Altera CycloneII/IV(EP2C8Q208/EP4CE15) ボードマニュアル

株式会社日昇テクノロジー

http://www.csun.co.jp

2012/02/18

修正履歴

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2012/01/15
2	Ver1.1	uCLinux 移植手順を追加（第八章）	2012/2/18

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがあります。最新版は弊社ホームページからご参照ください。「http://www.csun.co.jp」

目次

第一章 Cyclone II/IV ボード機能概要

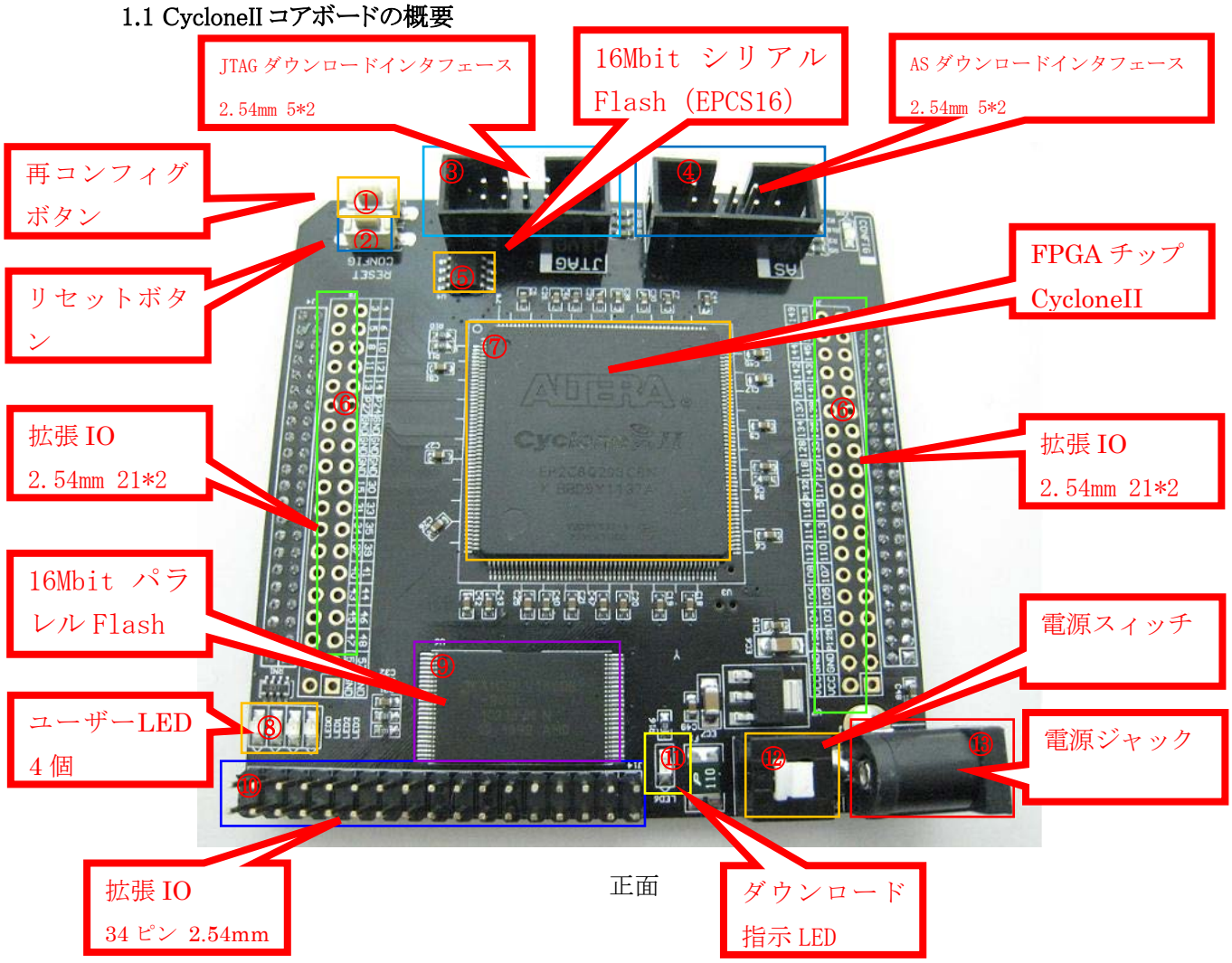
FPGA コアボードと拡張ボードで構成されている。

各種デバイスのリソース一覧：

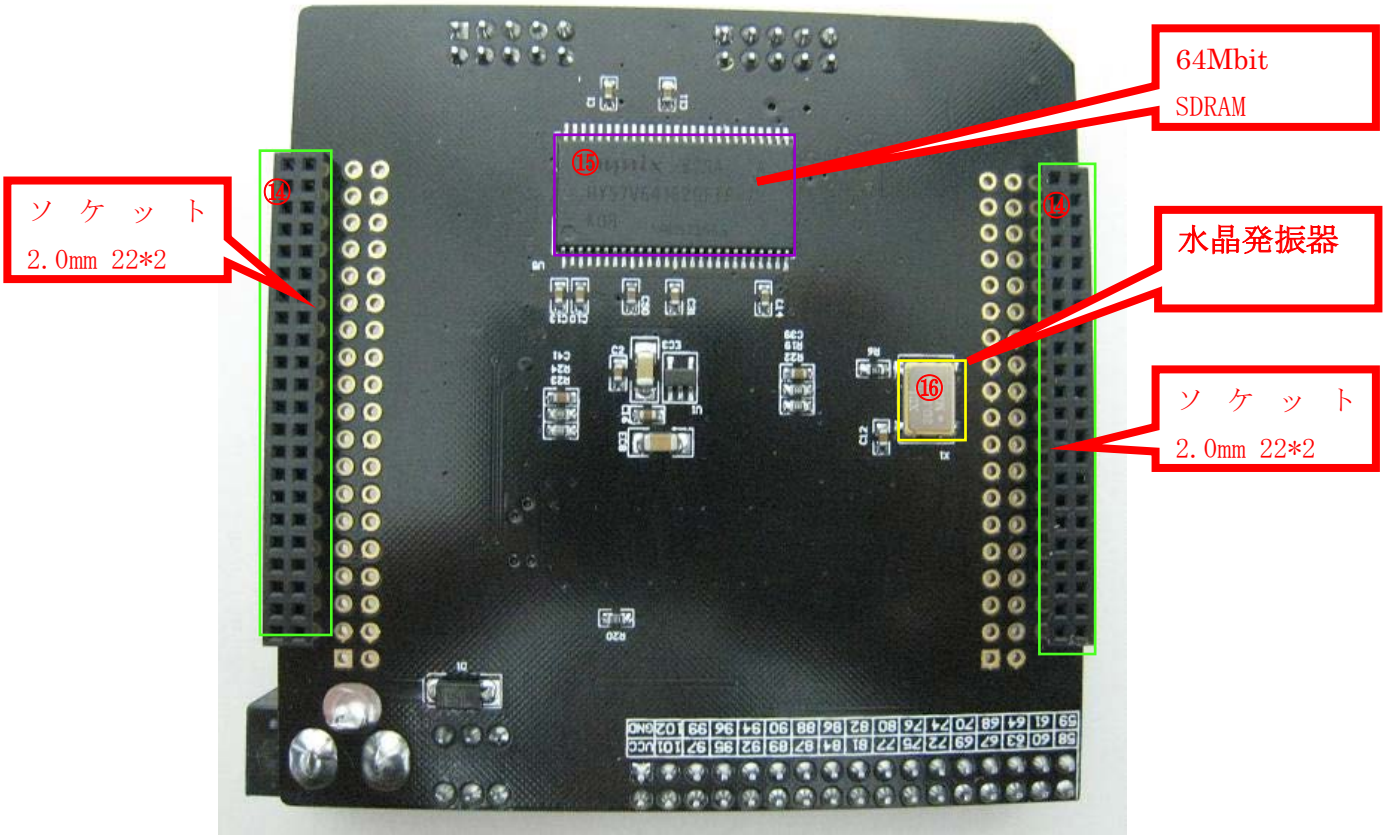
	デバイス				
特徴	EP2C5	EP2C8	EP2C20	EP3C25	EP4CE15
ロジック・エレメント数	4,608	8,256	18,752	24,624	15,408
RAM 総ビット数	119,808	165,888	239,616	608,256	1,020,096
エンベデッド乗計算機数	13	18	26	66	56※1
PLL 数	2	2	4	4	4
IO 数	143	139	142	148	343

※１：CycloneIV でエンベデッド乗計算機数ではなく、18 x 18 マルチプライヤ数になる

1.1 CycloneII コアボードの概要



正面



裏面

①再コンフィグボタン　：1個（押すとEPCSの情報をFPGAにロードする）
②リセットボタン　　：1個
③JTAGダウンロードインタフェース：1個

SOFファイルをダウンロードする。直接FPGAに書き込んで、速度は速いですが、電源切れたらなくなる。デバッグする時に利用するのをお勧め。

④ASダウンロードインタフェース　：1個

POFファイルをダウンロードする。コンフィギュレーションデバイスEPCS64に書き込む。速度はJTAGより遅いですが、電源切れても保持する。最後のプログラム或いは電源を再起動が必要な場合利用する。※書き込み終了したら、電源を切って、ケーブルを抜けてから、正常に次の操作が出来る。

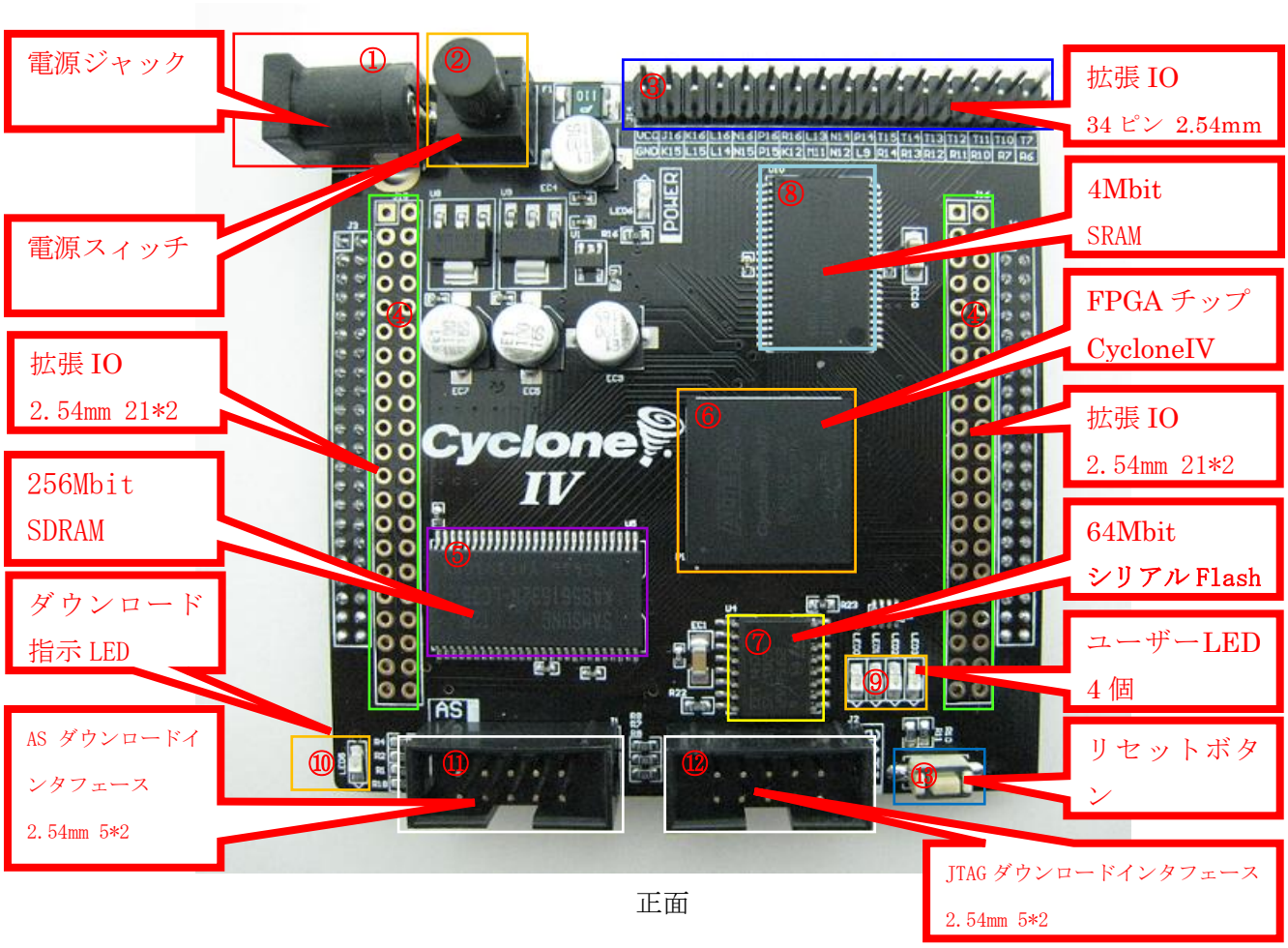
⑤シリアルFlash　　：16Mbit(M25P16)、EPCS16
⑥拡張IO　　：2.54mm 21*2
⑦FPGAチップ　　：CycloneII EP2C8Q208C
⑧ユーザーLED　　：4個
⑨パラレルFlash　　：16Mbit（2M*8bit）
⑩拡張IO　　：34ピン　2.54mm 17*2（半田付き）
⑪ダウンロード指示LED　：1個

