

HLS はじめてガイド 簡易チュートリアル

Ver.17.1

HLS はじめてガイド 簡易チュートリアル

目次

1. はじめに	3
2. 評価環境	3
3. インテル® HLS コンパイラ	5
3-1. インテル® HLS コンパイラの概要	5
3-2. インテル® HLS コンパイラ使用時に必要なソフトウェア	6
3-3. 事前準備 (環境変数の設定)	7
4. 一連の操作フロー	9
5. インテル® HLS コンパイラの操作	10
5-1. デザイン (test_tb.cpp、test.cpp) とスクリプト・ファイル (build.bat) の概要	10
5-1-1. test_tb.cpp	11
5-1-2. test.cpp	11
5-1-3. build.bat	12
5-2. インテル® HLS コンパイラの操作	13
5-2-1. コンパイル ・ Emulation	14
5-2-2. RTL Simulation	15
5-2-3. 動作の確認	18
5-2-4. 生成されるフォルダとファイル	23
6. Quartus® Prime の操作	26
6-1. Quartus® Project の概要	26
6-2. Quartus® Prime の操作	27
7. Nios® II SBT の操作	33
7-1. ソフトウェア・プログラム概要 (test.c)	33
7-2. Nios® II SBT の操作	34
改版履歴	40

1. はじめに

この資料は、Quartus® Prime v17.1 にて標準搭載された高位合成ツールのインテル® HLS コンパイラを使用した一連の操作手順について説明しています。C ソース・コードをコンパイル、エミュレーションし、HDL を生成後、Quartus® Prime の Platform Designer システムに取り込み、開発キットにて動作確認まで行っています。

本資料では、HDL 化するコンポーネントのインタフェースとして Avalon-MM Slave インタフェースとし、生成されたコンポーネントは Nios® II プロセッサからレジスタ制御し動作を行っています。

本資料では $a \times b$ のシンプルな乗算器を HDL 化するソース・プログラムを用意しています。

2. 評価環境

下記環境を対象にした一連の操作手順を説明しています。

プラットフォーム:	Windows® 7
Visual Studio:	Microsoft® Visual C++ 2010 Express
Quartus® Prime:	Standard Edition v17.1.1 Build 593
ModelSim® - Intel® FPGA Edition:	10.5b
開発キット:	Cyclone® V E FPGA 開発キット

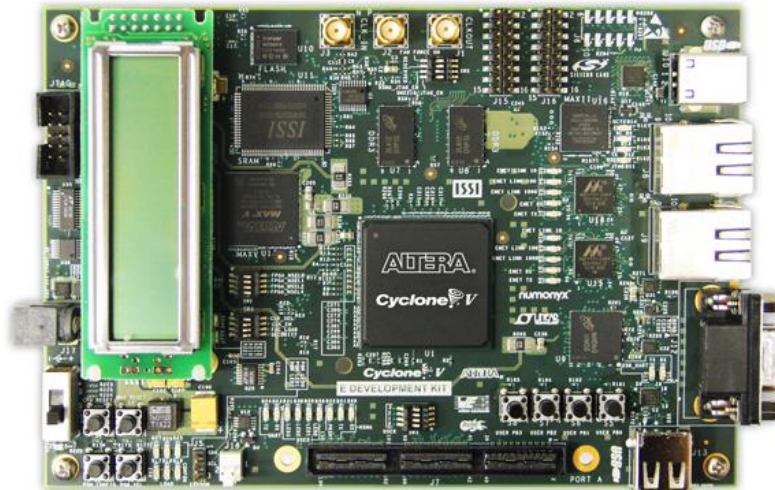


図 2-1 Cyclone® V E FPGA 開発キット

※ 異なる基板にて操作する場合は、Quartus® Prime のプロジェクトにてクロック周波数を変更し、ピン配置を使用するボードにあわせて編集することで対応できます。

ihc_work.zip に一連の操作に必要なファイルを準備しています。ihc_work.zip を解凍すると下記構成(図 2-1-1)になっています。

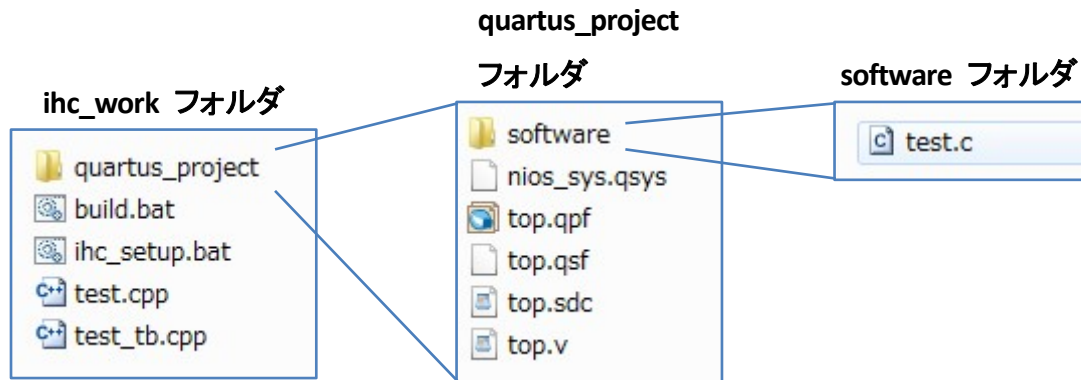


図 2-1-1 フォルダとファイル構成

- ihc_work フォルダ: インテル® HLS コンパイラの操作にて使用します。
- quartus_project フォルダ: Quartus® Prime の操作にて使用します。
- software フォルダ: Nios® II Software Build Tools for Eclipse (Nios® II SBT)の操作にて使用します。

各ファイルについては、操作説明時にあわせて説明します。

3. インテル® HLS コンパイラ

インテル® HLS コンパイラは、Quartus® Prime v17.1 より Pro Edition、Standard Edition、Lite Edition すべての Quartus® Prime Edition に標準で搭載されており、無償で使用することができます。

3-1. インテル® HLS コンパイラの概要

インテル® HLS コンパイラは、ANSI C/C++ のソース・コードを HDL 化することができる高位合成ツールです。

ソフトウェアで実現するよりもハードウェアで実現した方が高速動作可能な C ソース・コード内の機能を HDL 化することができます。HDL 化する際のインタフェースは実装するハードウェアにあわせて選択可能です。

- 選択可能なインタフェース

デフォルト・インタフェース

Avalon-ST

Avalon-MM

Avalon インタフェースを選択することで Platform Designer のコンポーネントの 1 つとして Platform Designer システム内に実装することができます。

本資料では、Avalon-MM Slave インタフェースを使用した手順を説明しています。

インテル® HLS コンパイラにてサポートされている機能は下記です。

- Emulation (build.bat 内では test-x86-64 で定義)

PC 上で関数の機能を検証します。コンパイル後に生成された実行ファイルを起動します。

- Generate (build.bat 内では test-fpga で定義)

component 指定した関数の HDL を生成します。

- Verification (build.bat 内では test-fpga-sim で定義)

component 指定した関数を HDL 化、main 関数からテストベンチを生成し、RTL Simulation を実行します。ModelSim® にて実行されたシミュレーション結果を .wlf ファイルとして生成します。

コンパイル後に生成された実行ファイルを起動します。

3-2. インテル® HLS コンパイラ使用時に必要なソフトウェア

インテル® HLS コンパイラを使用する際には、下記ソフトウェアが別途必要になります。

【Windows® の場合】

- Microsoft® Visual Studio 2010 Professional

※ 最新版の Microsoft® Visual Studio はサポートしていません。(2018 年 3 月現在)

- Quartus® Prime 17.1 以上

- 下記いずれかの HDL シミュレータ

- ModelSim® - Intel® FPGA Edition
- ModelSim® - Intel® FPGA Starter Edition
- その他 Mentor Graphics® 社 の ModelSim®

ModelSim® のサポート・バージョンについては、Quartus® Prime のリリース・ノート内「EDA Interface Information」を合わせてご参照ください。

[Intel Quartus Prime Pro Edition Software and Device Support Release Notes Version](#)

[Intel Quartus Prime Standard Edition Software and Device Support Release Notes](#)

【Linux の場合】

- GCC コンパイラ、C++ ライブラリ v4.7.7

※ 最新版の GCC コンパイラおよび C++ ライブラリはサポートしていません。

- 下記いずれかの HDL シミュレータ

- ModelSim® - Intel® FPGA Edition
- ModelSim® - Intel® FPGA Starter Edition
- その他 Mentor Graphics® 社 の ModelSim®

ModelSim® のサポート・バージョンについては、Quartus® Prime のリリース・ノート内「EDA Interface Information」を合わせてご参照ください。

[Intel Quartus Prime Pro Edition Software and Device Support Release Notes Version](#)

[Intel Quartus Prime Standard Edition Software and Device Support Release Notes](#)

インテル® HLS コンパイラのコマンド・ラインは g++ と互換性があり、下記拡張子のファイルは c++ ファイルと同様に扱います。

.c、.C、.cc、.cpp、.CPP、.c++、.cp、.cxx

3-3. 事前準備(環境変数の設定)

インテル® HLS コンパイラ使用時にいくつかの環境変数の設定が必要です。

IHCROOT: インテル® HLS コンパイラ
VC_INSTALL: Microsoft® Visual Studio
など

ihc_work¥ihc_setup.bat を使用し、必要な環境変数の設定が可能です。

ihc_setup.bat 内のパスを環境に合わせて編集し、実行してください。環境にあわせて編集した ihc_setup.bat を実行することで環境変数が定義されたコマンド・プロンプトが起動します。

※ ihc_setup.bat を実行したコマンド・プロンプト内でのみ環境変数が有効になります。

コマンド・プロンプトを閉じた場合は、再度 ihc_setup.bat を実行する必要があります。

(手順 1) ihc_setup.bat を右クリックし、“編集” を選択します。

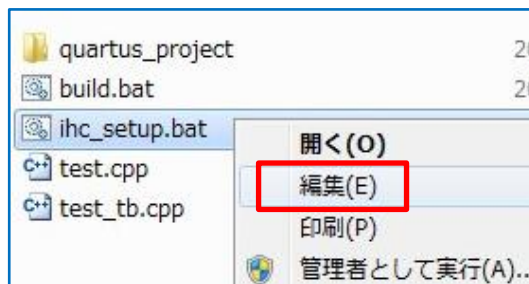


図 3-3-1 ihc_setup.bat の編集

(手順 2) 下記 3 行を環境にあわせて編集します。

6 行目: IHCROOT: Quartus® Prime インストール・フォルダ内の hls フォルダまでのパス

図 3-3-2 例: `set IHCROOT=D:¥tools¥Intel_FPGA¥v171_Std¥hls`

8 行目: VC_INSTALL: Microsoft® Visual Studio のフォルダ

デフォルトのインストール・フォルダです。

図 3-3-2 例: `set VC_INSTALL=C:¥Program Files (x86)¥Microsoft Visual Studio 10.0`

10 行目: LM_LICENSE_FILE: Quartus® Prime と ModelSim® - Intel® FPGA Edition のライセンス

図 3-2-2 例: `set LM_LICENSE_FILE=D:¥flexlm¥altera.dat;D:¥flexlm¥mentor.dat;%LM_LICENSE_FILE%
%LM_LICENSE_FILE% により、その他の設定に対して追加定義しています。`

17 行目: PATH: 使用予定の ModelSim® の実行ファイルの存在するフォルダまでのパス

図 3-3-2 例:

```
set PATH=%VC_INSTALL%\VC\bin\amd64;D:\tools\Intel_FPGA\v171_Std\modelsim_ae\win32aloem;%PATH%
```

