main C source code file for this Bare Metal sample application
sample_app_setting.c	Configuration C source code file for this Bare Metal sample application
sample_app_setting.h	Configuration header file for this Bare Metal sample application
sample_dmac.c	DMA Test C source code files for this bare-metal sample application

10-2-2. examples directory

Table 2 File Structure of the examples Directory10

examples Directory	Description
readme.txt	Text file describing how to use Examples
sample_cache_manage.c	Cache management sample program using HWLib
sample_cache_manage_readme.txt	sample_cache_manage.c sample readme text file
sample_clock_manager.c	sample clock manager using HWLib
sample_clock_manager_readme.txt	sample_clock_manager.c sample readme text file
sample_dma_mem.c	DMA transfer sample program using HWLib
sample_dma_mem_readme.txt	sample_dma_mem.c sample readme text file
sample_dmac.c	Sample program using HPS DMA (DMA-330) using HWLib
sample_dmac_readme.txt	sample_dmac.c sample readme text file
sample_ecc.c	Error Correcting Code (henceforth, ECC) management sample program using HWLib
sample_ecc_readme.txt	sample_ecc.c sample readme text file
sample_globaltmr.c	global timer sample program using HWLib
sample_globaltmr_readme.txt	sample_globaltmr.c sample readme text file
sample_gpio.c	General Purpose I/O (hereafter, GPIO) sample program using HWLib
sample_gpio_readme.txt	sample_gpio.c sample readme text file
sample_gptmr.c	sample program for general-purpose timer using HWLib
sample_gptmr_readme.txt	sample_gptmr.c sample readme text file
sample_interruptctrlSGI.c	Sample Interrupt Controller (primarily SGI) Programs Using HWLib
sample_interruptctrlSGI_readme.txt	sample_interruptctrlSGI.c sample readme text file
sample_time_measurement.c	Sample Programs Implementing Time Measurement Using HWLib
sample_time_measurement_readme.txt	sample_time_measurement.c sample readme text file
sample_watchdog.c	Sample Watchdog Timer (and Reset Manager) Programs Using HWLib
sample_watchdog_readme.txt	sample_watchdog.c sample readme text file

10-2-3. linkerscripts directory

Table 3: linkerscripts directory file structure10

linkerscripts directory	Description
arria10-dk-ram.ld	Linker script file for Arria 10 SoC for GNU C Compiler (and later, GCC) Used for linking programs (memory allocation) in this bare metal sample.
cycloneV-dk-ram.ld	Linker script file for Cyclone V/Arria V SoC for GCC Used for program linking (memory allocation) in this bare metal sample
soc_a10-scatter.scat	Linker script file for Arria 10 SoC for ARM C Compiler (hereafter, ARMCC) Used for program linking (memory allocation) in this bare metal sample
soc_cv_av-scatter.scat	Linker script file for Cyclone V/Arria V SoC for ARMCC Used for program linking (memory allocation) in this bare metal sample

10-2-4. registers/soc_a10 directory

Table 4: File structure of registers/soc_a10 directory10

registers/soc_a10 directory	Description
soc_a10_hps_addon_dma330.tcf	Register definition for DMA Controller (DMA-330) for Arria 10 SoC Configuration file for adding display items to register view of DS-5
soc_a10_hps_addon_mpul2_l2c310.tcf	Register definition for L2 Cache Controller (L2C-310) for Arria 10 SoC Configuration file for adding display items to register view of DS-5
soc_a10_hps_addon_mpuscu.tcf	Register definition for Cortex-A9 MPCore Internal Peripherals for Arria 10 SoC This configuration file is used to add items to the register view of the DS-5.

10-2-5. registers/soc_cv_av directory

Table 10: File structure of the registers/soc_cv_av directory

registers/soc_cv_av directory	Description
soc_cv_av_hps_addon_dma330.tcf	Register definition for DMA Controller (DMA-330) for Cyclone V/Arria V SoC Configuration file for adding display items to the DS-5 register view
soc_cv_av_hps_addon_mpul2_l2c310.tcf	Register definition for L2 Cache Controller (L2C-310) for Cyclone V/Arria V SoC Configuration file for adding display items to the DS-5 register view
soc_cv_av_hps_addon_mpuscu.tcf	Register definition for Cortex-A9 MPCore embedded peripherals for Cyclone V/Arria V SoC Configuration file for adding display items to the DS-5 register view