⑫電源スィッチ：1個

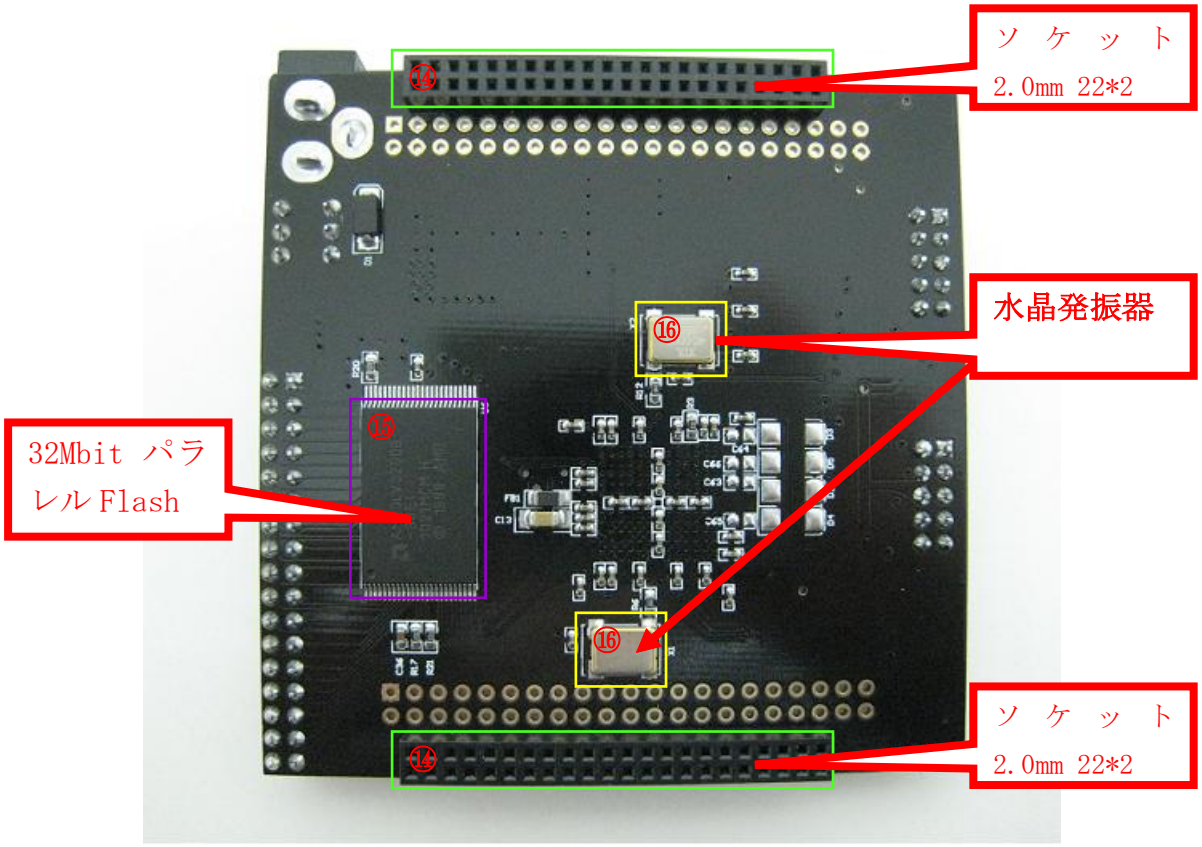
⑬電源ジャック：　5V/1A　2.1mmφ、極性：センタープラス

⑭ソケット　　：　2.0mm 22*2(拡張ボード接続用)
⑮SDRAM　　：　64Mbit(4*16bit)
⑯水晶発振器　：　20M
外形寸法　　：　76×76mm
4層基板、回路図を提供しております、豊富なサンプルソースも公開しております。

1.2 CycloneIV コアボードの概要



正面



裏面

①電源ジャック：　5V/2A　2.1mmφ、極性：センタープラス

②電源スィッチ：1個
③拡張IO：　34ピン　2.54mm 17*2（半田付き）
④拡張IO：　2.54mm 21*2
⑤SDRAM：　256Mbit(16*16bit)
⑥FPGAチップ：　CycloneIV EP4CE15F17C8N
⑦シリアルFlash：64Mbit(EPCS64)
⑧SRAM：　4Mbit(256K*16bit)
⑨ユーザーLED：4個
⑩ダウンロード指示LED：1個
⑪ASダウンロードインタフェース：1個

POFファイルをダウンロードする。コンフィギュレーションデバイスEPCS64に書き込む。速度はJTAGより遅いですが、電源切れても保持する。最後のプログラム或いは電源を再起動が必要な場合利用する。※書き込み終了したら、電源を切って、ケーブルを抜けてから、正常に次の操作が出来る。

⑫JTAGダウンロードインタフェース：1個

SOFファイルをダウンロードする。直接FPGAに書き込んで、速度は速いですが、電源切れたらなくなる。デバッグする時に利用するのをお勧め。

⑬リセットボタン：1個

⑭ソケット：2.0mm 22*2(拡張ボードと繋がる)

⑮パラレルFlash：32Mbit(4M*8bit)

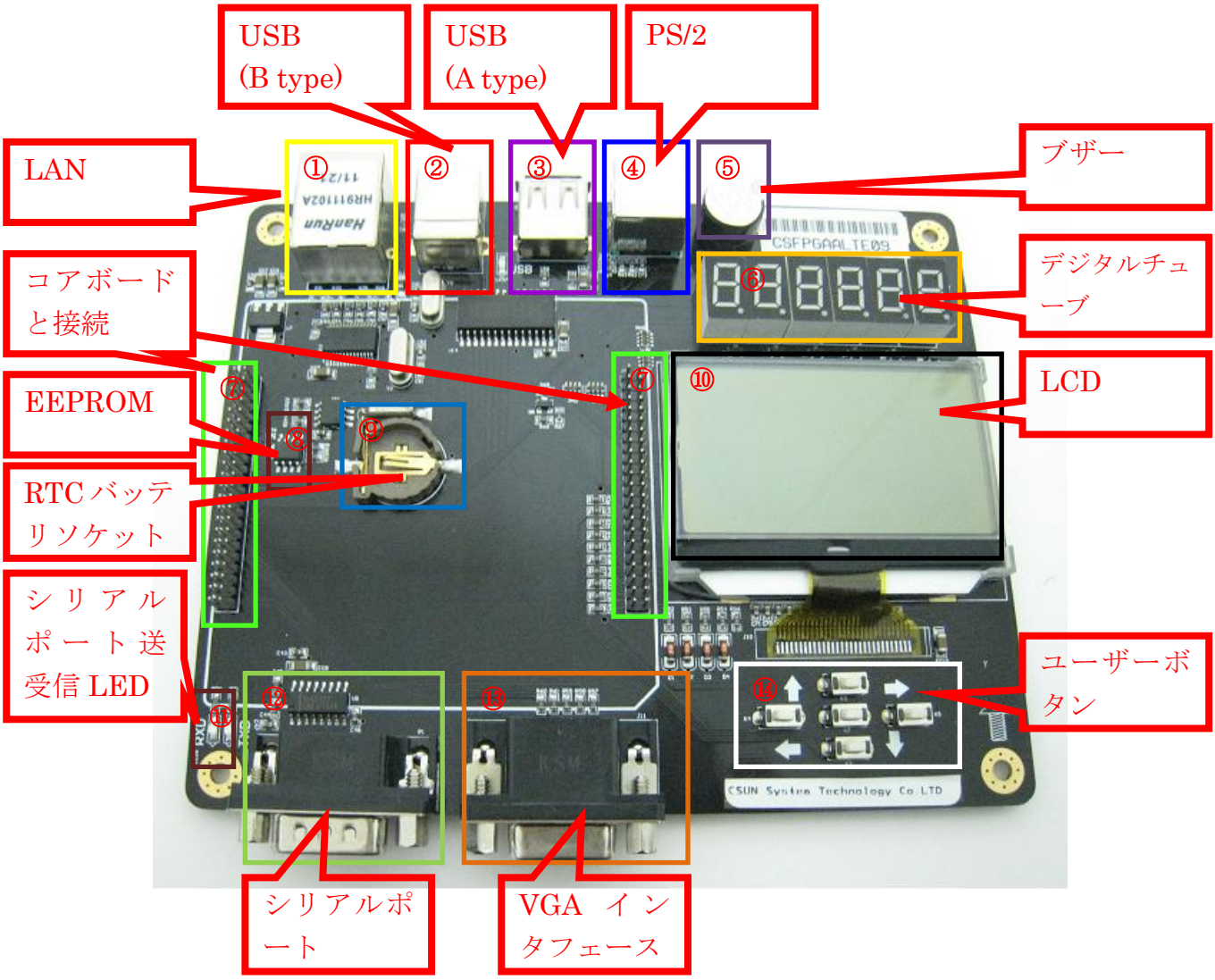
⑯水晶発振器：50M/40M

外形寸法：76×76mm

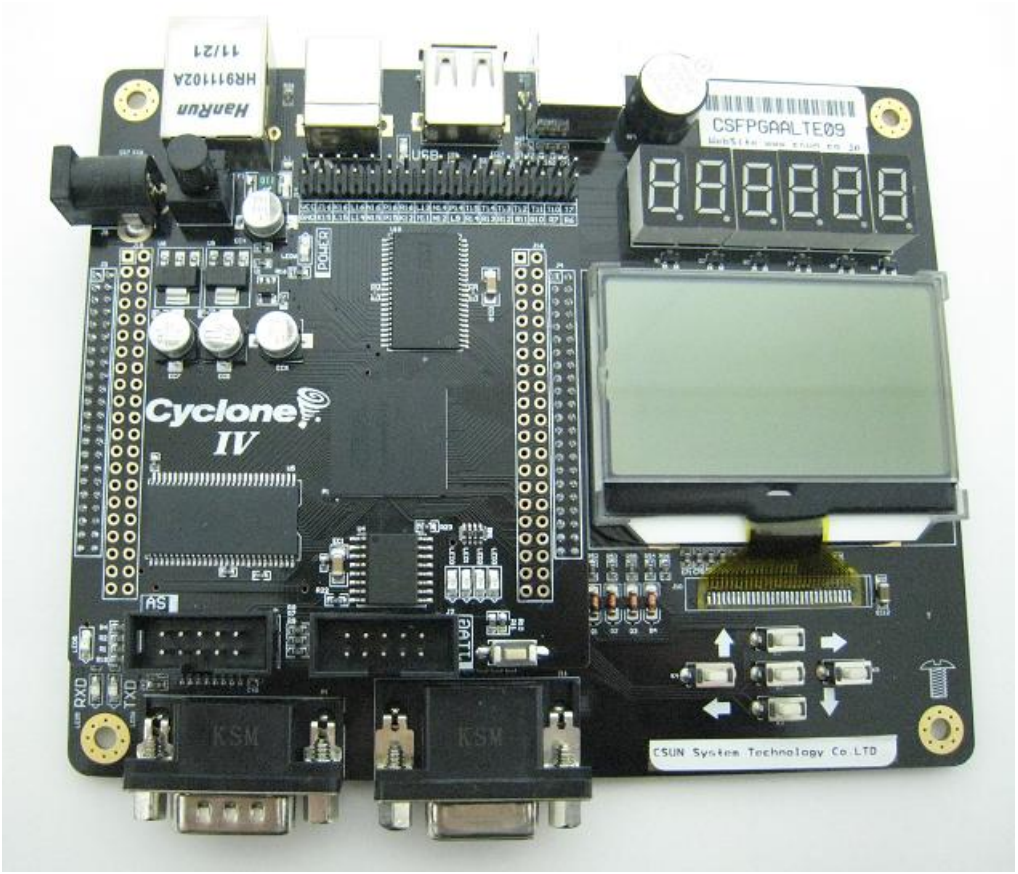
4層基板、回路図を提供しております、豊富なサンプルソースも公開しております。

1.3 FPGA 拡張ボードの概要

※CycloneII と CycloneIV コアボードは両方もこの拡張ボードで使えます。



①**LAN インタフェース**：ENC28J60 を搭載
②**USB**：B タイプ、CH376 チップ使用（デバイス方式）
③**USB**：A タイプ、CH376 チップ使用（ホスト方式）、FAT16 及び
FAT32 フォームウエアを内蔵、USB デバイス（ハードデスク、
メモリ、カードリーダ等）をサポート
④**PS/2**：PS/2 インタフェースのキーボードとマウスを接続可能
⑤**ブザー**
⑥**デジタルチューブ×6**
⑦**34 ピン 2.0mm 22*2(コアボードと繋がる)**
⑧**EEPROM**：型番が MSP16 の 24LC04 チップを使用、EPC16S と完全相性で
はなく、24LC04 が 512*8bit の EEPROM となり、
IIC インタフェースをサポート
⑨**RTC バッテリソケット**（DALLAS 社の DS1302 チップを使われ、このチップは高性能、低消費電力、RAM 付きのリアルタイムクロックチップとなります。）
⑩**128×64 LCD**：ST7565P コントロールチップを使い、DC/DC 回路内蔵、
ソフトウエアでコントラストを調整できます。
⑪**シリアルポート送受信 LED（左：受信；右：送信）**
⑫**シリアルポート**
⑬**VGA インタフェース**
⑭**ユーザーボタン**
外形寸法：146×113mm※突起物は除く
コアボードと接続する様子：