%PATH% により、その他の PATH 設定に対して追加定義しています。

※ 複数の ModelSim® の Edition やバージョンをインストールされている場合は、使用する ModelSim の実行ファイルの存在するフォルダまでパスを設定する必要があります。

```
@echo off
::Please Set Following Param
::#####
::Set IHC root path
set IHCROOT=D:\tools\Intel_FPGA\v171_Std\hls
::Set VC install path
set VC_INSTALL=C:\Program Files (x86)\Microsoft Visual Studio 10.0
::Set Quartus and ModelSim path
set LM_LICENSE_FILE=D:\flexlm\altera.dat;D:\flexlm\mentor.dat;%LM_LICENSE_FILE%
::#####
::Do not edit from here!!
::#####
::Set Tools Env
set VS100COMNTOOLS=%VC_INSTALL%\Common7\Tools
set PATH=%VC_INSTALL%\VC\bin\amd64;D:\tools\Intel_FPGA\v171_Std\modelsim_ae\win32aloem;%PATH%
set INCLUDE=C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1\Include;%VC_INSTALL%\VC\include;%INCLUDE%
set LIB=%VC_INSTALL%\VC\lib\amd64;C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1\Lib\x64;%LIB%
set LIBPATH=%VC_INSTALL%\VC\lib\amd64;C:\Program Files (x86)\Microsoft SDKs\Windows\v7.1\Lib\x64;%LIBPATH%

call %IHCROOT%\init_hls.bat

cmd
```

図 3-3-2 ihc_setup.bat 編集例

(手順 3) 上書き保存し、ihc_setup.bat を閉じます。

4. 一連の操作フロー

まずは、test_tb.cpp と test.cpp のソース・ファイルを使用し、インテル® HLS コンパイラにて HDL を生成します。続いて生成された HDL を Quartus® Prime 内で Platform Designer システムに取り込み、コンパイルを実行します。Quartus® Prime にて FPGA 用の書き込みファイル生成後、Nios® II SBT にて test.c を使用しソフトウェアの Build を実行後、実機にて Nios® II を動作させ、HDL 化したコンポーネントの動作を確認します。

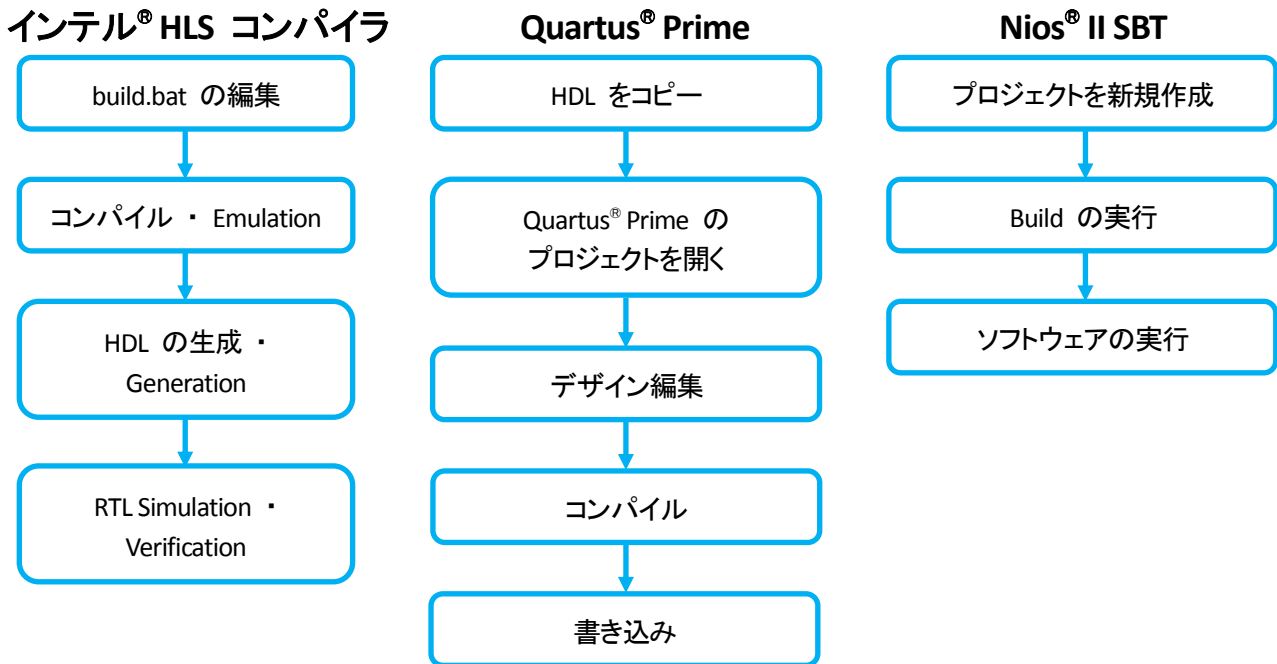


図 4-1 一連の操作フロー

詳細については、各章にて説明します。

5. インテル® HLS コンパイラの操作

インテル® HLS コンパイラの操作に関して説明します。

5-1. デザイン(test_tb.cpp、test.cpp)とスクリプト・ファイル(build.bat)の概要

本資料では、容易に動作が確認できるように .cpp のソース・ファイルでは、 $a \times b$ の簡単な動作を使用しています。

2 つの .cpp ファイル(test_tb.cpp、test.cpp)を使用しています。本資料では 2 つの .cpp ファイルを使用していますが、1 つの .cpp ファイルにまとめることもできます。

- test_tb.cpp

main 関数

シミュレーション時にはテストベンチに変換されます。

- test.cpp

HDL 化対象の関数

$a \times b$ の演算を実施

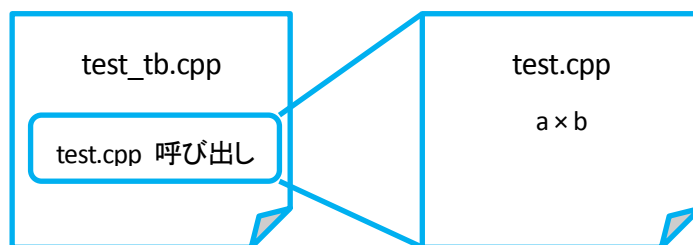


図 5-1-1 ソース・ファイルのイメージ

5-1-1. test_tb.cpp

main 関数のソース・ファイルです。HDL 化予定の test 関数を呼び出しています。動作としては、test 関数の 2 つの引数にそれぞれ 0 ~ 9 の値を代入し、test 関数による戻り値を表示するものです。test_tb.cpp 内では HDL 化対象となる test 関数に対して component のラベルを付加しています。

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  component int test (int a, int b);
5
6  int main ()
7  {
8      int i = 0;
9      int j = 0;
10
11     for (i = 0; i < 10; i++)
12         for (j = 0; j < 10; j++)
13         {
14             printf("%d * %d = %d\n", i, j, test(i,j));
15         }
16
17     return 0;
18 }
19
    
```

戻り値 integer

引数 a、b とともに integer の test 関数が HDL 化対象のため、component のラベルを付加

図 5-1-1-1 test_tb.cpp

5-1-2. test.cpp

HDL 対象のソース・ファイルです。a×b の演算を行うプログラムです。

3 行目: #include "HLS/hls.h"

インテル® HLS コンパイラにラベルを認識させるため、インクルード宣言

5 行目: hls_avalon_slave_component

対象の関数は Avalon-MM Slave のコンポーネントであることを宣言

6 行目: component int test

戻り値が integer の test という関数が HDL 化対象のコンポーネントであることを定義

7-8 行目: hls_avalon_slave_register_argument int a,

hls_avalon_slave_register_argument int b

引数 a、b 共に Avalon-MM Slave のレジスタとして定義

12 行目: a×b の値を返す

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "HLS/hls.h"
4
5  hls_avalon_slave_component
6  component int test (
7      hls_avalon_slave_register_argument int a,
8      hls_avalon_slave_register_argument int b
9  )
10 {
11     {
12         return a * b;
13     }
14 }
    
```

図 5-1-2-1 test.cpp

5-1-3. build.bat

インテル® HLS コンパイラにて必要なコマンドをスクリプト化したファイルです。

- 1 行目: @echo offset "SOURCE_FILES=test.cpp test_tb.cpp"set "HLS_CXX_FLAG="
 対象のソース・ファイルを test.cpp、test_tb.cpp として定義しています。
 main 関数の記述されたソース・ファイル(本資料では test_tb.cpp)を最後に記述します。

2 ~ 11 行目: コメントにて対象となる機能(TARGET)について説明しています。

- test-x86-64: Emulation
- test-fpga-sim: Verification
- test-fpga: Generate

12 行目以降: 変数の設定や実行されるコマンドなどが記述されています。

```
@echo offset "SOURCE_FILES=test.cpp test_tb.cpp"set "HLS_CXX_FLAGS=": This batch file will compile th
:: 1) test-msvc Compile the example design to the CPU
:: Uses Visual Studio 2010
:: 2) test-x86-64 Compile the example design to the CPU
:: Uses the Intel HLS Compiler
:: 3) test-fpga-sim Synthesize the example design to HDL
:: Generates wlf which is a result of the HDL simulation
:: Uses the Intel HLS Compiler
:: 4) test-fpga Synthesize the example design to HDL
:: Generates a cosimulation executable to simulate the HDL
:: Uses the Intel HLS Compiler
:: 5) clean Remove any temporary files generated by the compiler
:: Usage: build.bat <target>
:: Example: build.bat test-x86-64

:: Only one argument expected
if not "%2"==" " goto usage

:: Accept the user's target, else default to x86-64
if not "%1"==" " (
  set "TARGET=%1"
) else (
  set "TARGET=test-x86-64"
echo No target specified, defaulting to %TARGET%
echo Available targets: test-x86-64, test-fpga-sim, test-fpga, test-msvc, clean
)

:: Any tools installed with HLS can be found relative to the location of i++
for %%I in (i++.exe) do (
  set "HLS_INSTALL_DIR=%%~dp$PATH:1"
)
set "HLS_INSTALL_DIR=%HLS_INSTALL_DIR%.."

:: Set up the compile variables
if "%TARGET%" == "test-x86-64" (
  set "CXX=i++"
  set "CXXFLAGS=%HLS_CXX_FLAGS% -march=x86-64"
  set "LFLAGS=-o %TARGET%.exe"
) else if "%TARGET%" == "test-fpga-sim" (
  set "CXX=i++"
  set "CXXFLAGS=%HLS_CXX_FLAGS% -march=Arria10 --simulator modelsim -ghdl"
  set "LFLAGS=-o %TARGET%.exe"
) else if "%TARGET%" == "test-fpga" (
  set "CXX=i++"
  set "CXXFLAGS=%HLS_CXX_FLAGS% -march=Arria10"
  set "LFLAGS=-o %TARGET%.exe"
) else if "%TARGET%" == "test-msvc" (
  set "CXX=cl"
  set "CXXFLAGS=/I ""%HLS_INSTALL_DIR%\include"" /nologo /EHsc /wd4068 /DWIN32 /MD"
  set "LFLAGS=/link ""%HLS_INSTALL_DIR%\host\windows64\lib"" hls_emul.lib /out:%TARGET%.exe"
) else if "%TARGET%" == "clean" (
  del /S /F /Q test-msvc.exe test-fpga.exe test-fpga.prj test-x86-64.exe > NUL
  rmdir /S /Q test-fpga.prj > NUL
  goto:eof
) else (
  goto usage
)

:: Replace "" with " in the flags
set "CXXFLAGS=%CXXFLAGS:"=""
```

図 5-1-3-1 build.bat の一部

コマンド・プロンプトにおける入力方法は下記です。

build TARGET

TARGET には、実現したい機能を指定します。

<TARGET>

test-x86-64: Emulation

test-fpga: Generation

test-fpga-sim: Verification

```
E:\ihc_work>build test-fpga-sim
i++ -march=Arria10 --simulator modelsim -ghdl test.cpp test_tb.cpp -o test-fpga-sim.exe
```