10-2-6. target_board/a10socdk directory

[Table 5] File structure of the target_board/a10socdk directory10

target_board/a10socdk directory	Description
ghrd_10as066n2.sof	Arria 10 SoC development board .sof files
ghrd_10as066n2.sopcinfo	.sopcinfo files for Arria 10 SoC development board
hps_system.h	System header files (Header files generated from .sopcinfo above)
u-boot-spl	preloader files for Arria 10 SoC development board
u-boot-spl.dtb	preloader device tree for Arria 10 SoC development board
u-boot-with-spl.sfp	Flash writing bootloader for Arria 10 SoC development board (4 Preloader and 1 Ubuntu)

10-2-7. target_board/atlas directory

Table 6: File structure of target_board/atlas directory10

target_board/atlas directory	Description
soc_system.sof	.sof file for Atlas SoC development board
soc_system.sopcinfo	.sopcinfo file for Atlas SoC development board
hps_system.h	System Header Files (Header files generated from .sopcinfo above)
u-boot-spl	preloader files for Atlas SoC development boards
u-boot-spl.dtb	preloader device tree for Atlas SoC development boards
u-boot-with-spl.sfp	Flash writing boot loader for Atlas SoC development boards (4 Preloader and 1 Uboot)

10-2-8. target_board/c5socdk directory

Table 10: File structure of target_board/c5socdk directory

target_board/c5socdk directory	Description
soc_system.sof	.sof files for the Cyclone V SoC development board
soc_system.sopcinfo	.sopcinfo files for the Cyclone V SoC development board
hps_system.h	System header files (Header files generated from .sopcinfo above)
u-boot-spl	preloader files for the Cyclone V SoC development board
u-boot-spl.dtb	preloader Device Tree for Cyclone V SoC Development Board
u-boot-with-spl.sfp	Flash Writing Bootloader for Cyclone V SoC Development Board (consists of Preloader x4 and Boot x 1)

10-2-9. target_board/de10nano Directory

[Table 7] File Structure of target_board/de10nano Directory10

target_board/de10nano Directory	Description
soc_system.sof	.sof files for DE10-Nano development board
soc_system.sopcinfo	.sopcinfo files for DE10-Nano development board
hps_system.h	System header files (Header files generated from .sopcinfo above)
u-boot-spl	preloader files for DE10-Nano development board
u-boot-spl.dtb	Device tree for preloader files for DE10-Nano development board
u-boot-with-spl.sfp	Flash writing bootloader for the DE10 Nano development board (4 Preloader and 1 Uboot)

10-2-10. target_board/sodia directory

[Table 8] File structure of the target_board/sodia directory10

target_board/sodia directory	Description
soc_system.sof	Sodia development board .sof file
soc_system.sopcinfo	Sodia development board .sopcinfo file
hps_system.h	System header files (Header files generated from .sopcinfo above)
u-boot-spl	preloader files for Sodia development boards
u-boot-spl.dtb	Device tree for preloader files for Sodia development boards
u-boot-with-spl.sfp	Flash writing boot loader for Sodia development boards (4 Preloader and 1 Uboot)

10-2-11. util directory

[Table 9 File structure of the util directory10

util directory	Description
hwlib	HWLib directory
nios_hal	Nios® II Hardware Abstraction Layer (Since, Nios HAL) directory
cmd.c	Command utility C Source code file You can execute a process by adding it to COMMANDS_LIST commands[].
cmd.h	Command Utility Header File
l2mmu_setting.c	L2MMU Configuration Utility C Source Code File
l2mmu_setting.h	L2MMU Configuration Utility Header File
mem_util.c	Memory Utility C Source Code File
mem_util.h	Memory Utility Header File
usleep_soc.c	usleep utility C source code file
util_interrupt_log.c	Interrupt logging utility C source code file
util_interrupt_log.h	Interrupt logging utility header file
util_time_measurement.c	Timing utility C source code file
util_time_measurement.h	Timing utility header file

Revision History

Revision	years	Overview
1	March 2019	First Edition
3	May 2021	Fixed for version 20.1 (also updated document templates) - Support for tooling environment changes (DS-5 to Arm DS) - Removed MMU sample (sample_mmu.c) because MMU setup has been implemented in a common process for all samples - Removed Helio board support
4	March 2023	Fixed for v22.1 - Improved how Example projects are enabled (specified in config.mk) - New support for ARMCC6 as ARMCC5 is deprecated. - Added instructions for not installing SoC EDS.