Cyclone
IV
CSUN System Technology Co LTD

透明保護板を付ける様子

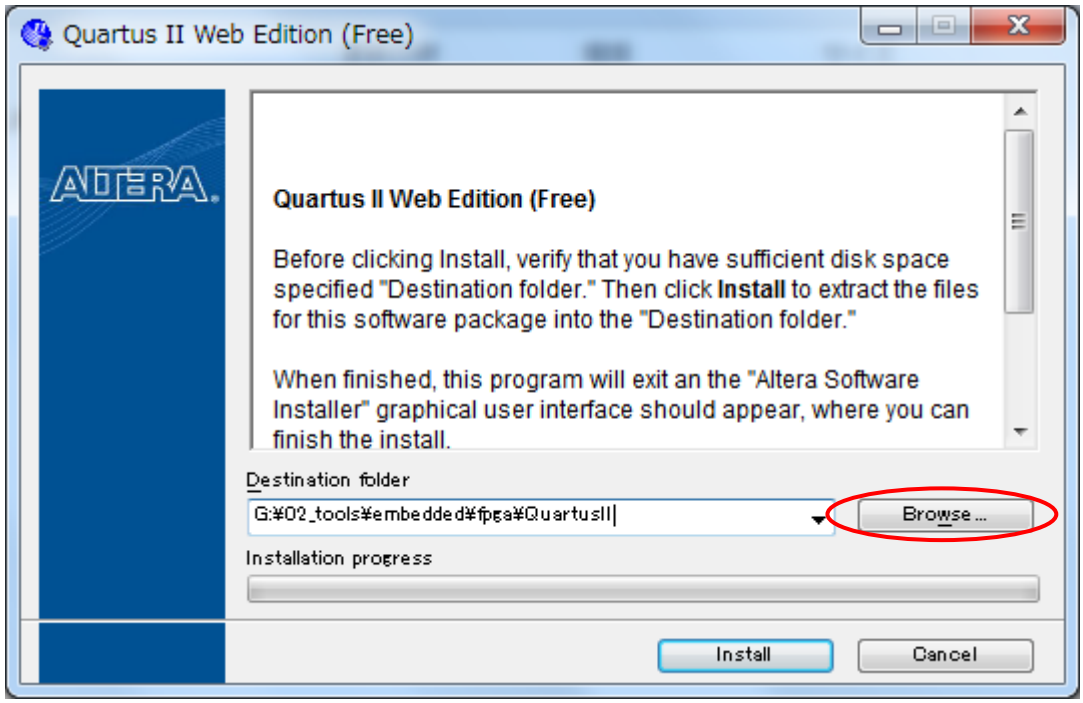
第二章 開発ツールをインストール

CPLD/FPGA の開発には、ALTERA から Quartus II Web Edition という無償版のツールが公開されているのでこちらを利用します。Quartus II には別に製品版があり、Web Edition は使用できるデバイスなどに制限がありますが、CycloneII/IV に関しては、どのデバイスも使用できるのでまったく問題ありません。Quartus II Web Edition は、総合開発環境になっており、このソフトウェアだけで、ソース・エディタや I/O ピンのアサインメント、論理合成、デバイスの書き込み用のプログラムなど、CPLD/FPGA の開発に必要な機能がすべて含まれています。また、Nios II エンベデッド・デザイン・スイートは Nios プロセッサー用の開発ツールです。

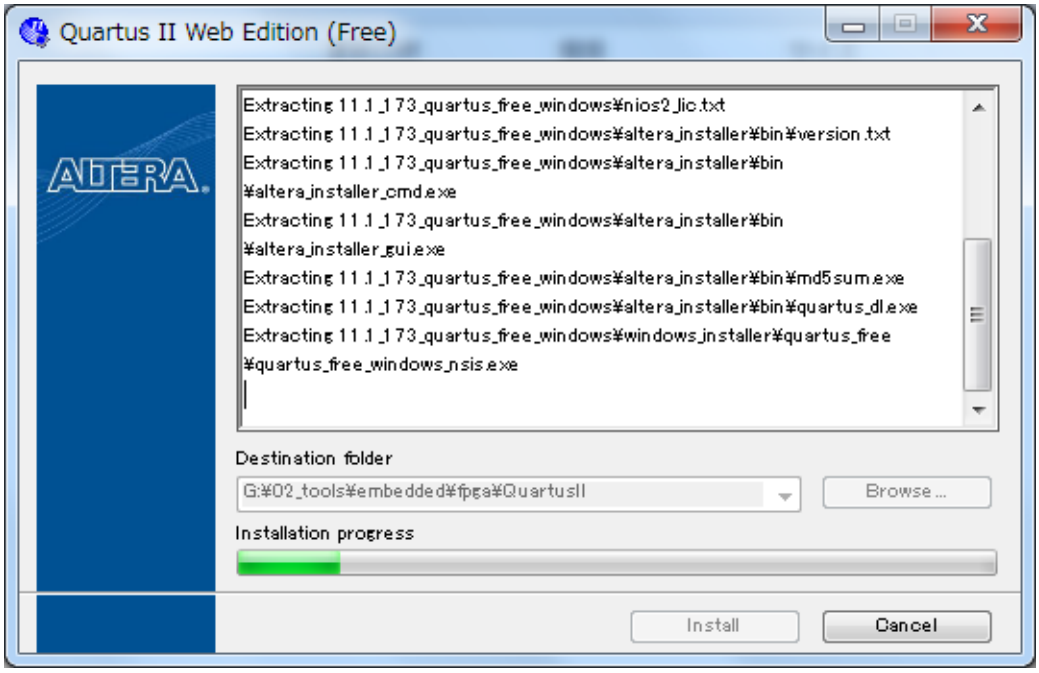
Quartus II Web Edition と Nios II エンベデッド・デザイン・スイートのダウンロードは、次の URL から行うことができます。
http://www.altera.co.jp/support/software/download/nios2/dnl-nios2.jsp
なお、ダウンロードする際は、最初に ALTERA のページにサイン・インを行い、ユーザ情報を登録する必要があります。現時点最新版は v11.1 です、インストールした後、ライセンス・ファイルを入れずそのまま利用できです。ライセンスを取得したい場合、Altera 社或いは代理店に連絡ください。

2.1 Quartus II Web Edition をインストールする

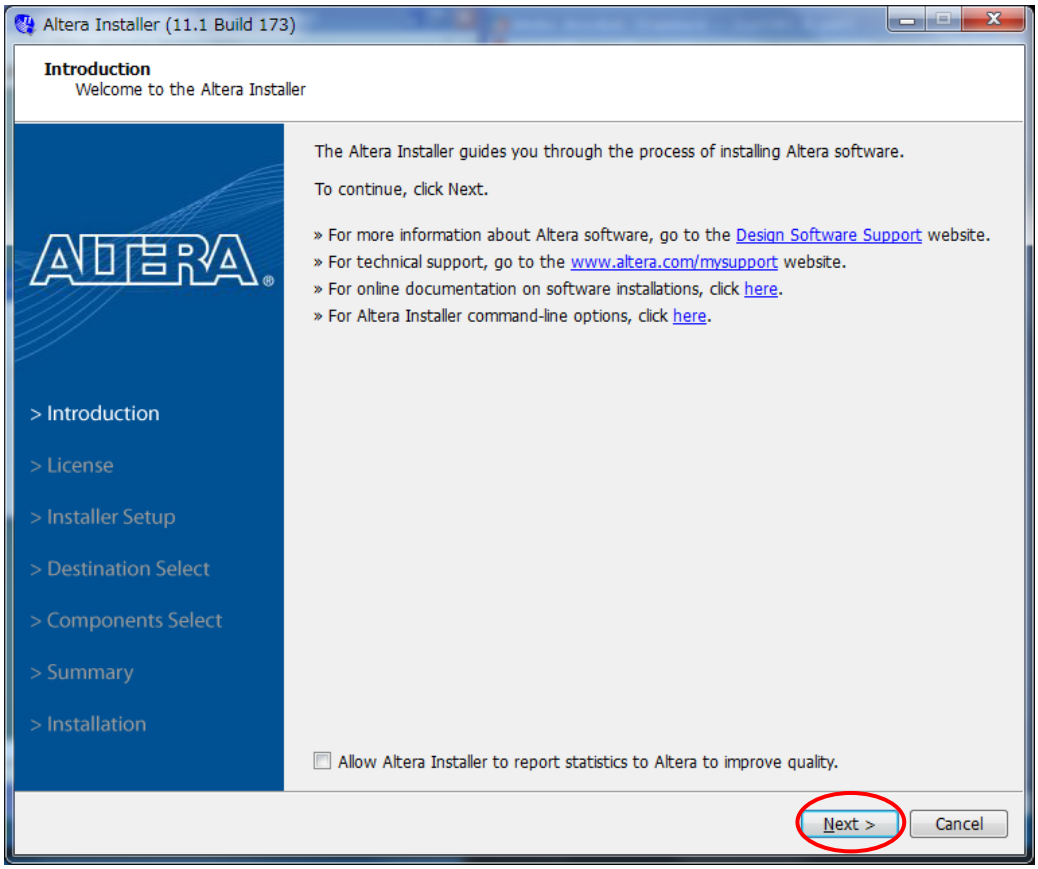
ダウンロードしたインストールファイルをダブクリック
QuatusII：
http://download.altera.com/akdlm/software/acds/11.1/173/standalone/11.1_173_quartus_free_windows.exe
NiosII:
http://download.altera.com/akdlm/software/acds/11.1/173/standalone/11.1_173_legacy_nios2_windows.exe