図 5-1-3-2 コマンド・プロンプトにおける記述例

5-2. インテル® HLS コンパイラの操作

インテル® HLS コンパイラの基本フローは下記です。

インテル® HLS コンパイラ

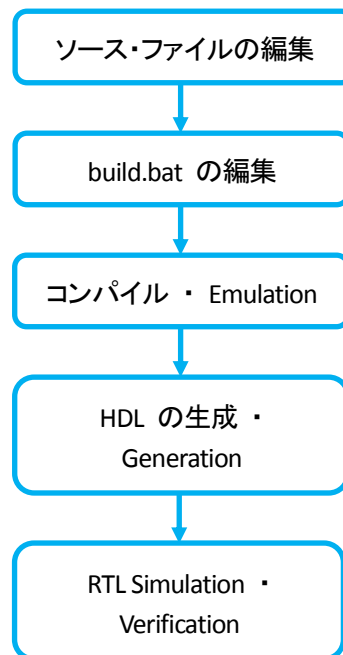


図 5-2-1 インテル® HLS コンパイラのフロー・イメージ図

本資料では、test_tb.cpp、test.cpp、build.bat にて @echo offset "SOURCE_FILES=test.cpp test_tb.cpp " で、すでに編集済みのため、下記工程のみを実施します。

- コンパイル ・ Emulation
- Verification

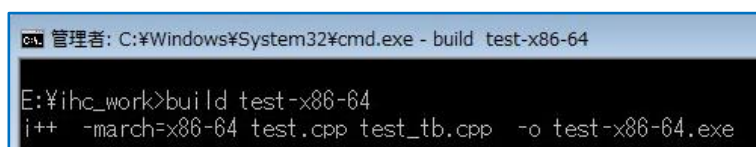
5-2-1. コンパイル ・ Emulation

(手順 1) 第 3-3 章『事前準備 (環境変数の設定)』にて設定済みの ihc_setup.bat をダブルクリック、もしくは右クリック ▶ 管理者として実行を選択します。

必要な環境変数を設定し、コマンド・プロンプトが起動します。

(手順 2) 下記コマンドを入力し、Enter キーを押します。

```
buid test-x86-64
```



```

管理者: C:\Windows\System32\cmd.exe - build test-x86-64
E:\ihc_work>buid test-x86-64
i++ -march=x86-64 test.cpp test_tb.cpp -o test-x86-64.exe

```

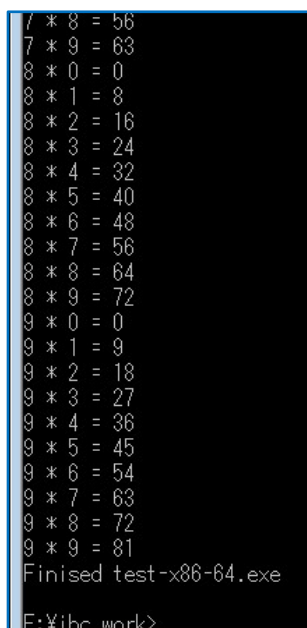
図 5-2-1-1 build test-x86-64 の実行

下記コマンドが実行されます。

```
i++ -march=x86-64 test.cpp test_tb.cpp -o test-x86-64.exe
```

test_tb.cpp と test.cpp をコンパイルし、test-x86-64.exe を生成します。

生成された test-x86-64 を実行し、0x0 ~ 9x9 が実行され、結果を表示します。



```

7 * 8 = 56
7 * 9 = 63
8 * 0 = 0
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
9 * 0 = 0
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
Finised test-x86-64.exe
E:\ihc_work>

```

図 5-2-1-2 test-x86-64 の実行結果の表示

生成された .exe の実行結果より、動作に問題がないことを確認します。

5-2-2. RTL Simulation

第 5-2-1 『コンパイル・Emulation』にて動作に問題がなかったため、test.cpp を HDL 化し、main 関数からテストベンチを生成し、RTL Simulation の実行まで行います。build.bat では、RTL Simulation 後に vsim コマンドを実行し、ModelSim® を起動するようになっています。

(手順 1) 下記コマンドを入力し、Enter キーを押します。

```
build test-fpga-sim
```



```
9 * 9 = 81
Finised test-x86-64.exe
E:¥ihc_work>build test-fpga-sim
```

図 5-2-2-1 build test-fpga-sim の実行

下記コマンドが実行されます。

```
i++ -march=Arria10 --simulator modelsim -ghdl test.cpp test_tb.cpp -o test-fpga-sim.exe
```

--simulator modelsim: シミュレータを ModelSim® に指定

--ghdl: RTL Simulation の結果波形を wlf ファイルをして生成



```
E:¥ihc_work>build test-fpga-sim
i++ -march=Arria10 --simulator modelsim -ghdl test.cpp test_tb.cpp -o test-fpga-sim.exe
```

図 5-2-2-2 build test-fpga-sim の実行

Arria® 10 を対象にした test-fpga-sim.exe が生成されます。

生成された test-fpga-sim.exe を実行すると test.cpp を HDL、test_tb.cpp をテストベンチ化し、ModelSim® にて RTL Simulation が実行され、結果をコマンド・プロンプトに表示します。

```
Run test-fpga-sim.exe to execute the test.
0 * 0 = 0
0 * 1 = 0
0 * 2 = 0
0 * 3 = 0
0 * 4 = 0
0 * 5 = 0
```

図 5-2-2-3 test-fpga-sim.exe の実行結果の表示

RTL Simulation 実行後に vsim コマンドを実行するため、ModelSim® が起動します。

```
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
Finised test-fpga-sim.exe
Launch Modelsim
Please open the wlf on ModelSim
Reading D:/tools/Intel_FPGA/v171_Std/modelsim_ae/tcl/vsim/pref.tcl
```

図 5-2-2-4 ModelSim® - Intel FPGA Edition の起動コマンド

(手順 2) 起動した ModelSim® にて生成された .wlf ファイルを開きます。

File ▶ Open を選択します。

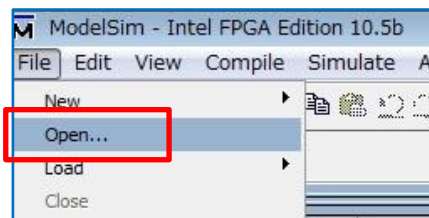


図 5-2-2-5 ModelSim File メニュー

(手順 3) 下記 vsim.wlf を選択し、“Open” ボタンを押します。

¥ihc_work¥test-fpga-sim.prj¥verification¥vsim.wlf

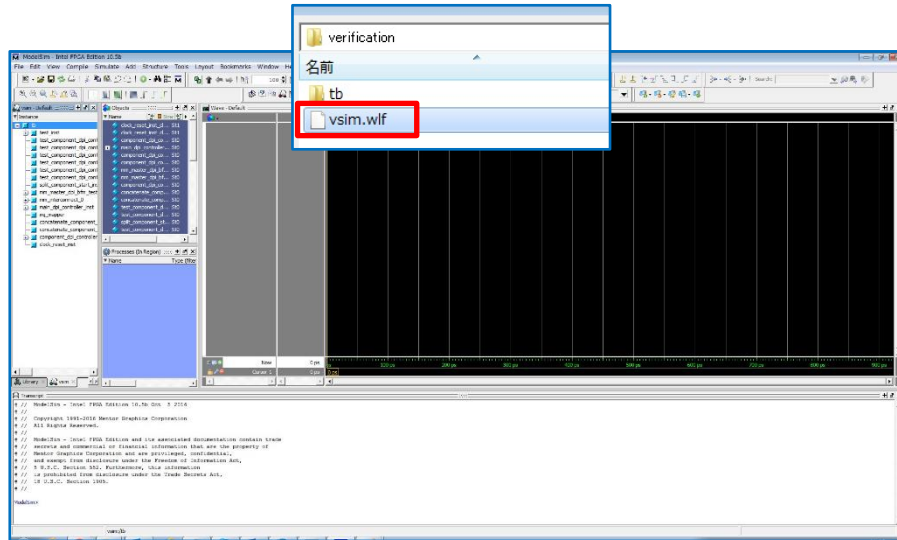


図 5-2-2-6 Open File ウィンドウ

(手順 4) test_inst を選択し、右クリック ▶ Add Wave を選択します。

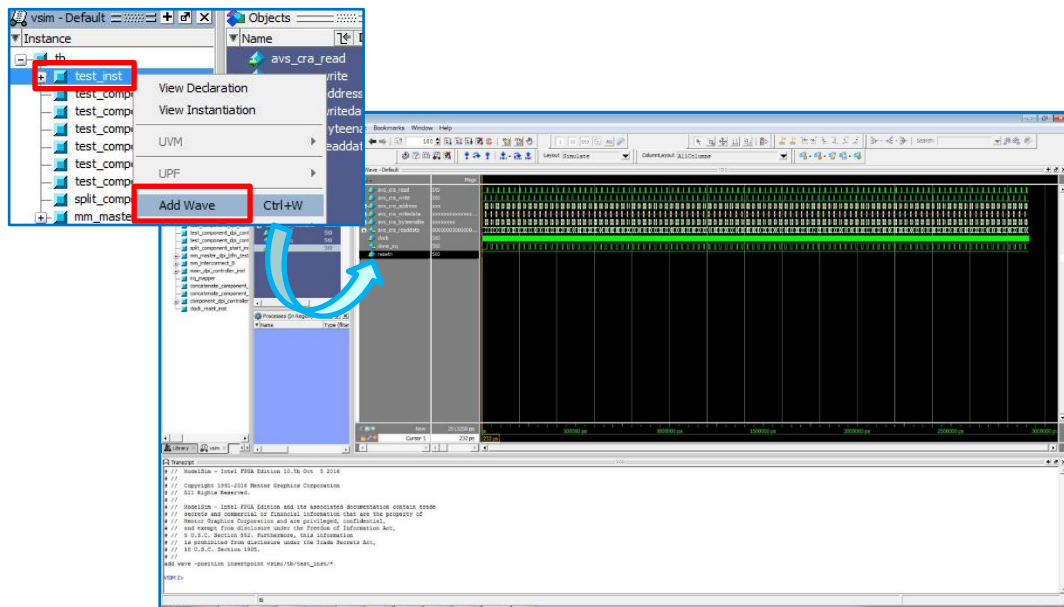


図 5-2-2-7 結果波形の表示

5-2-3. 動作の確認

ハードウェアの動作を確認することでレジスタの制御手順を理解することができます。レジスタは下記生成ファイルにて確認できます。

```
¥ihc_work¥test-fpga-sim.prj¥components¥test¥test_csr.h
```

.csr.h 内の Register Address は byte 単位のアドレスになっています。各レジスタが 64bit のため 8 byte address にすると Register Address は 0x0、0x8、0x10・・・0x30 のように 8 ずつインクリメントしています。

64bit 単位の Address に変更すると 0x0、0x1、0x2、0x3、0x4、0x5、0x6 のように 1 ずつインクリメントします。この、64bit 単位の Address を使用してレジスタにアクセスします。

```

/* This header file describes the CSR slave for the test component */
#ifndef __TEST_CSR_REGS_H__
#define __TEST_CSR_REGS_H__

/* Memory Map Summary */
/* Register Address, Access, Register Contents (64-bits), Description */
/* Address 0x0: Read the busy status of the component */
/* Address 0x1: Write 1 to signal start to the component */
/* Address 0x2: 0 - Disable interrupt, 1 - Enable interrupt */
/* Address 0x3: Signals component completion done is read-only and interrupt_status is write 1 to clear */
/* Address 0x4: Return data */
/* Address 0x5: Argument a */
/* Address 0x6: Argument b */
NOTE: Writes to reserved bits will be ignored and reads from reserved bits will return undefined values.
/* Register Address Macros */
/* Byte Addresses */
#define TEST_CSR_BUSY_REG (0x0)
    
```

図 5-2-3-1 test_csr.h

RTL Simulation の結果と上記 test_csr.h を使用し、動作を確認していきます。バスの Radix は Unsigned にしています。

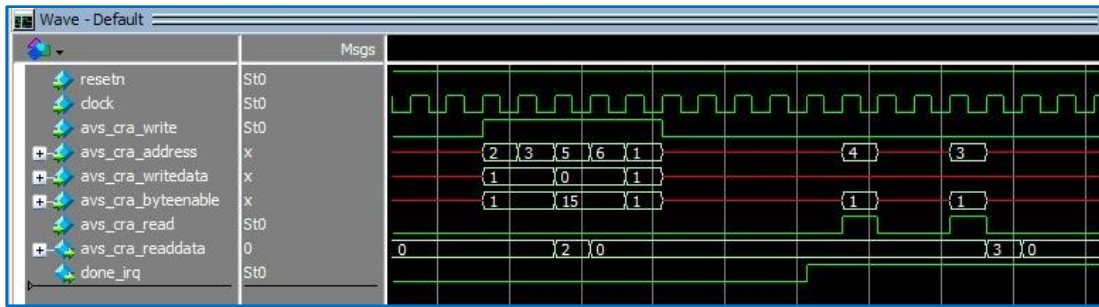


図 5-2-3-2 RTL Simulation 結果 1

図 5-2-3-2 RTL Simulation 結果 1 では、下記順番で Write を実施しています。

表 5-2-3-1 Write シーケンス

64bit 単位 Address		writedata	byteenable	概要
0x2	Interrupt	0x1	0x1	Interrupt を Enable
0x3	Interrupt Status を Clear	0x1	0x1	Interrupt を Clear
0x5	Argument a	0x0	0x15	Argument a に 32bit の 0 を Write
0x6	Argument b	0x0	0x15	Argument b に 32bit の 0 を Write
0x1	Start	0x1	0x1	Start

図 5-2-3-2 RTL Simulation 結果 1 では、下記順番で read を実施しています。

表 5-2-3-2 Read シーケンス

64bit 単位 Address		byteenable	概要
0x4	Return Data	0x1	Return data の Read
			readdata に結果が表示
0x3	done	0x1	Done

図 5-2-3-2 RTL Simulation 結果 1 の流れは、下記の通りです。

1. Interrupt を Enable (0x2 に 0x1)
 2. Interrupt Status を Clear (0x3 に 0x1)
 3. Argument a に Write (0x5 に 0)
 4. Argument b に Write (0x6 に 0)
 5. Start を Write (0x1 に 0x1)
 6. Return Data (0x4)
- readdata に結果 (0) を表示
7. done (0x3) を Read

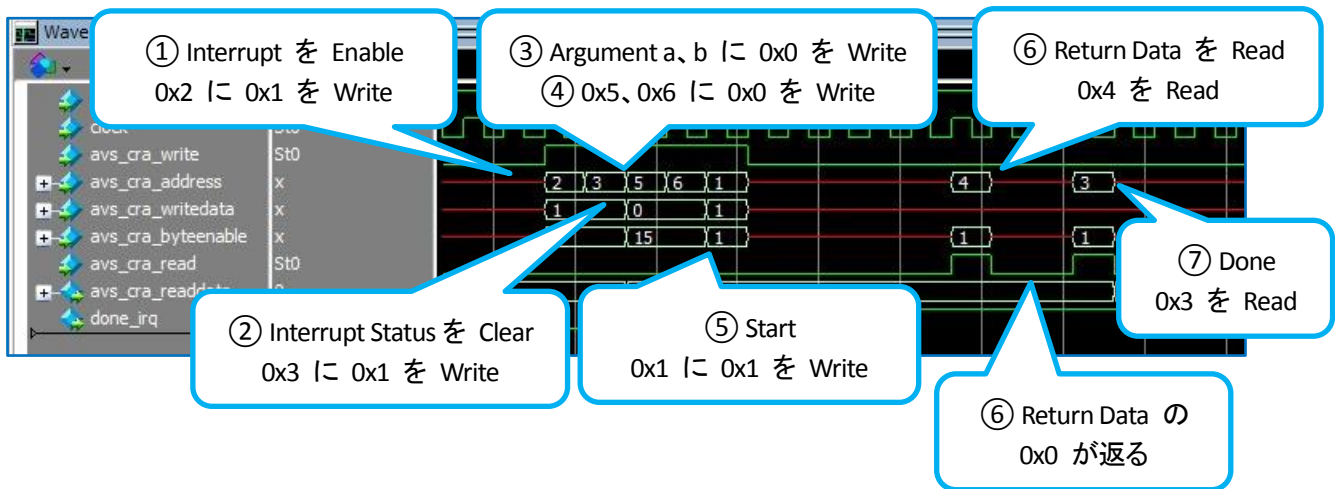


図 5-2-3-3 RTL Simulation 結果 1

上記レジスタの制御にて Argument a, b にデータを Write し、Return Data より結果が返る手順になります。

続いて結果を確認していきます。a × b を HDL 化し、実行した結果を main 関数に戻す結果では、0x0、0x1、0x2 … が期待通りの動作になっていることを確認できます。

```

Run test-fpga-sim.exe to execute the test.
0 * 0 = 0
0 * 1 = 0
0 * 2 = 0
0 * 3 = 0
0 * 4 = 0
0 * 5 = 0
0 * 6 = 0
0 * 7 = 0
0 * 8 = 0
    
```

図 5-2-3-4 HDL 化後の実行結果

しかし、ModelSim® - Intel® FPGA Edition にて波形結果を確認すると Argument a (0x5) の Writedata が 152316720185344 などの値になっています。

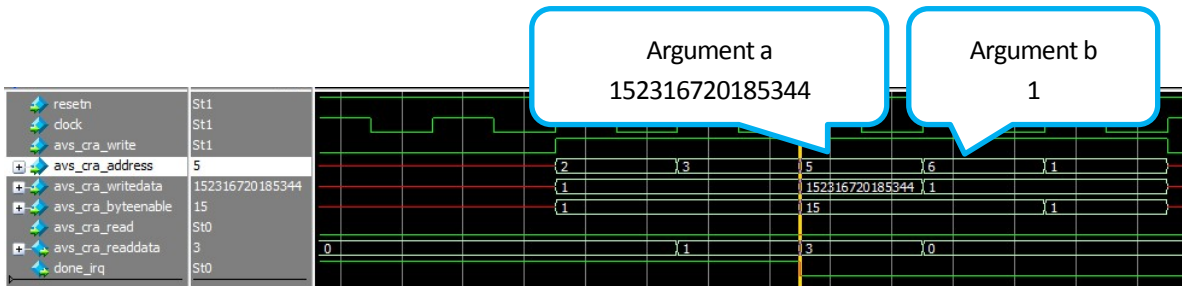


図 5-2-3-5 a × b = 0 × 1 の波形結果

Memory Map にある通り、Argument a や Argument b は 32bit 幅です。上位 bit は無視できます。下位 32bit が実際に使用されるデータです。

```

/*****
 * Memory Map Summary
 *****/

```

Register Address	Access	Register Contents (64-bits)	Description
0x0	R	{reserved[62:0], busy[0:0]}	Read the busy status of the component 0 - the component is ready to accept a new start 1 - the component cannot accept a new start
0x8	W	{reserved[62:0], start[0:0]}	write 1 to signal start to the component
0x10	R/W	{reserved[62:0], interrupt_enable[0:0]}	0 - Disable interrupt, 1 - Enable interrupt
0x18	R/Wc1r	{reserved[61:0], done[0:0], interrupt_status[0:0]}	signals component completion done is read-only and interrupt_status is write 1 to clear
0x20	R	{reserved[31:0], returndata[31:0]}	Return data
0x28	R/W	{reserved[31:0], a[31:0]}	Argument a
0x30	R/W	{reserved[31:0], b[31:0]}	Argument b

NOTE: writes to reserved bits will be ignored and reads from reserved bits will return undefined values.