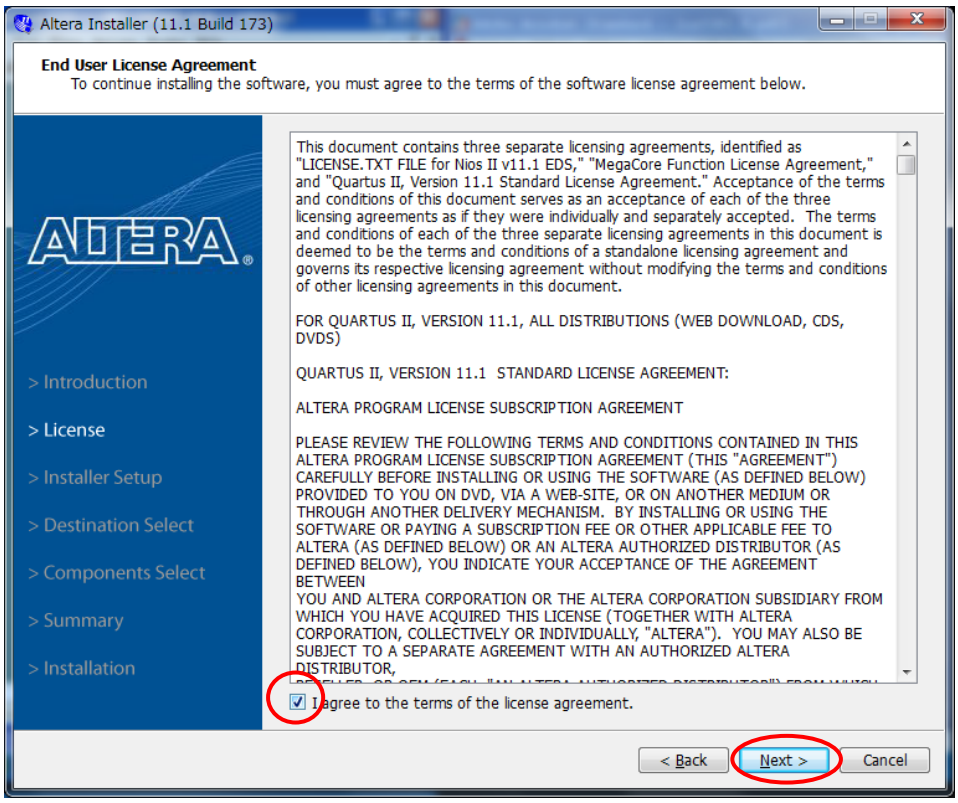


「Browse...」ボタンをクリックして解凍先を変更できます。
変更が終わったら、「Install」ボタンをクリックしてインストールを始めましょう。



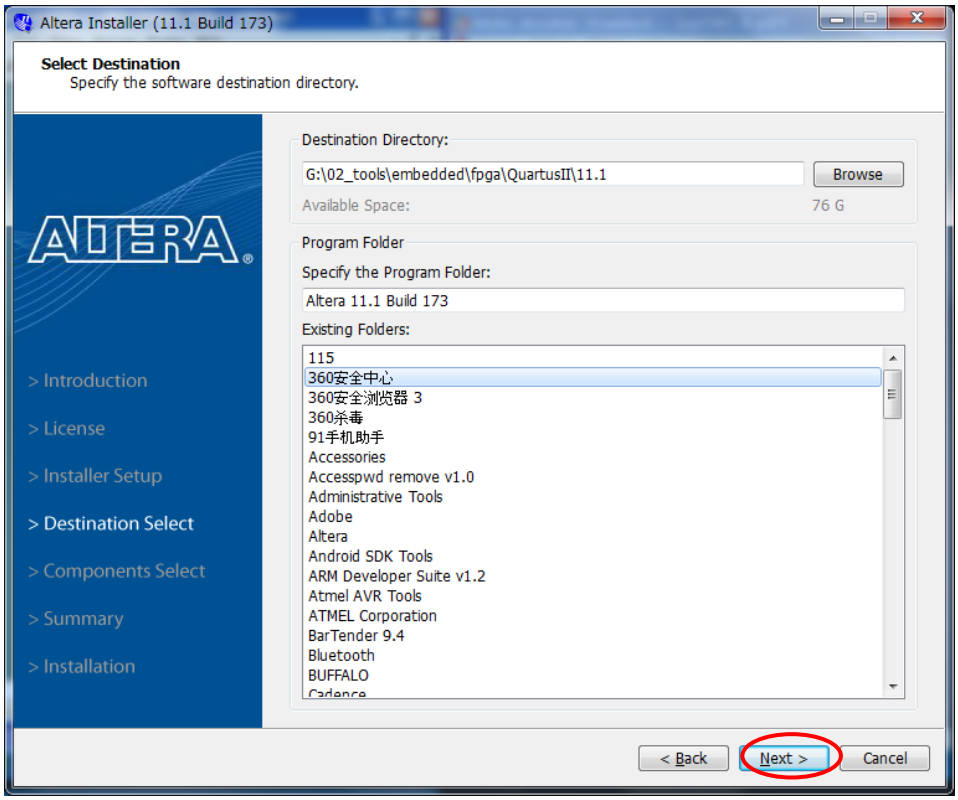
解凍完了後、正式にインストールに入ります。





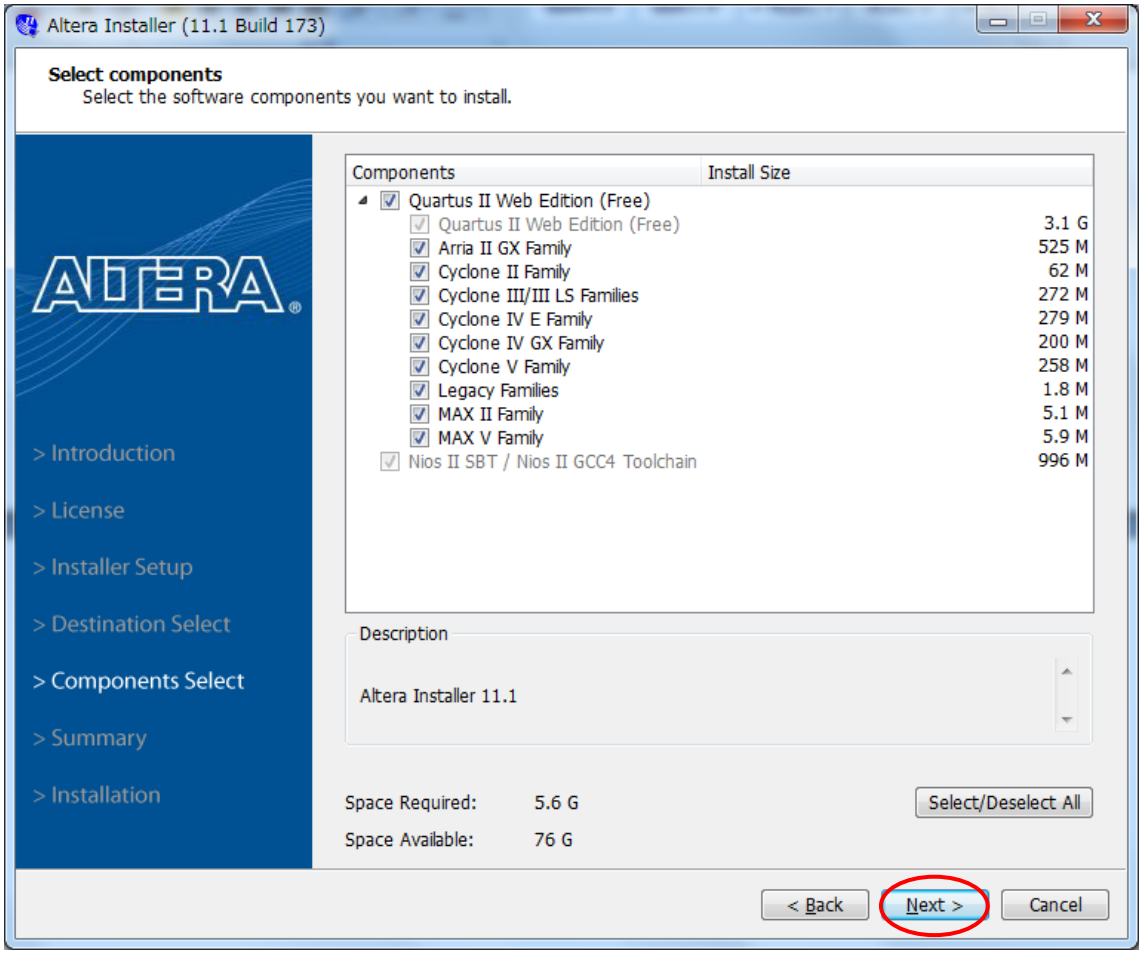
英文のライセンスが出てきます。同意できる場合は、「I accept the terms of the license

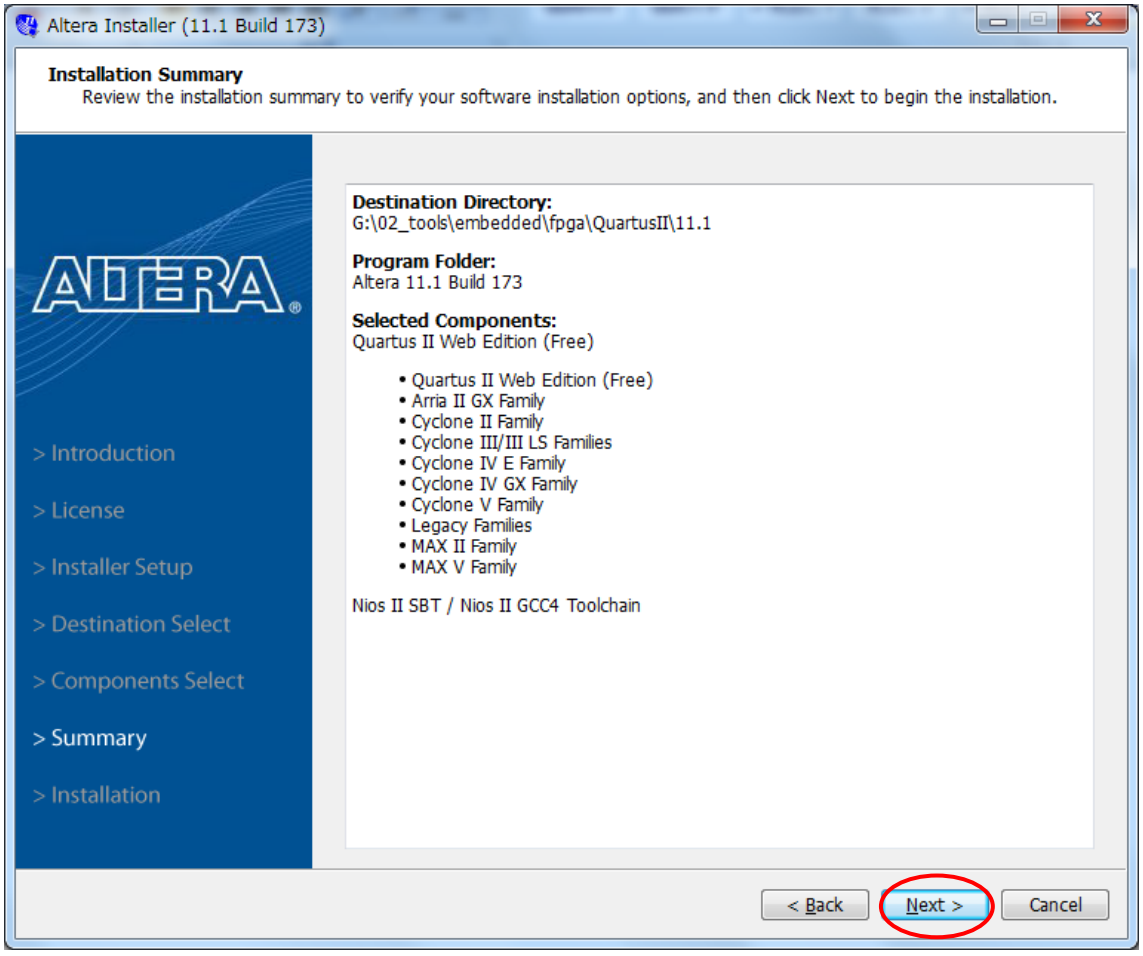
agreement」を選択して、「Next」ボタンを押します。



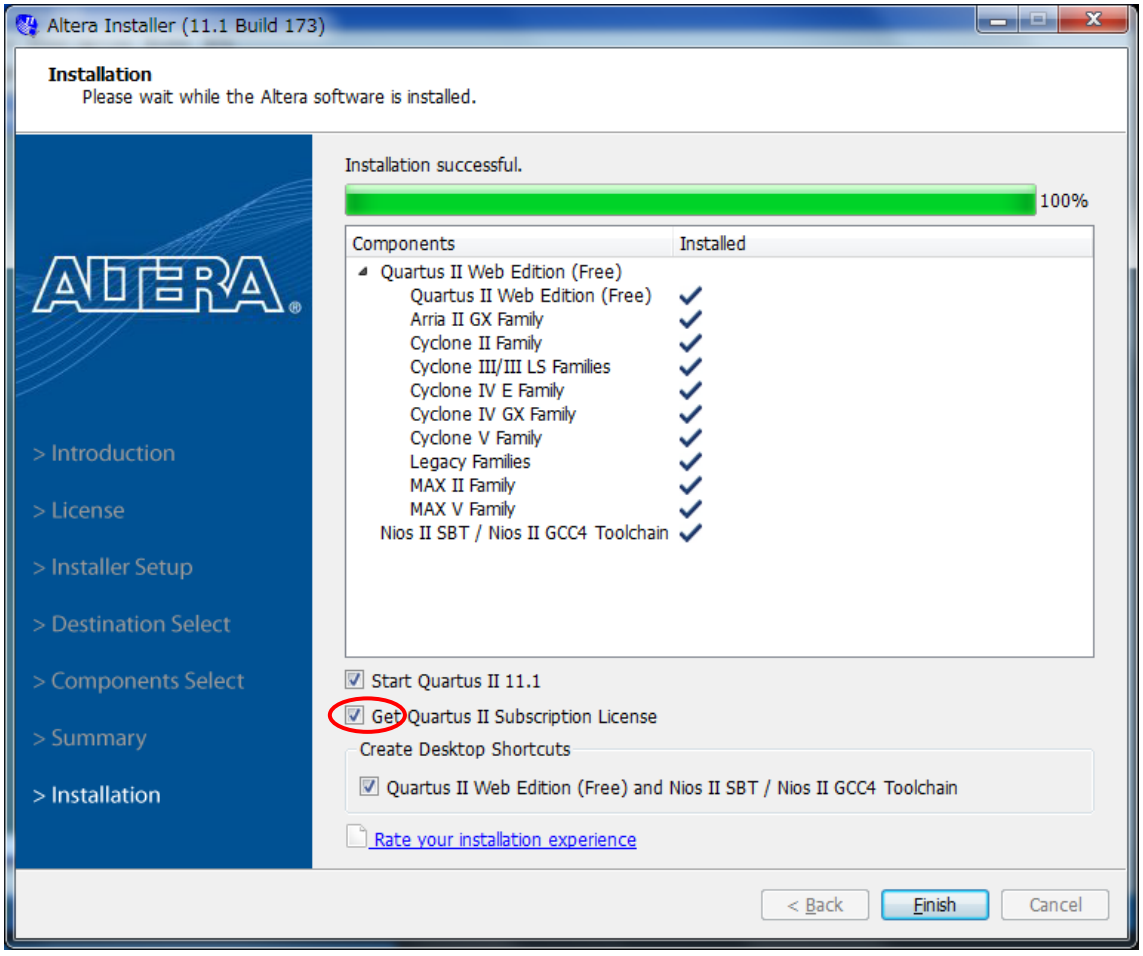
インストール先を変更する場合、「Browse...」をクリック

「Next」ボタンを押し、次のコンポーネント選択画面が出てきます。必要のコンポーネントだけを選択できますが、ここデフォルトのまま進みます。


Altera Installer (11.1 Build 173)
Select components
Select the software components you want to install.
> Introduction
> License
> Installer Setup
> Destination Select
> Components Select
> Summary
> Installation
Components
Install Size
Quartus II Web Edition (Free)
Quartus II Web Edition (Free) 3.1 G
Arria II GX Family 525 M
Cyclone II Family 62 M
Cyclone III/III LS Families 272 M
Cyclone IV E Family 279 M
Cyclone IV GX Family 200 M
Cyclone V Family 258 M
Legacy Families 1.8 M
MAX II Family 5.1 M
MAX V Family 5.9 M
Nios II SBT / Nios II GCC4 Toolchain 996 M
Description
Altera Installer 11.1
Space Required: 5.6 G
Space Available: 76 G
Select/Deselect All
< Back
Next >
Cancel

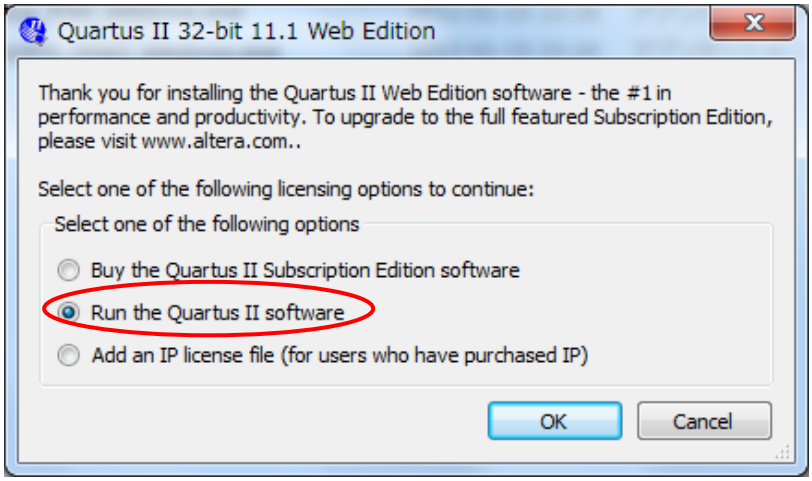


インストール概要を確認のうえ、「Next」をクリックしてインストールを開始します。
時間が掛りますので、コーヒーを飲みながら、暫くお待ちください。
最後、一時用ファイルを削除するかどうかを聞かれるダイアログが出て来る際、「yes」を選択し、下記インストール完了画面が出ます。

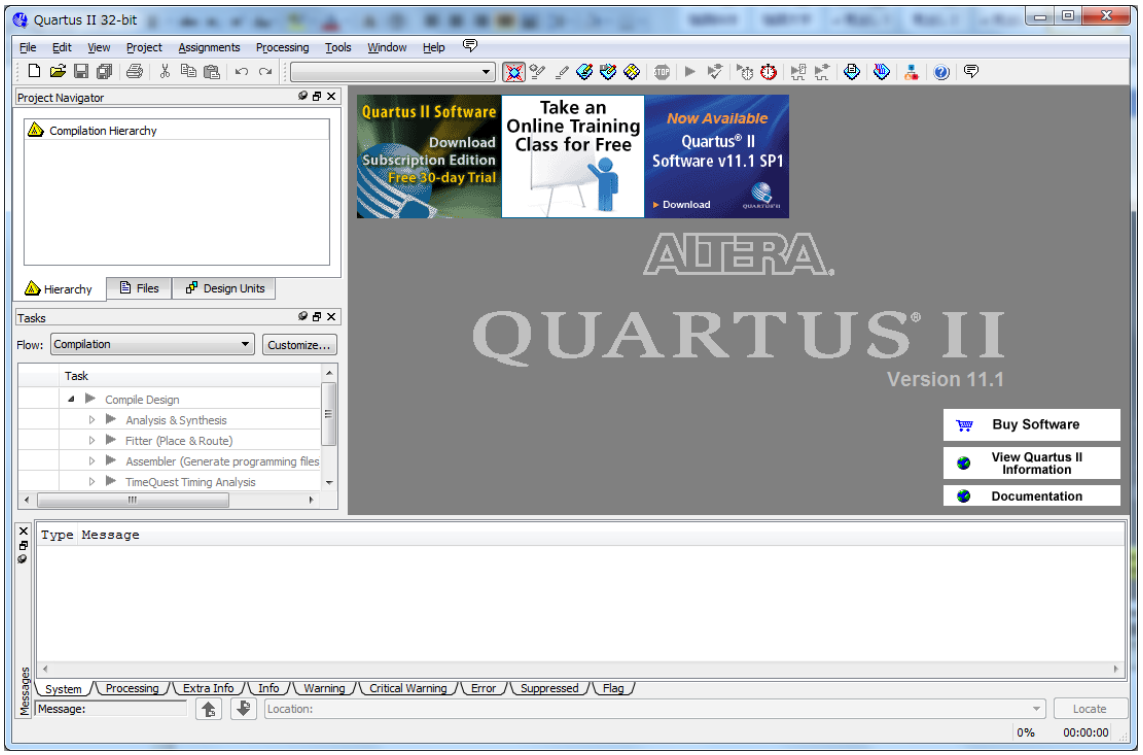


ライセンスを取得しない場合、「Get QuartusII Subscription License」を外して「Finish」を押しインストールが終わります。

インストールされた Quartus II 評価版をさっそく起動してみます。一番最初に起動したときだけ、次のようなダイアログが現れ、「**Run the Quartus II software**」を選択してください。「OK」ボタンを押します。