図 5-2-3-6 test_csr.h

改めて writedata を見てみると下位 32bit は "00000000" になっているため、期待どおりの動作になっています。

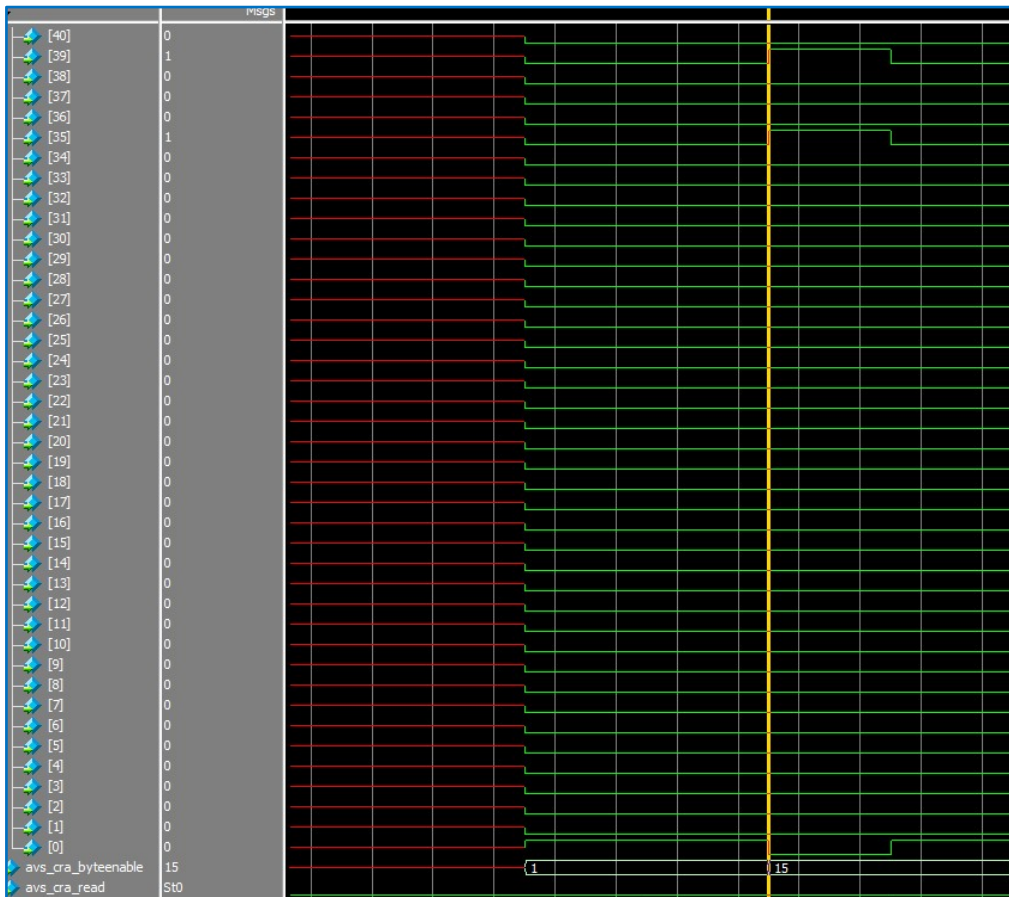


図 5-2-3-7 writedata 下位 32bit 拡大表示

0x0、0x1 … 9x8、9x9 が期待通りの結果になっていることを確認できます。

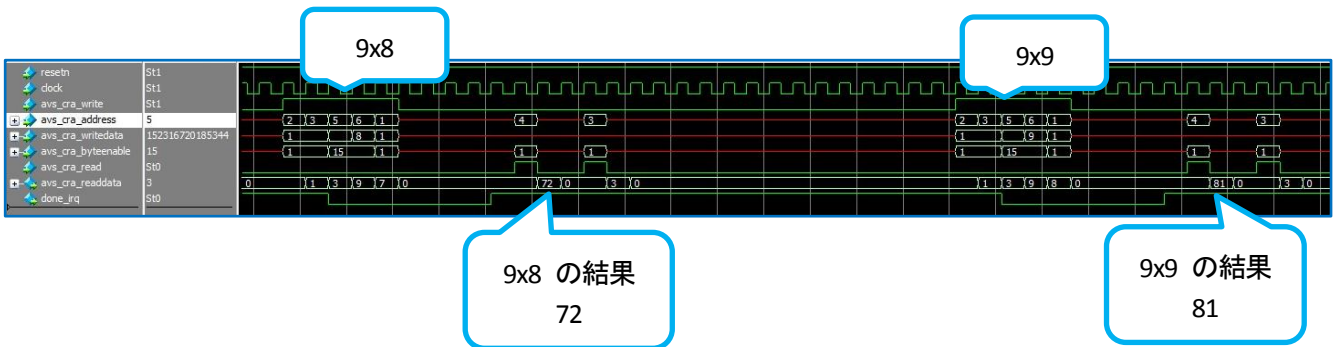


図 5-2-3-8 RTL Simulation 結果波形

RTL Simulation により、HDL に問題ないことが確認できました。

5-2-4. 生成されるフォルダとファイル

test-fpga や test-fpga-sim を実行すると対象のコンポーネントが HDL 化されます。生成されるフォルダ構成は下記です。

test-fpga-sim.prj

- L components: 生成された HDL ファイル
 - L test: Platform Designer システムに読み込む際に必要なファイル
- L quartus: コンパイルされた Quartus® Project
- L reports: 各種レポート
- L verifications: テストベンチとスクリプト・ファイルを保存

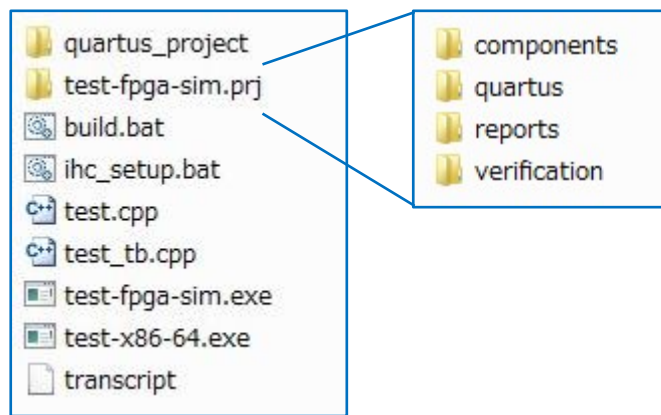


図 5-2-4-1 ihc_work フォルダ

- components フォルダ

生成された HDL ファイルを含みます。

Platform Designer システムに取り込む際に必要なファイル一式が格納されています。

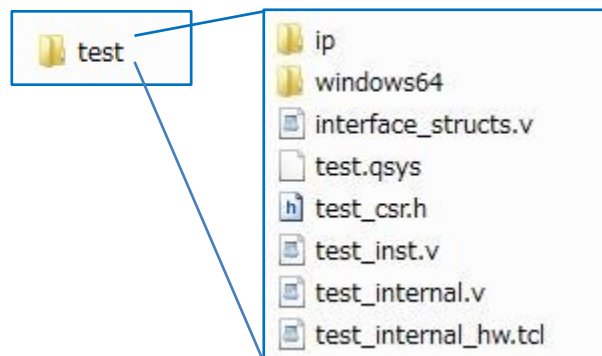


図 5-2-4-2 components フォルダ

- quartus フォルダ

test.cpp を最上位階層とした HDL を使用し Quartus® Prime にて一度コンパイルを実施しています。その際に使用した Quartus® Prime のプロジェクトに関するファイル一式が格納されています。

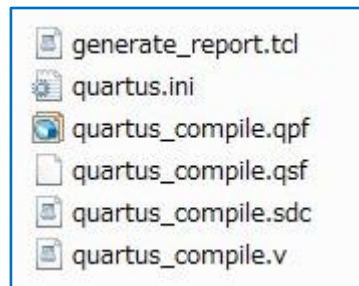


図 5-2-4-3 quartus フォルダ

- reports フォルダ

Quartus® Prime にてコンパイルした際のレポートが格納されています。

report.html では、

- Summary

指定したデバイスやコマンド、Fmax やリソース見積もり結果

- Loops analysis

ループのパイプライン化、ボトルネック

- Area analysis of system

システムに必要な機能ごとのリソース見積もり数

- Area analysis of source

各ブロックのリソース見積もり数

- Component viewer

生成されたシステムのグラフィカル表示

- Component memory viewer

コンポーネント内のメモリ情報

- Verification statistics

レイテンシ

が確認できます。

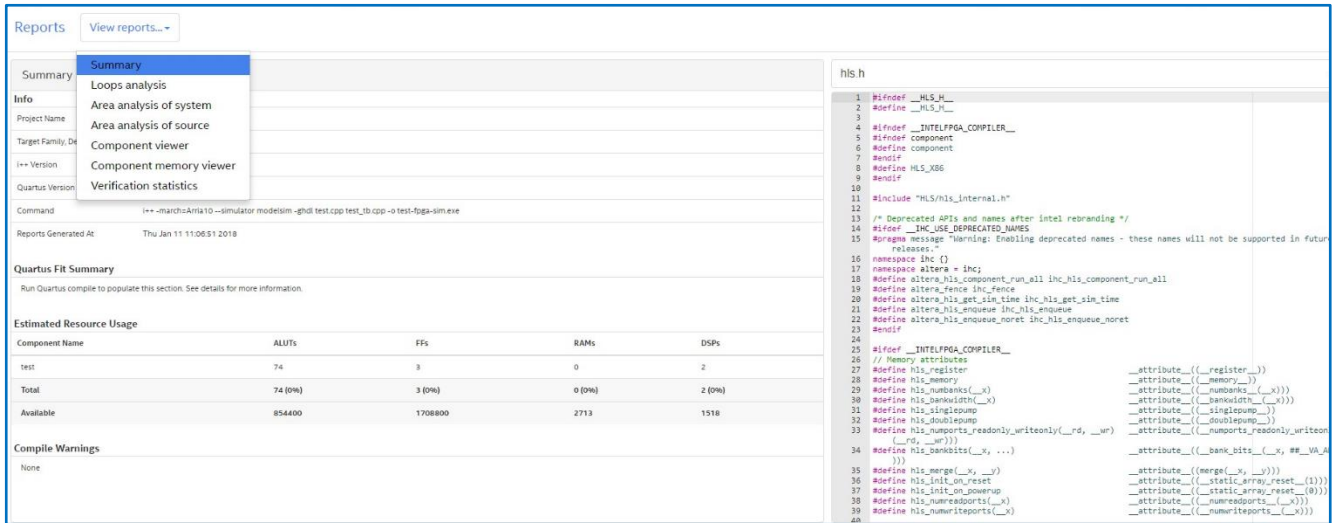


図 5-2-4-4 report

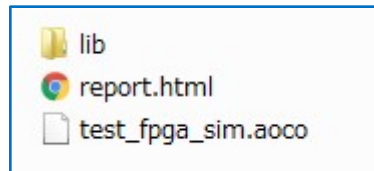


図 5-2-4-5 reports フォルダ

- verification フォルダ

ModelSim® で使用されたファイルやシミュレーション結果波形(.wlf)が格納されています。

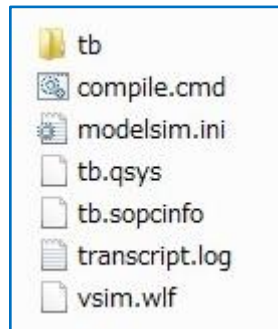


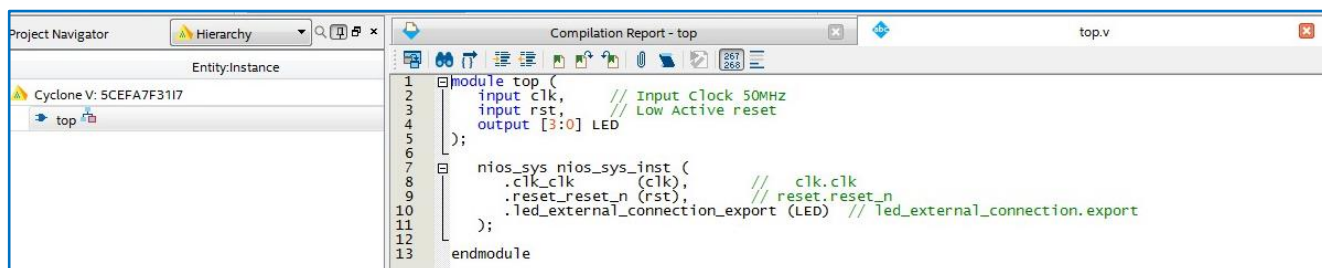
図 5-2-4-6 verification フォルダ

6. Quartus® Prime の操作

Quartus® Prime の操作に関して説明します。

6-1. Quartus® Project の概要

本資料では、Cyclone® V E FPGA 開発キットを対象にしたピン配置などの設定を行った Quartus® Prime のプロジェクト・ファイル(top.qpf)をすでに用意しています。



```

1 module top (
2     input clk,           // Input clock 50MHz
3     input rst,          // Low Active reset
4     output [3:0] LED
5 );
6
7 nios_sys nios_sys_inst (
8     .clk_clk (clk),
9     .reset_reset_n (rst), // reset, reset_n
10    .led_external_connection_export (LED) // led_external_connection.export
11 );
12
13 endmodule

```

図 6-1-1 top.qpf

また、Platform Designer システムでは、すでに HDL 化した test component を接続したシステムを用意しています。実装されているコンポーネントは下記です。

- Clock Source

Platform Designer システムで使用する 50MHz のクロックを定義しています。

- Nios® II Processor

- On-Chip Memory

- JTAG UART

ソース・ファイル内で printf 実行時に Nios® II SBT のコンソール上に表示させるために使用しています。

- PIO

Cyclone® V E FPGA 開発キット上の LED を動作させるために使用しています。

- test

インテル® HLS コンパイラにて HDL 化されたコンポーネントです。

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
✓		clk_50mhz	Clock Source	clk	exported				
		clk_in	Clock Input	reset	clk_50mhz				
		clk_in_reset	Reset Input	Double-click to export					
		clk	Clock Output	Double-click to export					
		clk_reset	Reset Output	Double-click to export					
✓		nios2	Nios II Processor						
		clk	Clock Input	Double-click to export	clk_50mhz				
		reset	Reset Input	Double-click to export	[clk]				
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		irq	Interrupt Receiver	Double-click to export	[clk]			IRQ 0	IRQ 31
		debug_reset_request	Reset Output	Double-click to export	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x8800	0x8fff		
		custom_instruction_m...	Custom Instruction Master	Double-click to export					
		ocram	On-Chip Memory (RAM or ROM)						
		clk1	Clock Input	Double-click to export	clk_50mhz				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	# 0x4000	0x7fff		
		reset1	Reset Input	Double-click to export	[clk1]				
✓		jtag_uart	JTAG UART						
		clk	Clock Input	Double-click to export	clk_50mhz				
		reset	Reset Input	Double-click to export	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x9040	0x9047		
		external_slave	External Slave	Double-click to export	[clk]				
✓		test_hls	test						
		avs_cra	Avalon Memory Mapped Slave	Double-click to export	[clock]	# 0x9000	0x903f		
		clock	Clock Input	Double-click to export	clk_50mhz				
		irq	Interrupt Sender	Double-click to export	[clock]				
		reset	Reset Input	Double-click to export	[clock]				
✓		led	PIO (Parallel I/O)						
		clk	Clock Input	Double-click to export	clk_50mhz				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0000	0x000f		
		external_connection	Conduit	led_external_connection					

図 6-1-2 nios_sys.qsys

6-2. Quartus® Prime の操作

Quartus® Prime の基本フローは下記です。

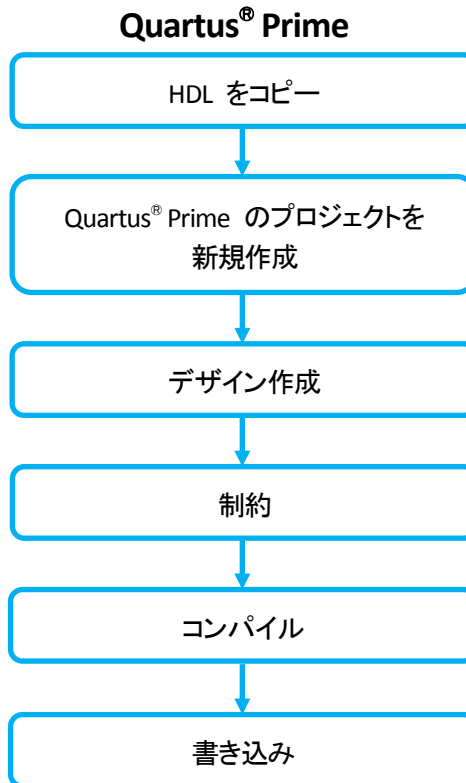


図 6-2-1 Quartus® Prime 操作フロー・イメージ

本資料では、Quartus® Prime のプロジェクトを用意済みのため、

1. HDL をコピー
2. Quartus® Prime のプロジェクトを開く
3. Platform Designer システムの完成
4. コンパイル
5. 書き込み

を行います。

(手順 1) 第 5-2-2 章『RTL Simulation』にて実行した際に生成された下記フォルダをコピーし、Quartus® Prime の Project フォルダに貼り付けます。

コピーするフォルダ: ¥ihc_work¥test-fpga-sim.prj¥components¥test

貼り付け先フォルダ: ¥ihc_work¥quartus_project

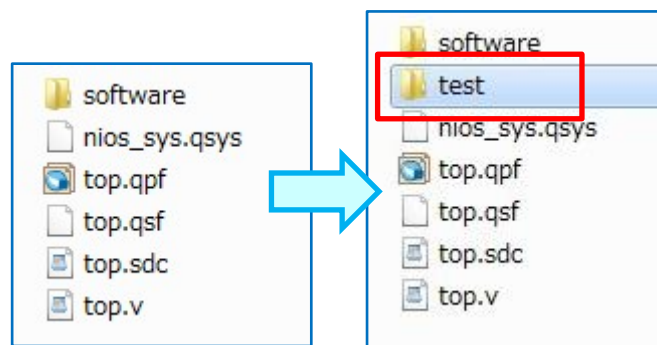




図 6-2-2 test フォルダの貼り付け

test フォルダを Quartus® Prime のプロジェクト・フォルダに用意することで、Platform Designer システム内で Custom Component として呼び出すことができます。

(手順 2) Quartus® Prime を起動します。

(手順 3)  Open Project にて ¥ihc_work¥quartus_project¥top.qpf を選択し、プロジェクトを開きます。

(手順 4)  アイコンをクリックし、Platform Designer を開きます。

(手順 5) nios_sys.qsys を選択し、“開く” ボタンを押します。

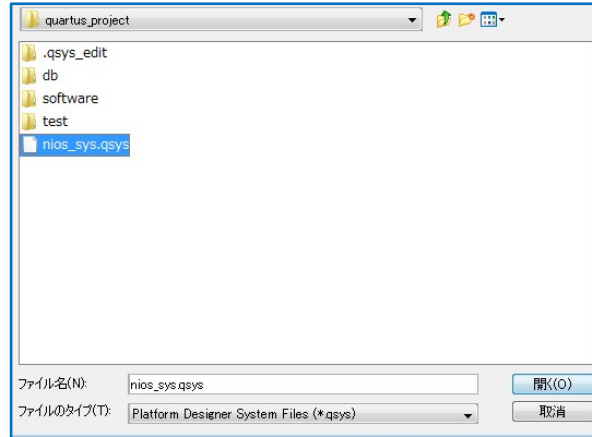


図 6-2-3 Platform Designer

すでに HDL 化した component test を接続した Platform Designer システムが開きます。

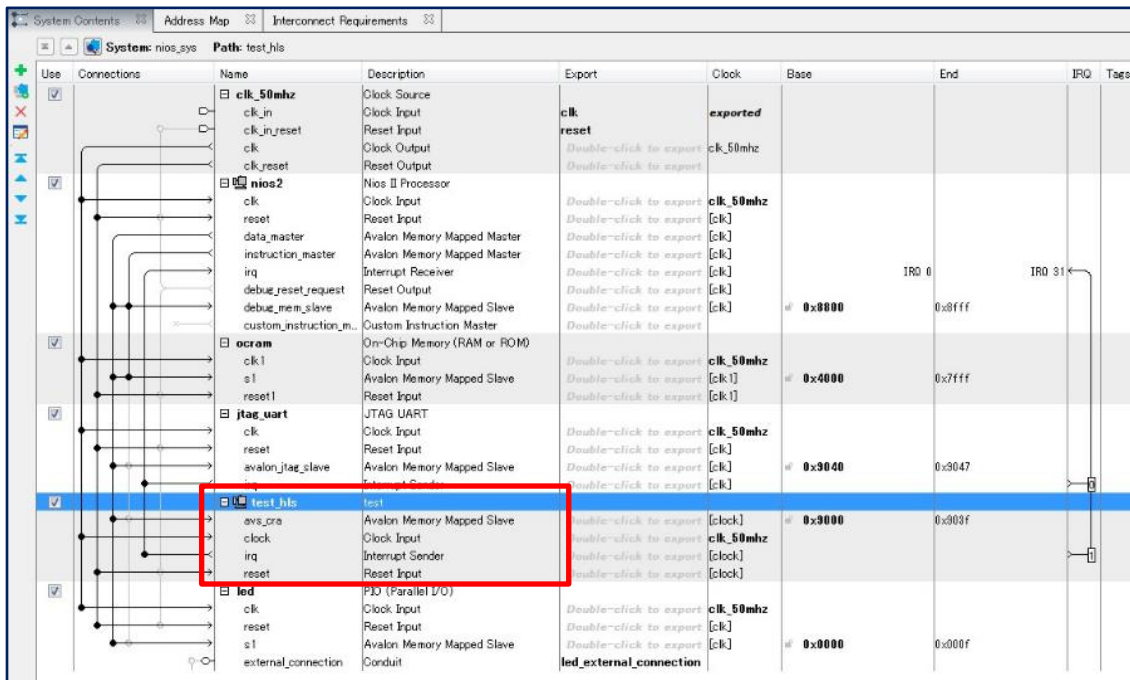


図 6-2-4 nios_sys のシステム

なお、Quartus® Prime のプロジェクト・フォルダに test フォルダを貼り付けたことで、Platform Designer 内の IP Catalog に HLS ▶ test が認識されています。

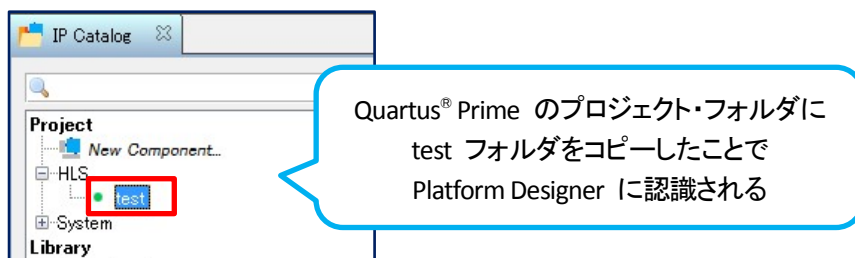


図 6-2-5 Platform Designer 内の IP Catalog

(手順 6) “Generate HDL” ボタンを押し、言語を選択後 “Generate” ボタンを押します。

本資料では、Synthesis 用に Verilog を選択しています。

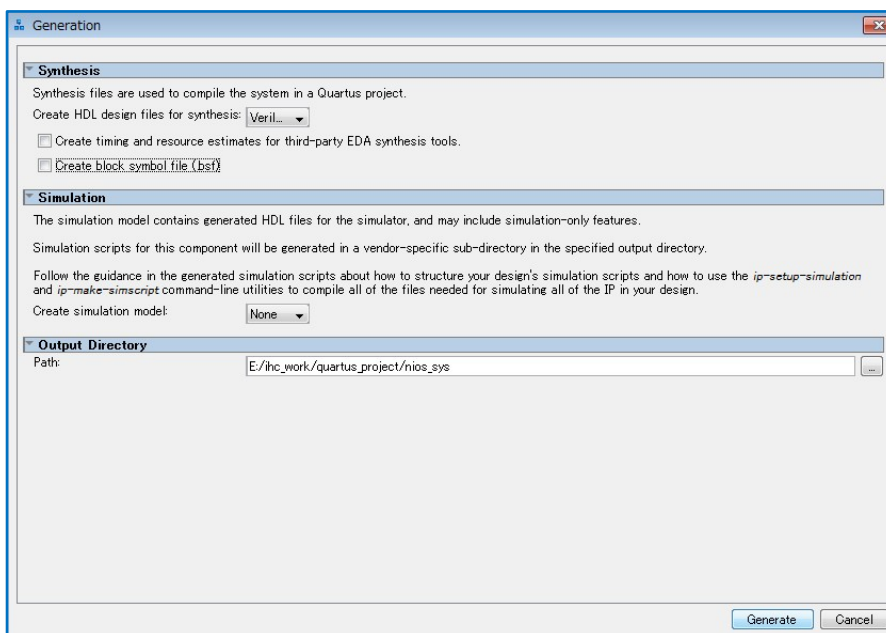


図 6-2-6 Generate

ファイルの生成後、“Close” ボタンを押し、“Finish” ボタンを押して Platform Designer を閉じます。

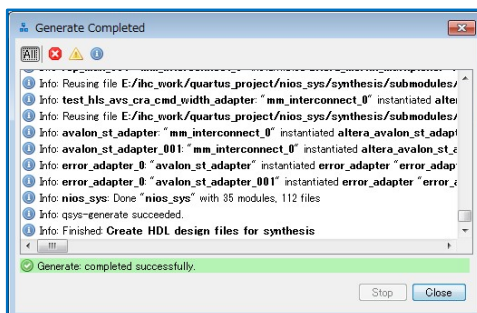



図 6-2-7 Generate Completed

(手順 7) ピン配置などの制約はすべて終了しているため、Start Compilation アイコン  を押し、コンパイルを実行します。マシン・スペックに依存しますが、コンパイルは、約 10 分程度かかります。