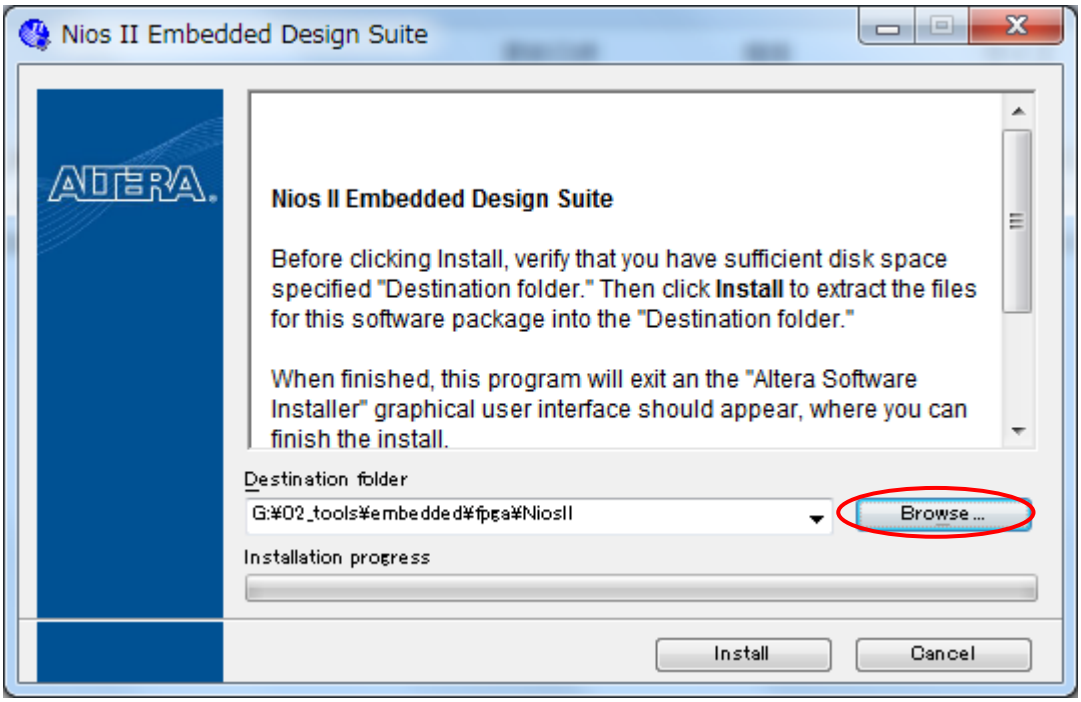


Quartus IIの画面出てきます。

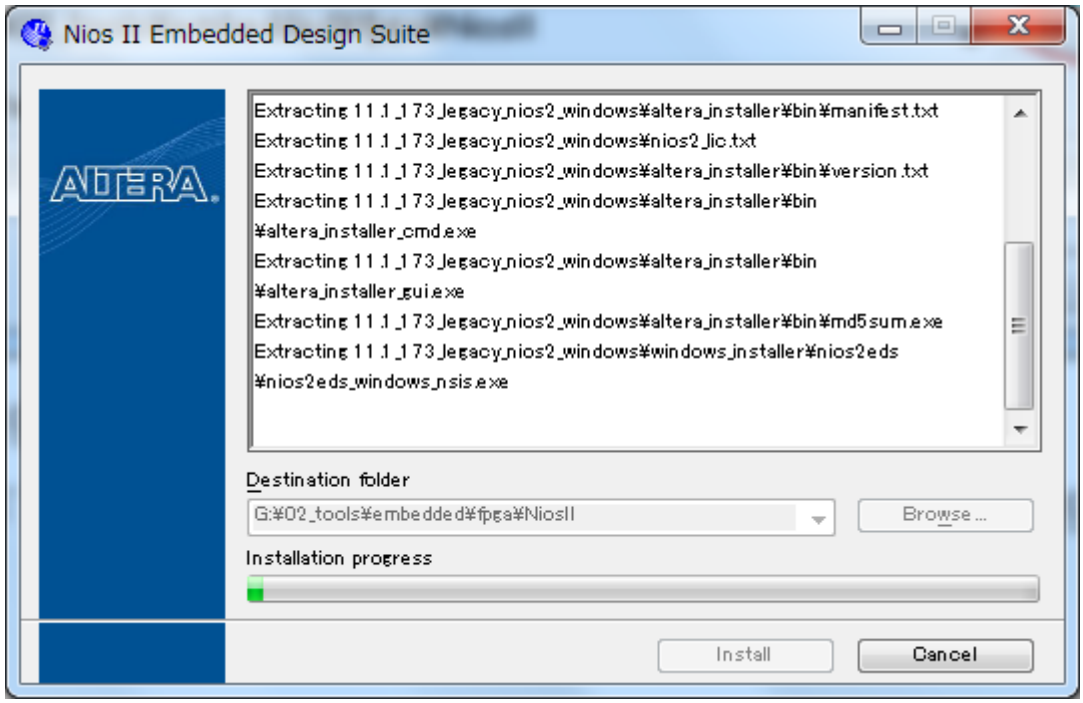


2.2 Nios II エンベデッド・デザイン・スイートをインストールする

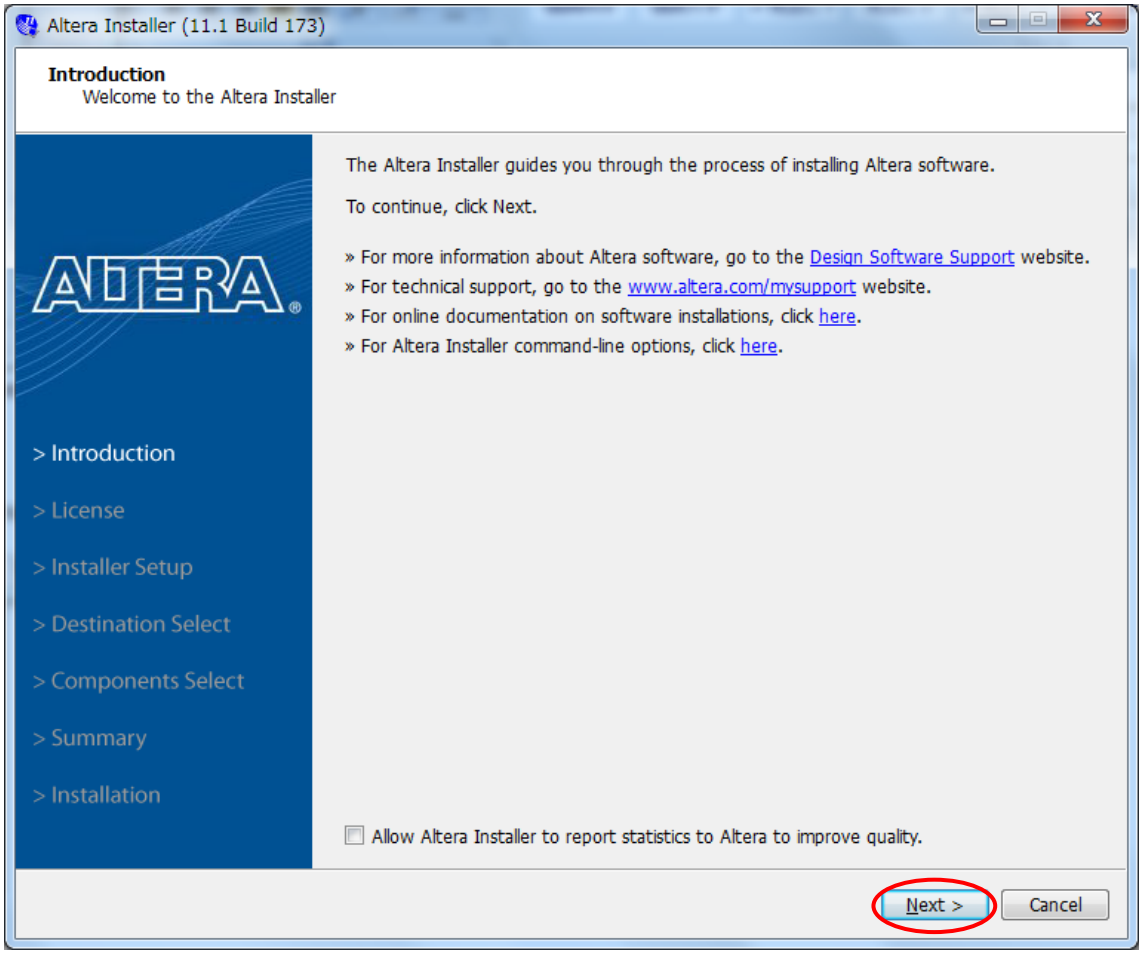
ダウンロードしたインストールファイルをダブクリック

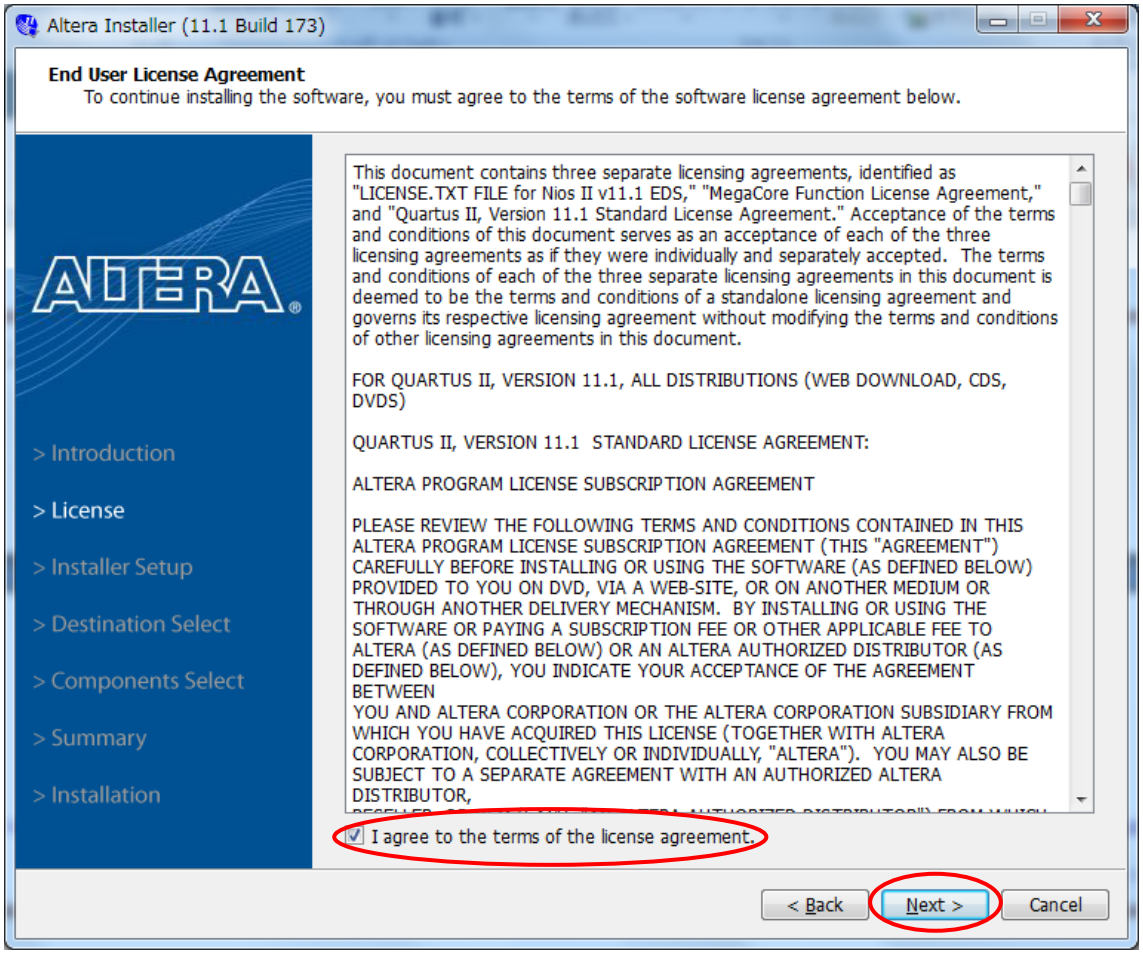


「Browse...」ボタンをクリックして解凍先を変更できます。
変更が終わったら、「Install」ボタンをクリックしてインストールを始めましょう。

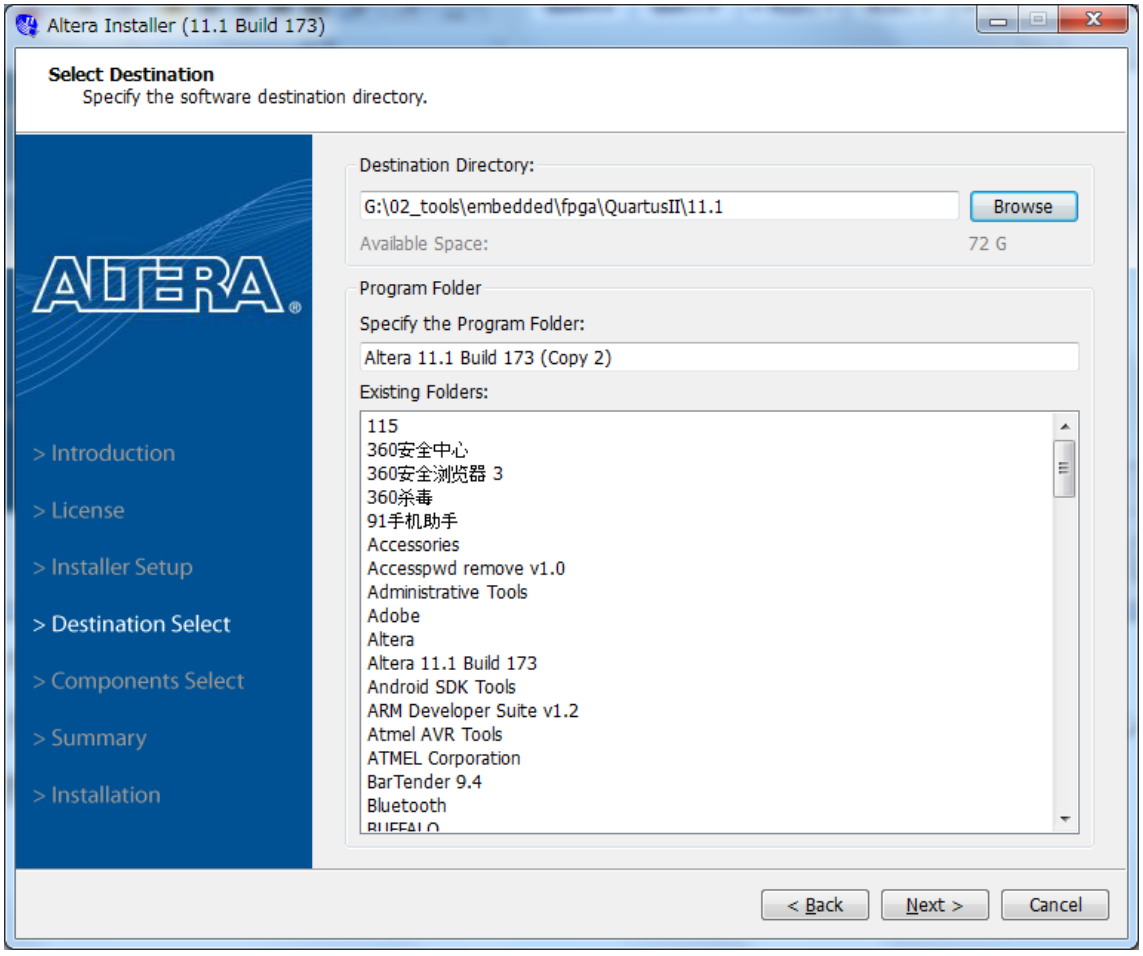


解凍完了後、正式のインストールに入ります。



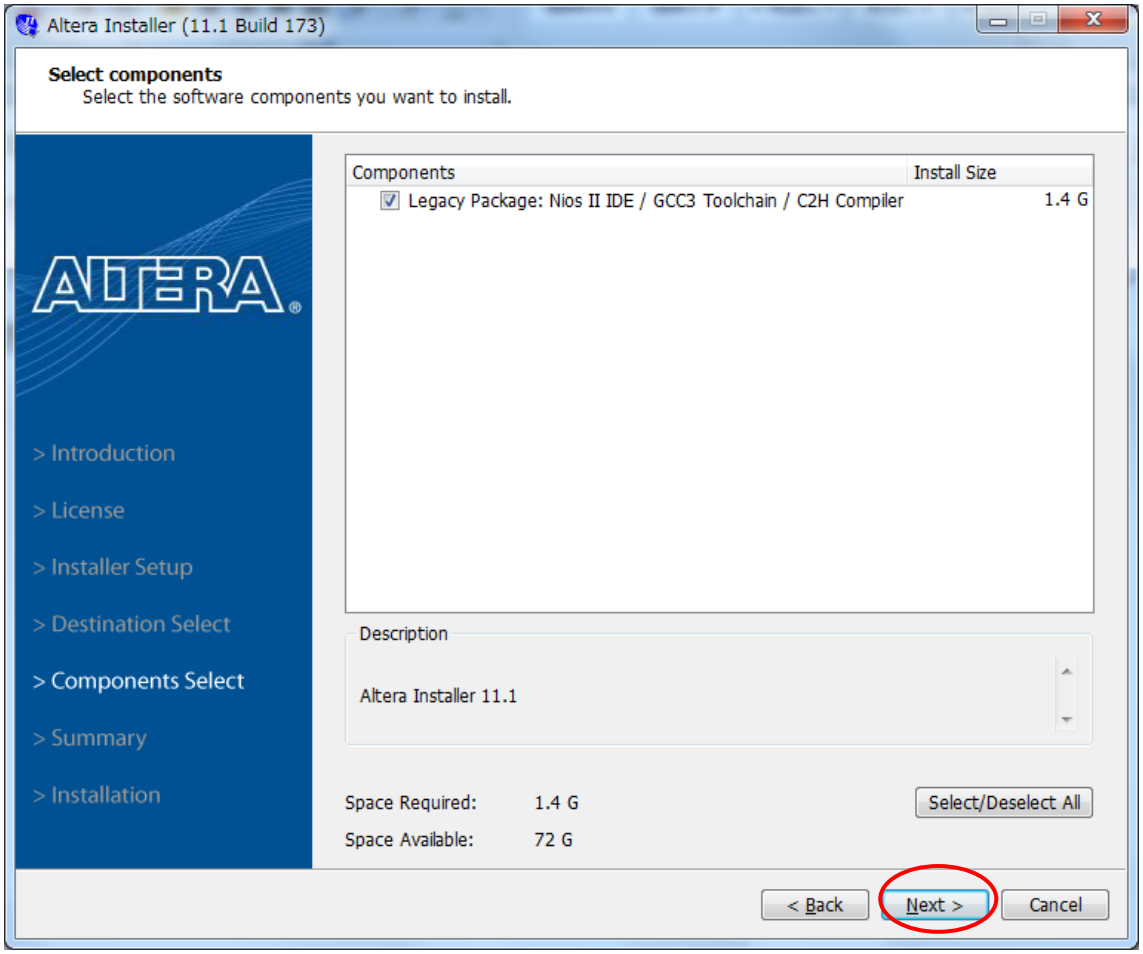


英文のライセンスが出てきます。同意できる場合は、「I accept the terms of the license agreement」を選択して、「Next」ボタンを押します。