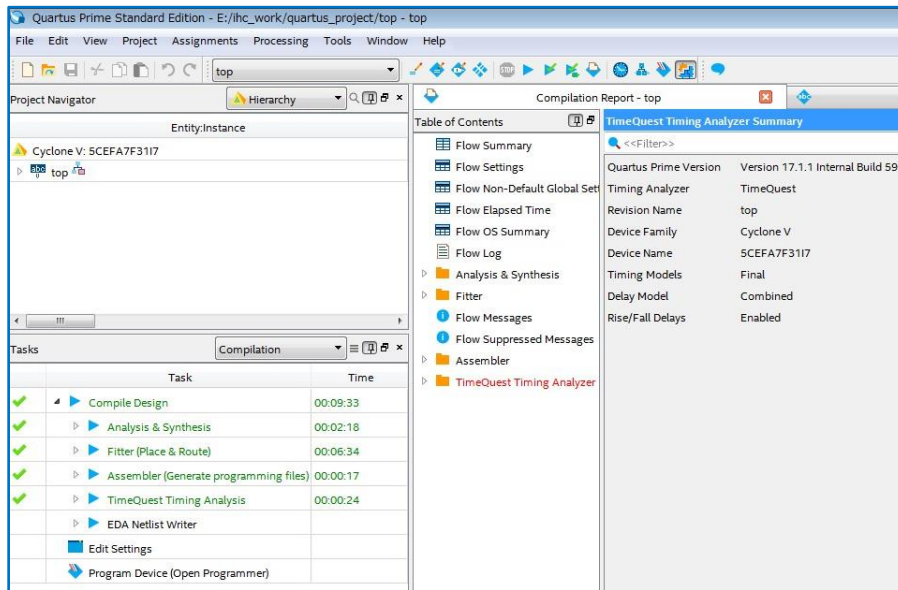




図 6-2-8 Quartus® Prime におけるコンパイル終了

TimeQuest Timing Analyzer にて Error が発生していますが、入出力ピンに対してタイミング制約を行っていないためです。本資料では内部動作周波数が 50MHz を達成しているため、無視しています。

(手順 8) Cyclone® V E FPGA 開発キット上の Cyclone® V E にデータを書き込む準備を行います。

Cyclone® V E FPGA 開発キットと電源ケーブル、Embedded USB-Blaster II ケーブルを接続し、基板に電源を入れます。

(手順 9)  アイコンを押し、Programmer を起動します。

(手順 10)  ボタンを押し、[Hardware Settings] タブにて Available hardware items からケーブルを選択し、“Close” ボタンを押しします。

(手順 11)  ボタンを押し、JTAG Chain 上のデバイスを認識させます。

Cyclone® V E FPGA 開発キットのデフォルト状態では、Cyclone® V E と MAX® V が JTAG Chain 上に接続されています。

File	Device	Checksum	Usercode
<none>	5CEFA7	00000000	<none>
<none>	5M2210Z	00000000	00000000
<none>	CFI_512Mb		

図 6-2-9 Auto Detect の結果

(手順 12) Cyclone® V E を選択し、 ボタンを押します。

(手順 13) %ihc_work%\quartus_project\output_files\top.sof ファイルを選択し、“Open” ボタンを押します。

(手順 14) Program/Configure に を入れ、 ボタンを押し、書き込みを実行します。

File	Device	Checksum	Usercode	Program/Configure	Verify	Blank-Check	Examine	Security Bit	Erase	ISP CLAMP
output_files/top.sof	5CEFA7F31	01BC6467	01BC6467	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<none>	5M2210Z	00000000	00000000	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<none>	CFI_512Mb			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

図 6-2-10 書き込み実施

上記手順により、Nios® II を含むデザインが Cyclone® V E に書き込まれ、動作を開始しています。

第 7 章『Nios® II SBT の操作』により Nios® II 用のソフトウェアを動作させ、HDL 化したシステム全体の動作結果を確認します。

7. Nios® II SBT の操作

Nios® II SBT の操作に関して説明します。

7-1. ソフトウェア・プログラム概要 (test.c)

Cyclone® V E に実装した Embedded Processor である Nios® II 用のソフトウェア・プログラム (test.c) を `¥ihc_work¥quartus_project¥software` に用意しています。

```

1  #include "sys/alt_stdio.h"
2  #include "alt_types.h"
3  #include "altera_avalon_pio_regs.h"
4  #include <stdio.h>
5  #include "system.h"
6  #include <unistd.h>
7  #include "../test/test_csr.h"
8
9
10 #define test_hls_offset 0x9000
11
12 int main()
13 {
14     printf("Hello from Nios II\n");
15     // printf("*****\n");
16
17     int a = 0;
18     int b = 0;
19
20     for (a = 0; a < 10; a++)
21     {
22         for (b = 0; b < 10; b++)
23         {
24             IOWR_32DIRECT(test_hls_offset + TEST_CSR_INTERRUPT_ENABLE_REG, 0, 1);
25             IOWR_32DIRECT(test_hls_offset + TEST_CSR_INTERRUPT_STATUS_REG, 0, 1);
26             printf("*****\n");
27             printf("case %d x %d\n", a, b);
28             printf("*****\n");
29             printf("before write a is %d, b is %d\n", IORD_32DIRECT(test_hls_offset + TEST_CSR_ARG_A_REG, 0), IORD_32DIRECT(test_hls_offset + TEST_CSR_ARG_B_REG, 0));
30             IOWR_32DIRECT(test_hls_offset + TEST_CSR_ARG_A_REG, 0, a);
31             IOWR_32DIRECT(test_hls_offset + TEST_CSR_ARG_B_REG, 0, b);
32             printf("after write a is %d, b is %d\n", IORD_32DIRECT(test_hls_offset + TEST_CSR_ARG_A_REG, 0), IORD_32DIRECT(test_hls_offset + TEST_CSR_ARG_B_REG, 0));
33             printf("\n");
34
35             IOWR_32DIRECT(test_hls_offset + TEST_CSR_START_REG, 0, 1);
36
37             usleep(50000);
38
39             printf("return data is %d\n", IORD_32DIRECT(test_hls_offset + TEST_CSR_RETURNDATA_REG, 0));
40             printf("\n");
41             printf("%d x %d is %d\n", a, b, IORD_32DIRECT(test_hls_offset + TEST_CSR_RETURNDATA_REG, 0));
42             printf("*****\n");
43         }
44     }
45     printf("finish!\n");
46     return 0;
47 }
48
49
50

```

図 7-1-1 Nios® II 用ソフトウェア・プログラム test.c

test.c では、第 5-2-3 章『動作の確認』にて確認した通り、レジスタを制御することで test コンポーネントへのアクセスを行っています。

- 24 行目: Interrupt を Enable (TEST_CSR_INTERRUPT_ENABLE_REG に 0x1)
- 25 行目: Interrupt Status を Clear (TEST_CSR_INTERRUPT_STATUS_REG に 0x1)
- 30 行目: Argument a に Write (TEST_CSR_ARG_A_REG)
- 31 行目: Argument b に Write (TEST_CSR_ARG_B_REG)
- 35 行目: Start を Write (TEST_CSR_START_REG に 0x1)
- 39 行目: Return Data (TEST_CSR_RETRURNDATA_REG)

7-2. Nios® II SBT の操作

Nios® II SBT の基本フローは下記です。

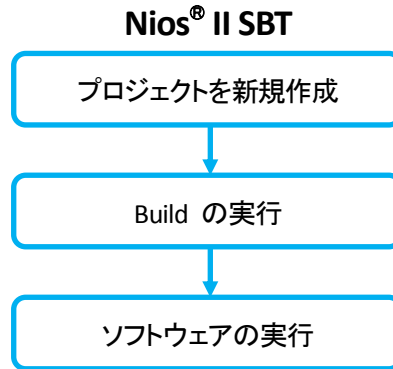


図 7-2-1 Nios® II SBT のフロー・イメージ

上記に従って操作説明します。

(手順 1) Nios® II SBT を起動します。

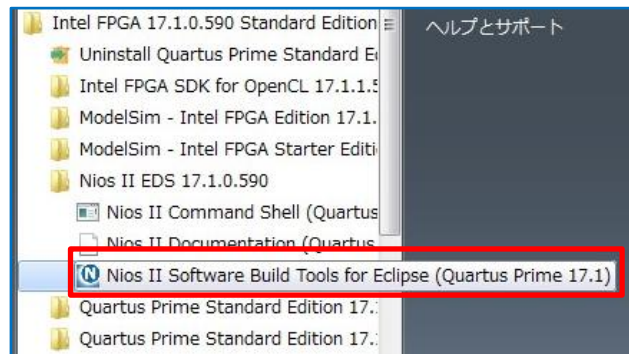


図 7-2-2 スタート・メニュー

(手順 2) 下記フォルダを選択し、“OK” ボタンを押します。

¥ihc_work¥quartus_project¥software

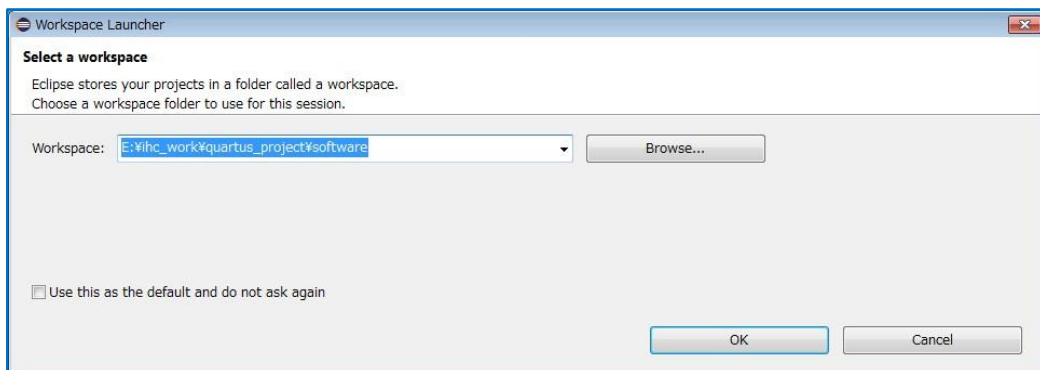


図 7-2-3 Workspace の選択

(手順 3) File ▶ New ▶ Nios II Application and BSP from Template を選択します。

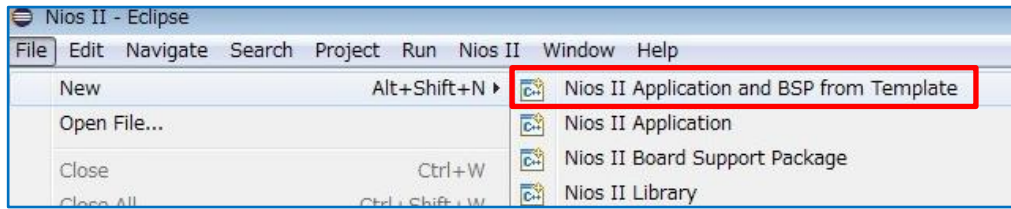


図 7-2-4 File メニュー

(手順 4) 下記 4 項目を指定し、“Finish” ボタンを押します。

- SOPC Information File name: ¥ihc_work¥quartus_project¥nios_sys.sopcinfo
- CPU name: nios2
- Project name: 任意
図 7-2-5 例では、nios_soft
- Templates: Blank Project