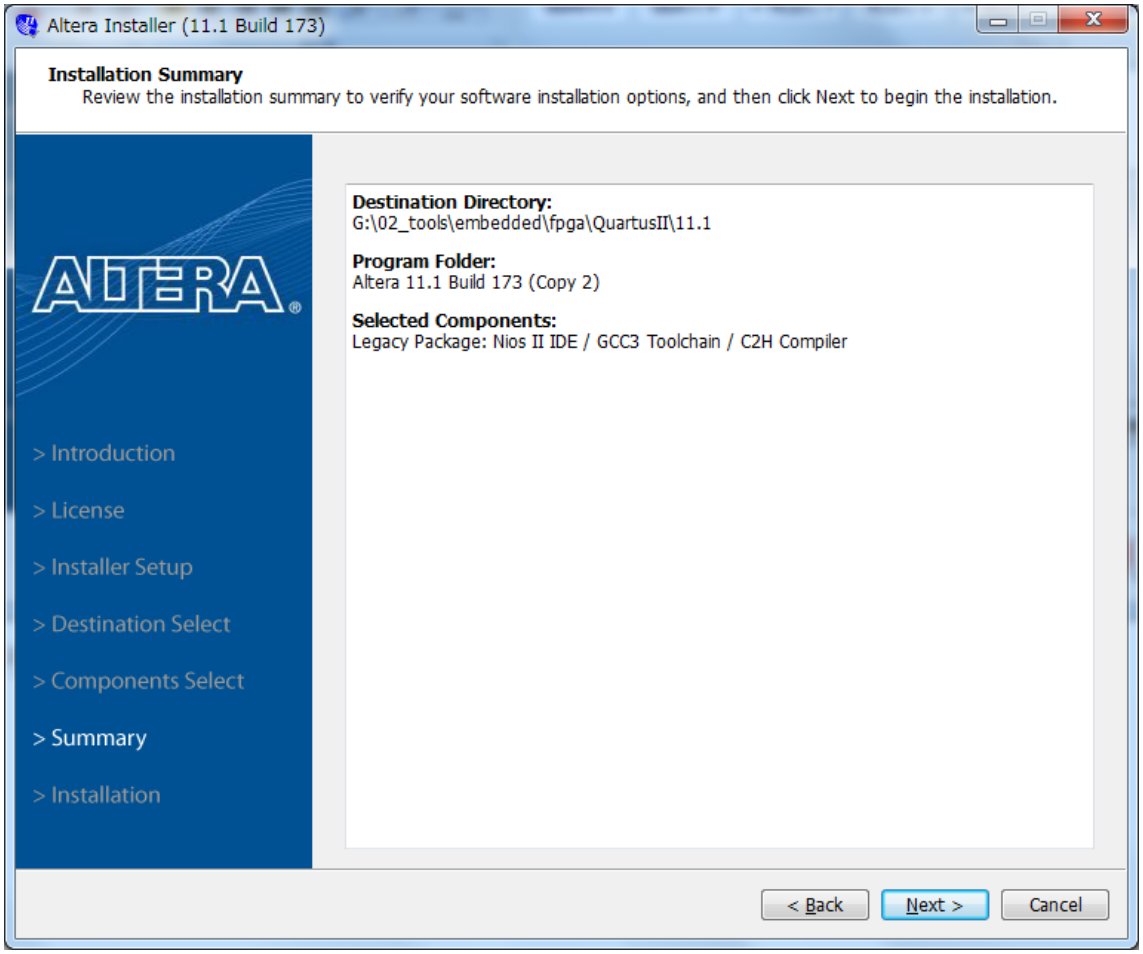


インストール先をQuatusIIのインストールフォルダ指定必要です。「Browse...」をクリックし変更してください。

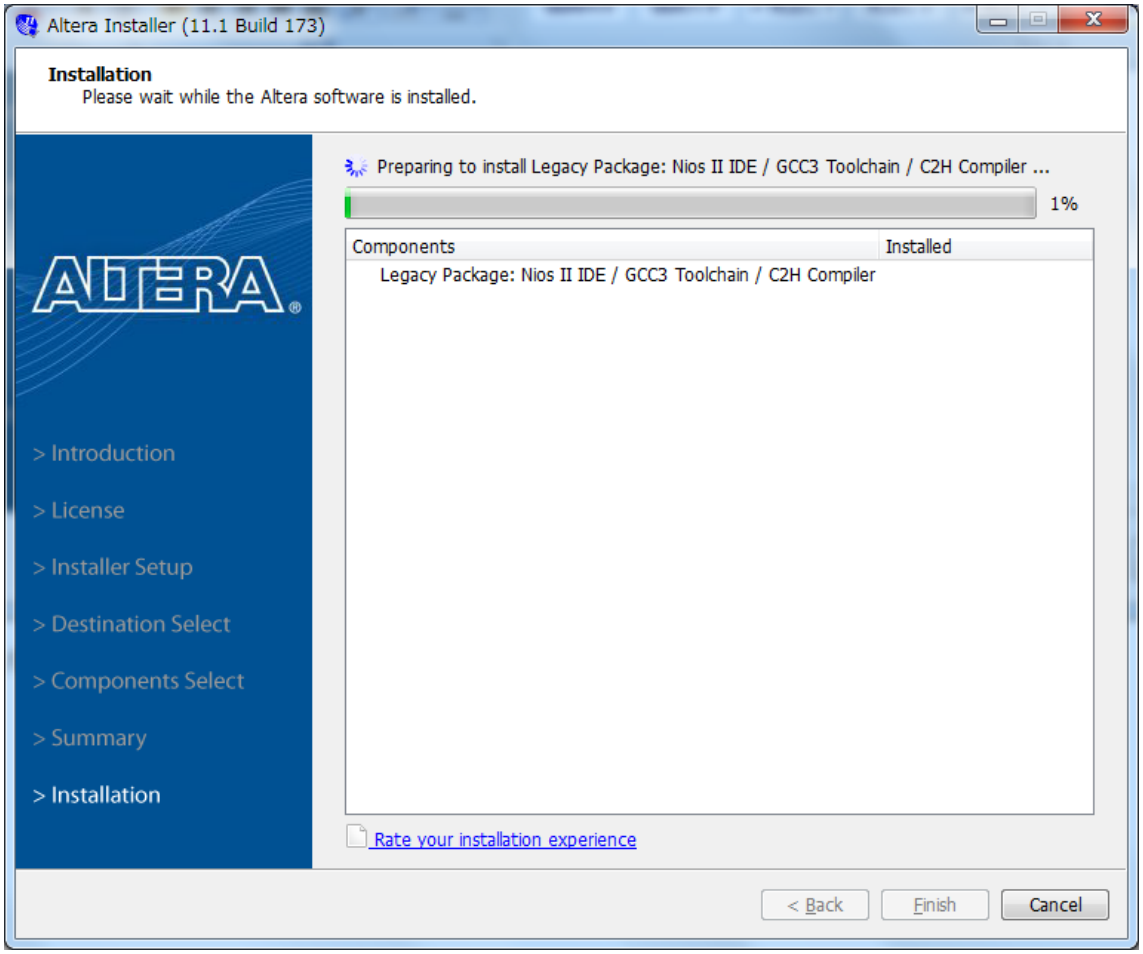
「Next」ボタンを押し、次のコンポーネント選択画面が出てきます。



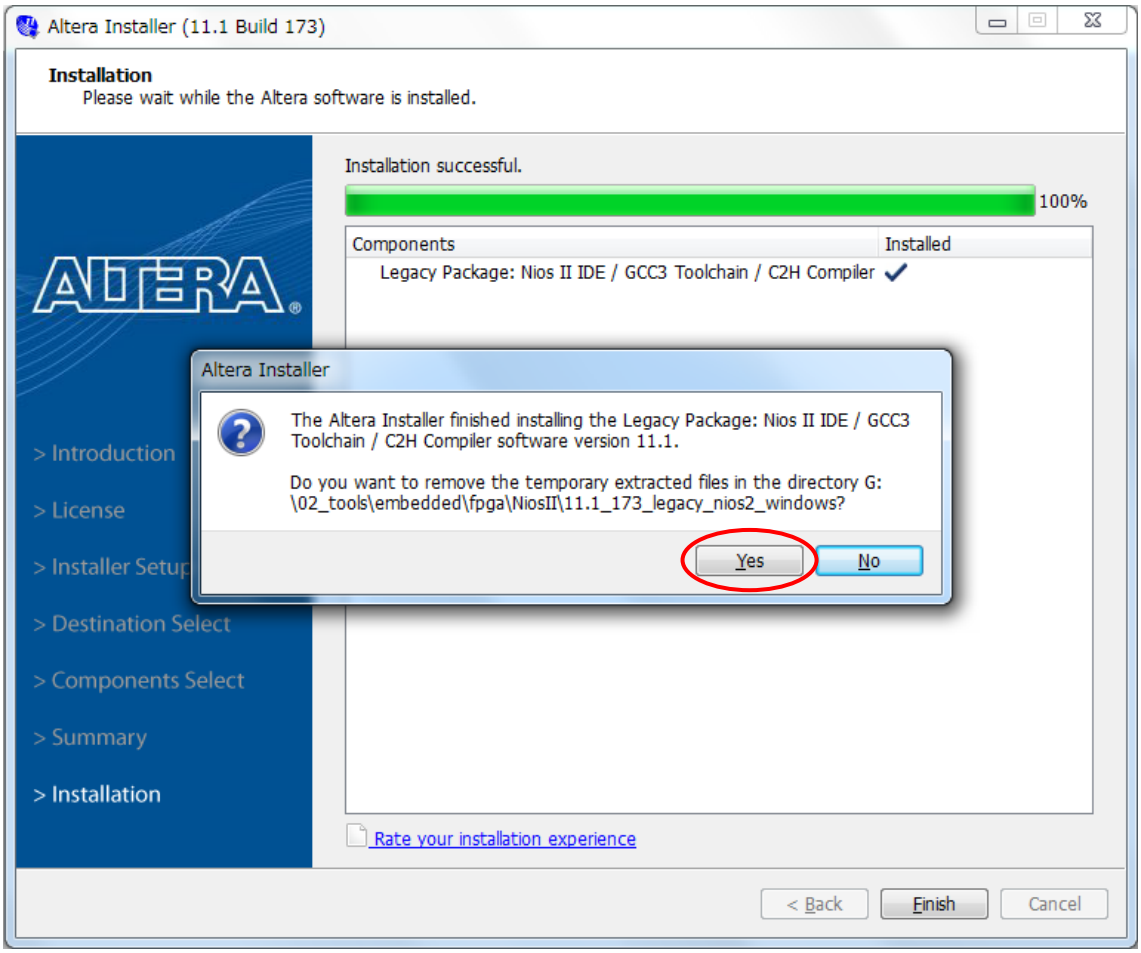
一つしかありませんので、そのまま「Next」を押しお進みください。



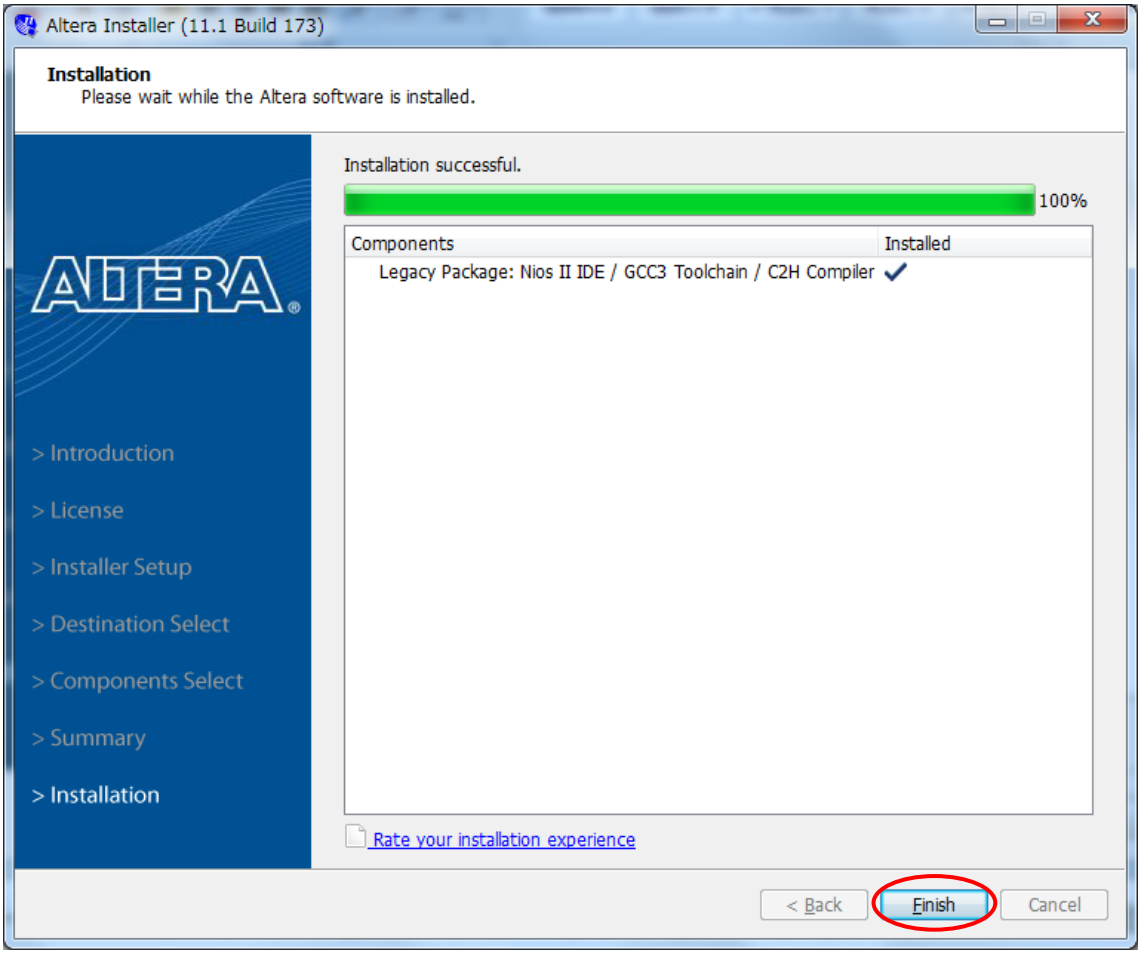
インストール概要を確認のうえ、「Next」をクリック



暫くお待ちください。100%になったら、下記の画面が出てきます。



インストールの一時ファイルを削除するかどうかを聞かれるダイアログが出ます、「Yes」をクリックして削除しましょう。

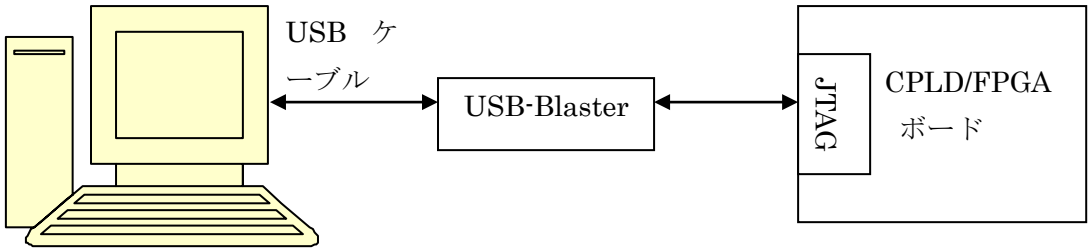


「Finish」ボタンを押し、インストールが終わりました。

ここまで必要のツールをすべてインストールしました、SP1をインストールする場合、もう一度実施してください。（QuartusIIとNiosIIのバージョンが同じ必要ですので、QuartusIIのSP1がインストールされた場合、必ずNiosIIのSP1もインストールしてください。）

2.3 USB-Blaster ドライバーをインストールする

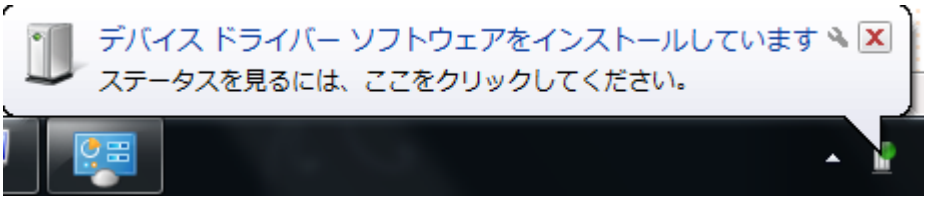
通常、Cyclone シリーズにコンフィグレーション・データを書き込むために、アルテラが発売している専用ダウンロード・ケーブル(ByteBlaster MV や ByteBlasterII や USB 接続タイプの USB-Blaster など)を購入しなければなりません。



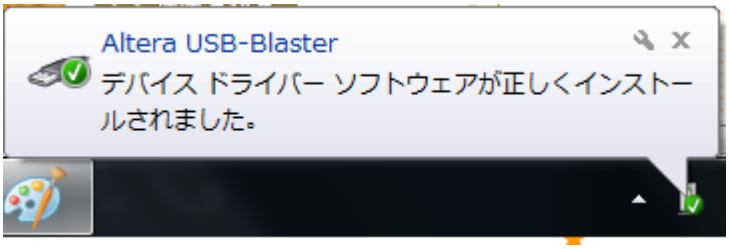
弊社は専用ダウンロード・ケーブル USB-Blaster 同等のデバイスを提供しております。
http://www.csun.co.jp/SHOP/200901025.html
次に示す手順に従って、USB-Blaster のデバイス・ドライバをインストールしてください。

■Win7 の場合：

USB-Blaster を USB ケーブルでパソコンと繋ぐと、Win7 の右下で自動的に下記の画面が現れ、ドライバーを自動的にインストールされます。



暫く待って、以下のインストール完了画面が出ます。



自動にうまくインストール出来ない場合、下記手順に従って手動でインストールしてください。

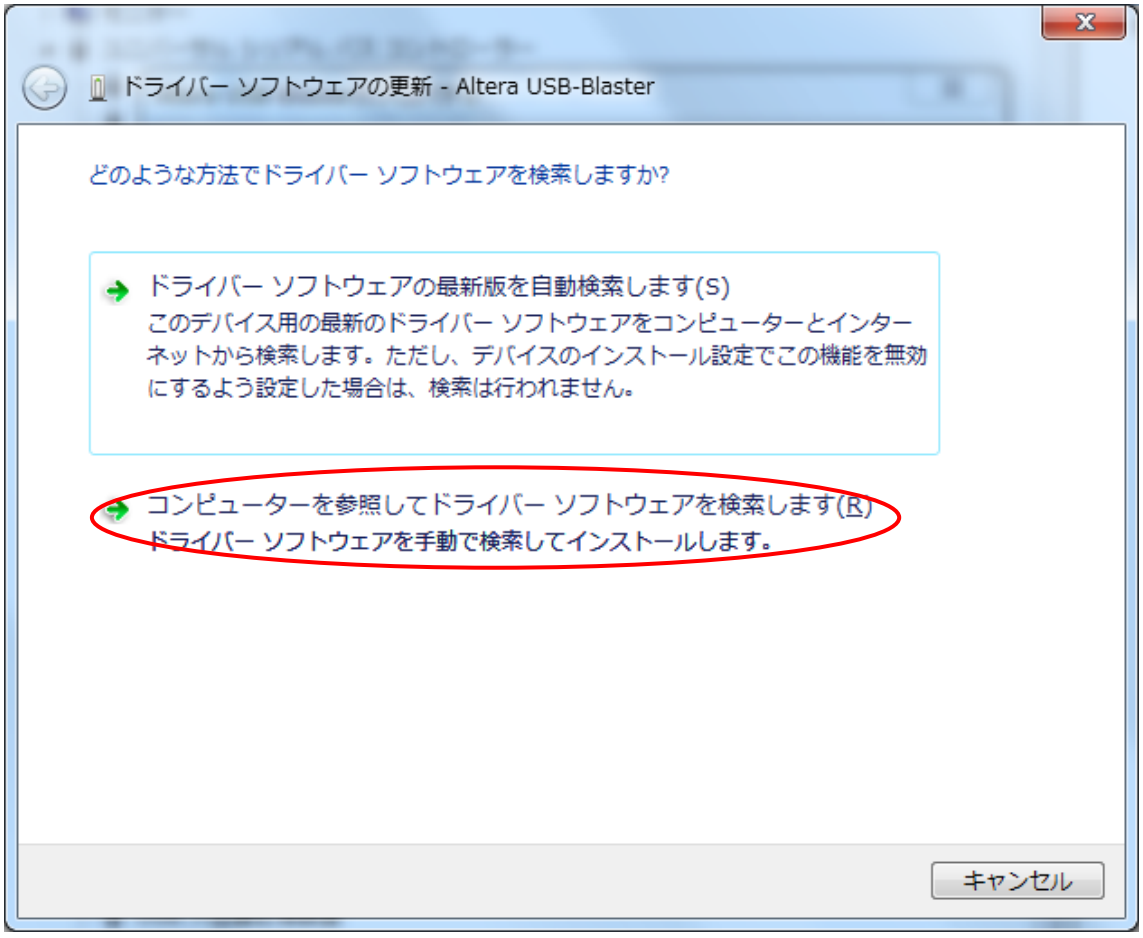
①デバイスマネージャーを開く

「スタートメニュー」―〉「コンピュータ」を右クリック―〉「管理」－＞

「デバイスマネージャー」を選択

②デバイスマネージャーの中、「ユニバーサルシリアルバスコントローラ」をクリック、正しくインストールされないデバイス（USB－Blasterの方）を選択し「プロパーティ」を右クリック