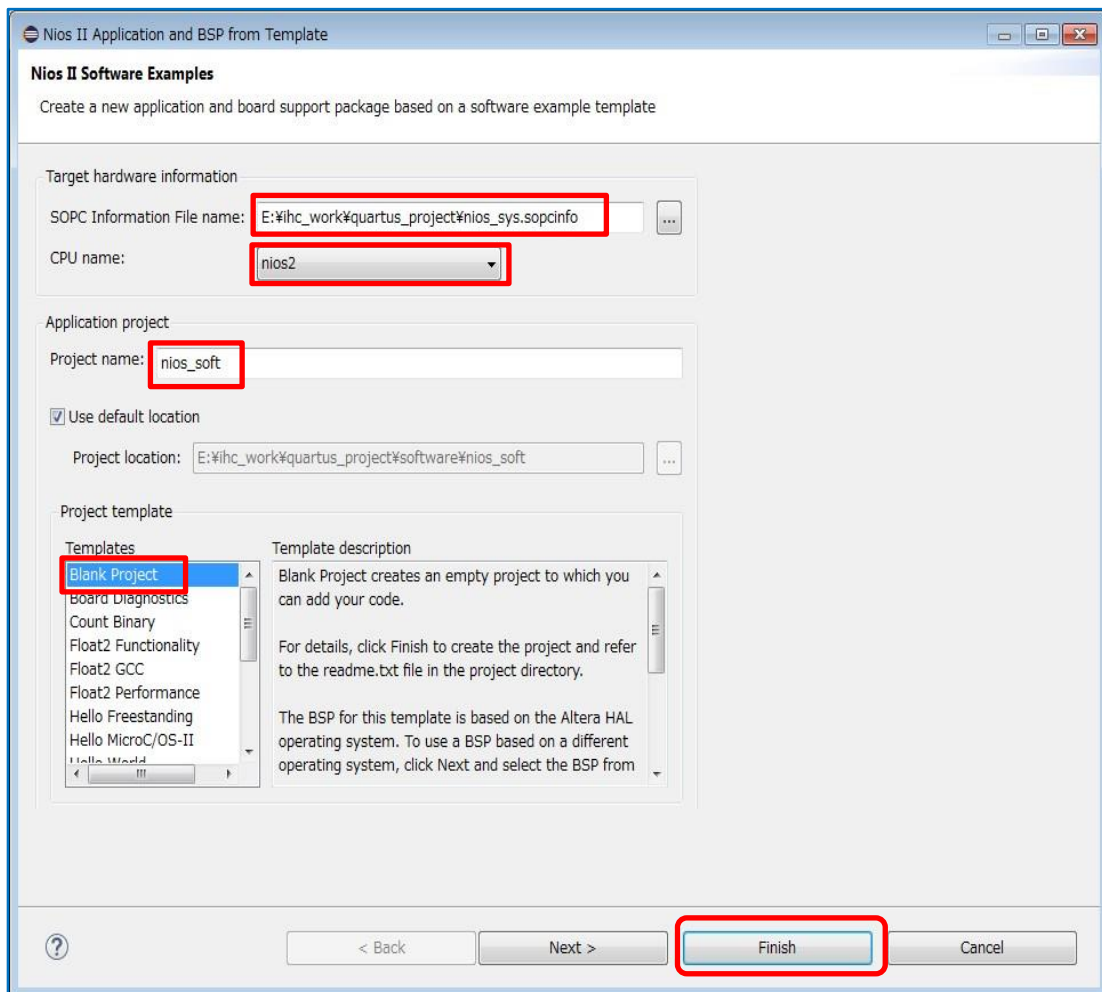


図 7-2-5 Nios II Application and BSP from Template

(手順 5) エクスプローラを開き、E:\ihc_work\quartus_project\software\test.c をドラッグし、Nios® II SBT 上の nios_soft フォルダにドロップします。

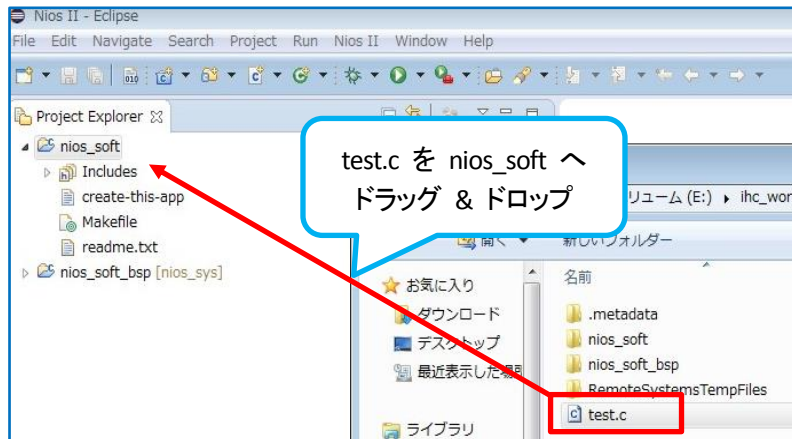


図 7-2-6 test.c のドラッグ & ドロップ

Copy files にチェックを入れ、“OK” ボタンを押します。

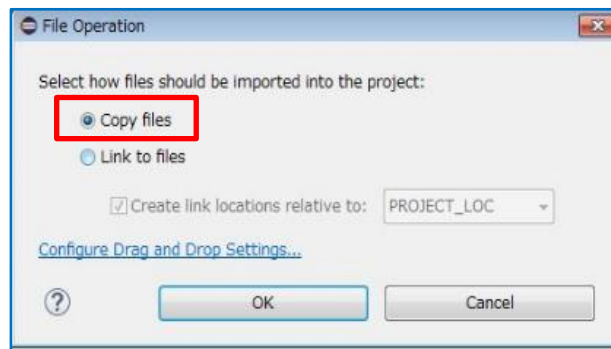


図 7-2-7 File Operation

プロジェクト nios_soft に test.c が反映されます。



図 7-2-8 nios_soft

(手順6) nios_soft_bsp を右クリックし、Nios II ▶ BSP Editor を選択します。

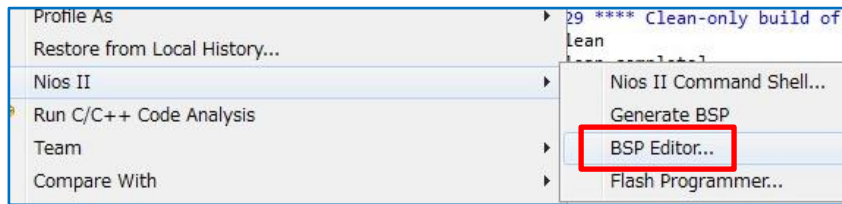


図 7-2-9 BSP Editor 起動

(手順7) [Main] タブにて下記項目を指定します。

stdout: jtag_uart

enable_small_c_library: ✓

enable_reduced_device_drivers: ✓

それ以外はデフォルト

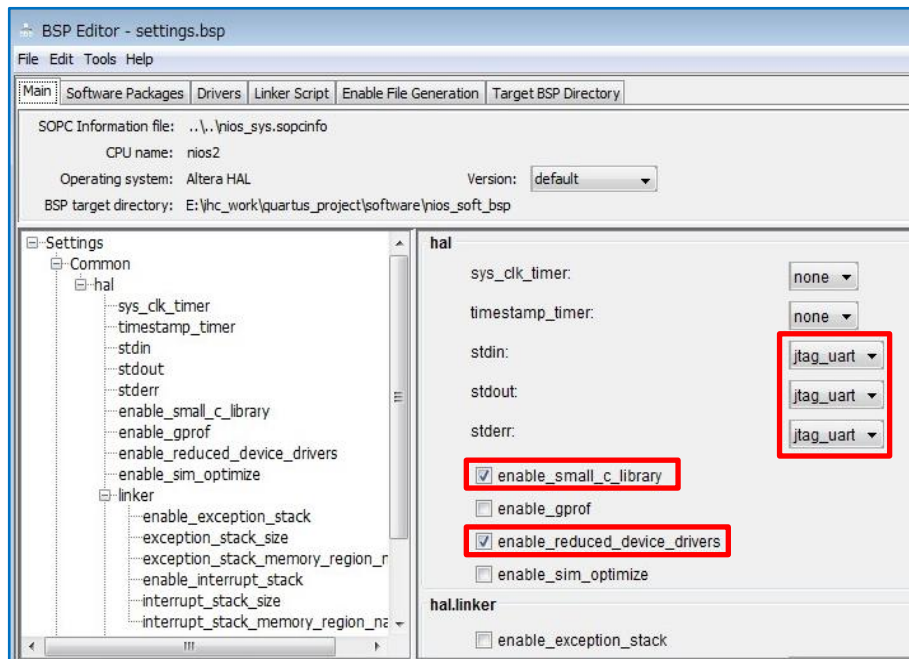


図 7-2-10 Main タブ

(手順8) [Linker Script] タブにて Linker Region Name などが ocram になっている確認します。

Linker Section Name	Linker Region Name	Memory Device Name
.bss	ocram	ocram
.entry	reset	ocram
.exceptions	ocram	ocram
.heap	ocram	ocram
.rodata	ocram	ocram
.rwdata	ocram	ocram
.stack	ocram	ocram
.text	ocram	ocram

図 7-2-11 Linker Script タブ

(手順 9) “Generate” ボタンを押し、Generate 終了後、“Exit” ボタンを押し、BSP Editor を閉じます。

(手順 10) nios_soft を右クリックし、Build Project を選択し、Build を実行します。

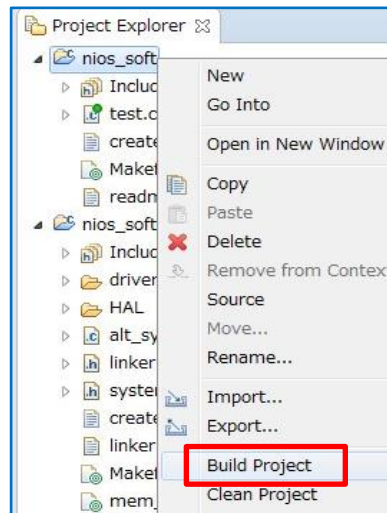


図 7-2-12 Build Project

(手順 11) nios_soft を右クリックし、Run As ▶ 3 Nios II Hardware を選択します。

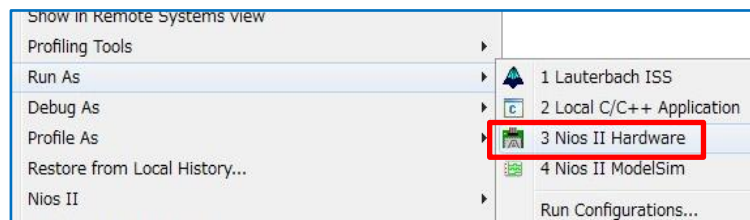


図 7-2-13 Nios II Hardware

Nios II Console 上に Hello from Nios II が表示され、 0×0 から 9×9 まで演算結果が表示され finish!! が表示され、Nios® II プログラムが終了します。

```

nios_soft Nios II Hardware configuration - cable: USB-BlasterII on localhost [US
Hello from Nios II

*****
      case 0 x 0
*****
before write a is 0, b is 0
after write a is 0, b is 0

return data is 0

0 x 0 is 0
*****
*****
      case 0 x 1
*****
before write a is 0, b is 0

9 x 7 is 63
*****
*****
      case 9 x 8
*****
before write a is 9, b is 7
after write a is 9, b is 8

return data is 72

9 x 8 is 72
*****
*****
      case 9 x 9
*****
before write a is 9, b is 8
after write a is 9, b is 9

return data is 81

9 x 9 is 81
*****
*****
finish!!
    
```

図 7-2-14 Nios II の処理結果

実行後、Terminate and Remove Launch ボタンを押します。

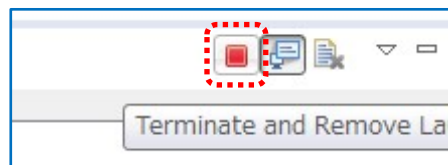


図 7-2-15 Terminate and Remove Launch

以上で一連の操作手順は終了です。

改版履歴

Revision	年月	概要
1	2018 年 4 月	初版
2	2018 年 9 月	<p>誤記訂正</p> <ul style="list-style-type: none"> • Page.12 誤) "SOURCE_FILEStest.cpp test_tb.cpp" set "HLS_CXX_FLAG=" 正) "SOURCE_FILES=test.cpp test_tb.cpp "set "HLS_CXX_FLAG=" • Page.14 誤) "SOURCE_FILEStest.cpp test_tb.cpp" および built test-x86-64 正) "SOURCE_FILES=test.cpp test_tb.cpp " および build test-x86-64 • Page.18 誤) ¥ihc_work¥test-fpga-sim.prj¥components¥test¥test.csr.h 正) ¥ihc_work¥test-fpga-sim.prj¥components¥test¥test_csr.h • Page.37 誤) [Linker] タブ 正) [Linker Script] タブ

免責およびご利用上の注意

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

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