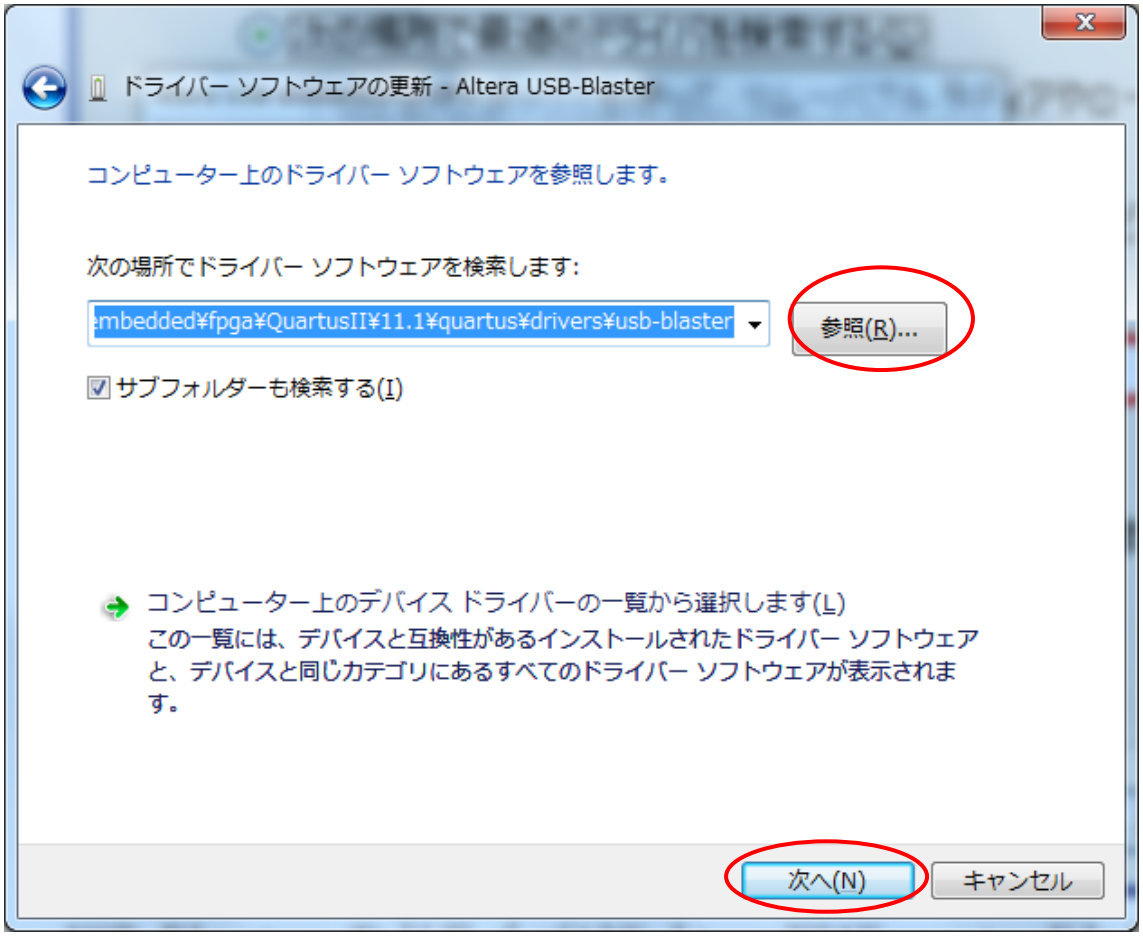
「ドライバー」タブを選択、「ドライバーの更新」をクリック



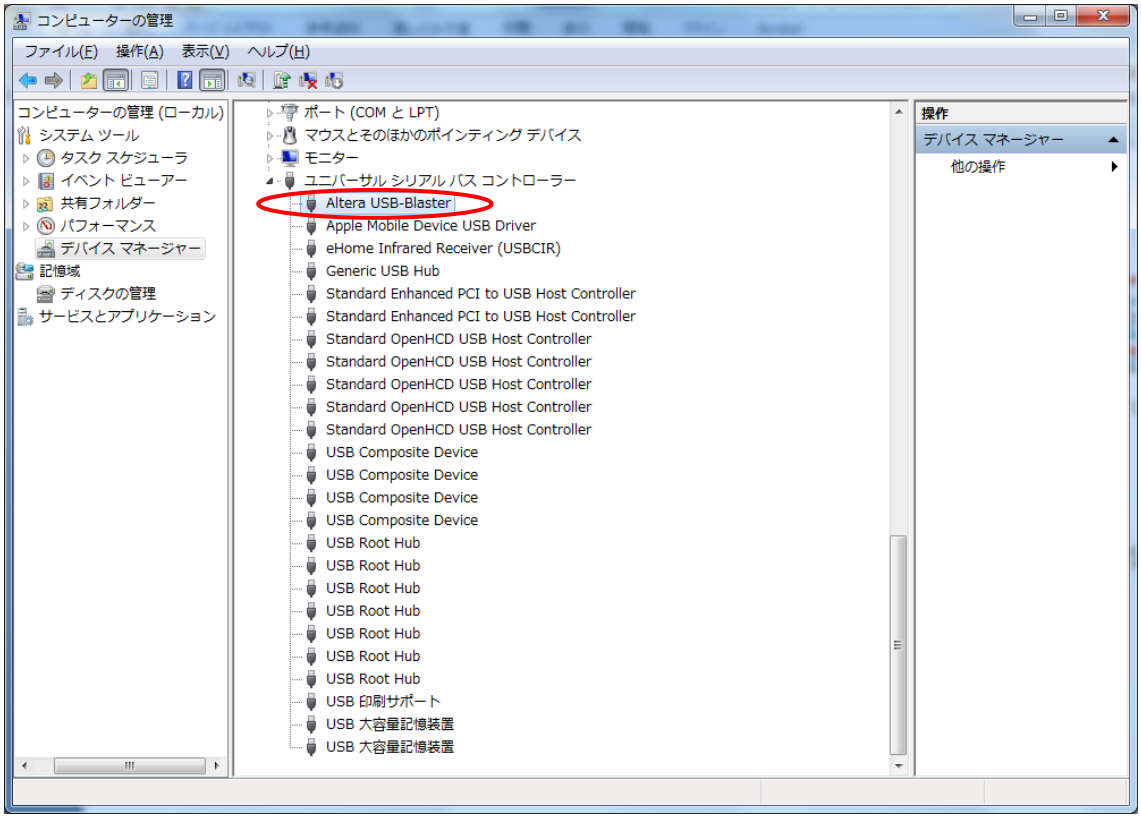
次の画面が出ます、この画面で「Browse」ボタンをクリック、QuartusIIインストール先を特定し（赤）、中のドライバーフォルダ（青）を指定してください。

「G:¥02_tools¥embedded¥fpga¥QuartusII¥11.1¥quartus¥drivers¥usb-blaster」

「次へ」ボタンをクリックすると、インストールを始めます。



そのまますぐ完了します、正常にインストールされた場合、デバイスマネージャーの中で下記の様子になります。



■WindowsXP の場合、インストール手順は以下のマニュアルの P32「3.2 USB-Blaster をインストールする」

Cyclone Ⅱ -EP2C20_manual.pdf

第三章 ハードウエア開発

第二章まで開発に必要なツール等を用意してきました、これから CycloneIV ボードに基づき詳しい NIOSII 開発を説明します（CycloneII ボードでも殆ど適用）。本章で Quartus II 11.1 を使って新しいプロジェクトを作成、NIOSII の IP コアを構築、コンパイル、ダウンロードします。ステップずつ画像と合わせて詳しく説明しますので、皆様は順調にハードウエア開発内容を完成できるだろう。

参考資料：

■オンライン資料： Quartus II 開発ソフトウェア

http://www.altera.co.jp/literature/lit-qts.jsp

■Quartus II HandBook（英語）

http://www.altera.co.jp/literature/hb/qts/quartusii_handbook.pdf

■Quartus II メニュー「help」→「Getting Started Tutorial」

■SOPC Builder User Guide（英語）

http://www.altera.co.jp/literature/ug/ug_sopc_builder.pdf

3.1 概要

NIOS 開発を行うため、幾つ条件必要です。

①SCM 知識

②C 言語知識

③Quartus II 開発の流れを分かる事

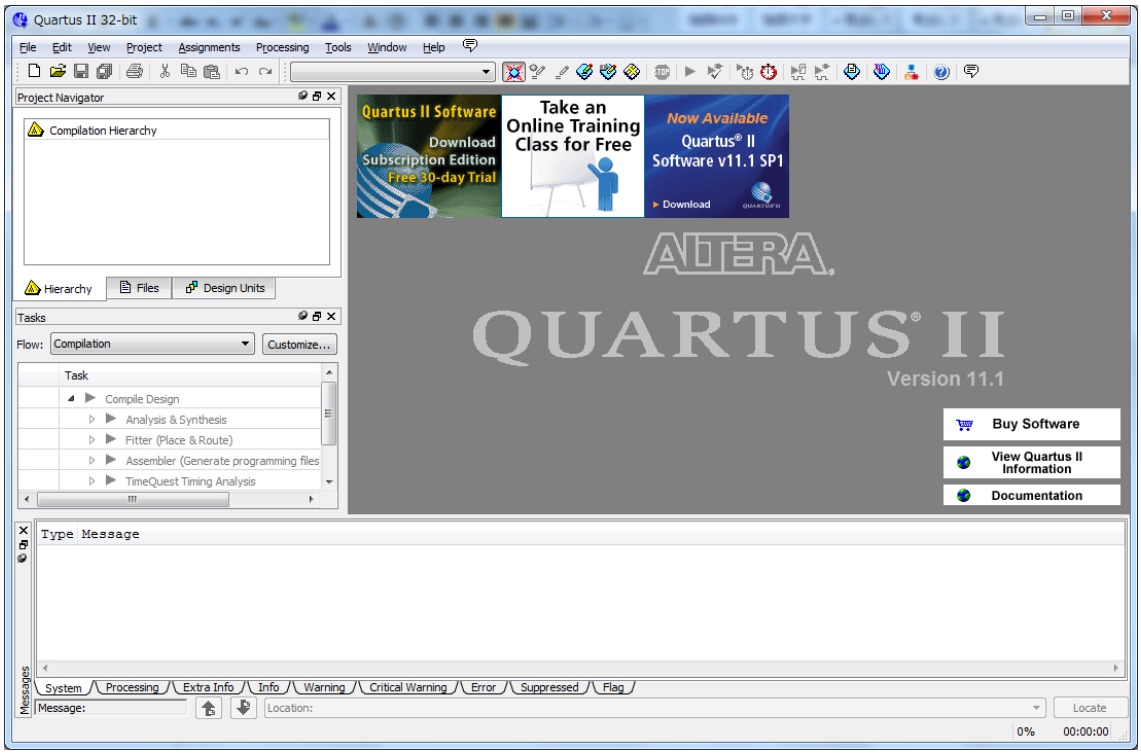
④FPGA ボード

※本マニュアルは弊社の CycloneIV ボードに基づき説明します。

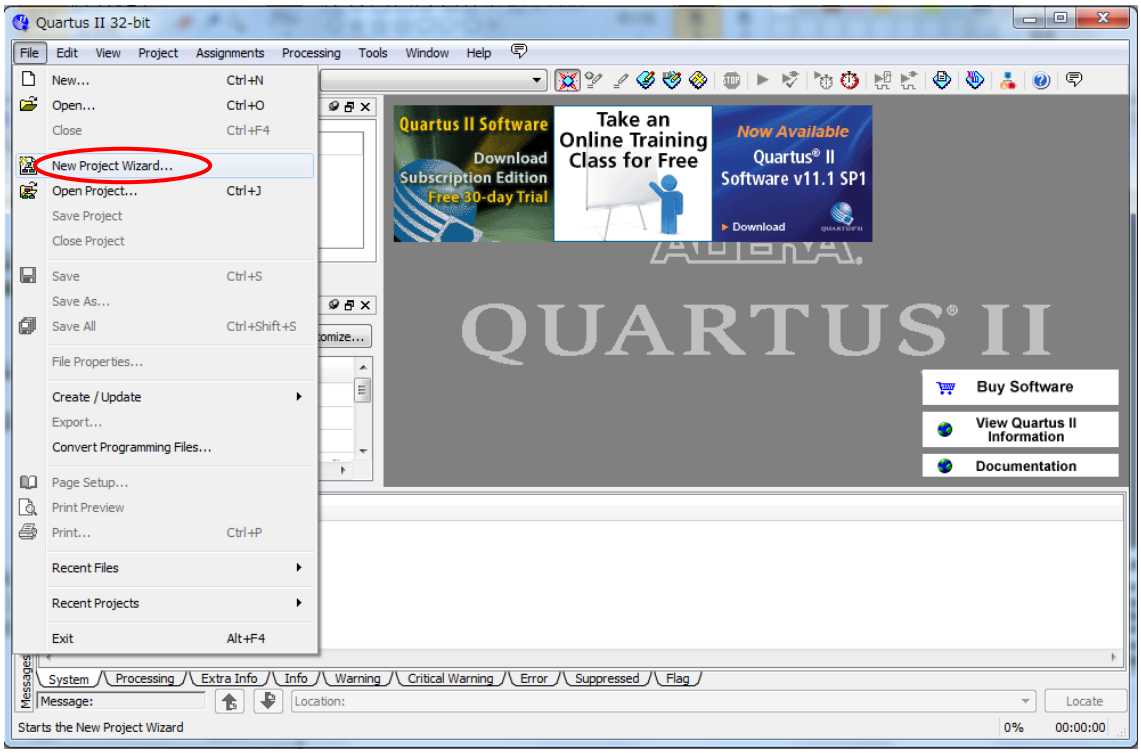
NIOS II はユーザが設定可能の 32 ビット RISC エンベデッドプロセッサです、これは SOPC（System On a Programmable Chip）のコアとなります。プロセッサはソフトマクロの形で実現されます、高度な柔軟性と設定拡張性を持ちます。NIOS II の開発はハードウエアとソフトウエアを二つ分けられます、ハードウエア開発は Quartus II で行われ、ソフトウエア開発は NIOS IDE で実施されます。まず、ハードウエア開発を説明しましょう、いわゆるハードウエア開発はニーズに応じ Quartus II と SOPC Builder で必要のソフトマクロを作ります。

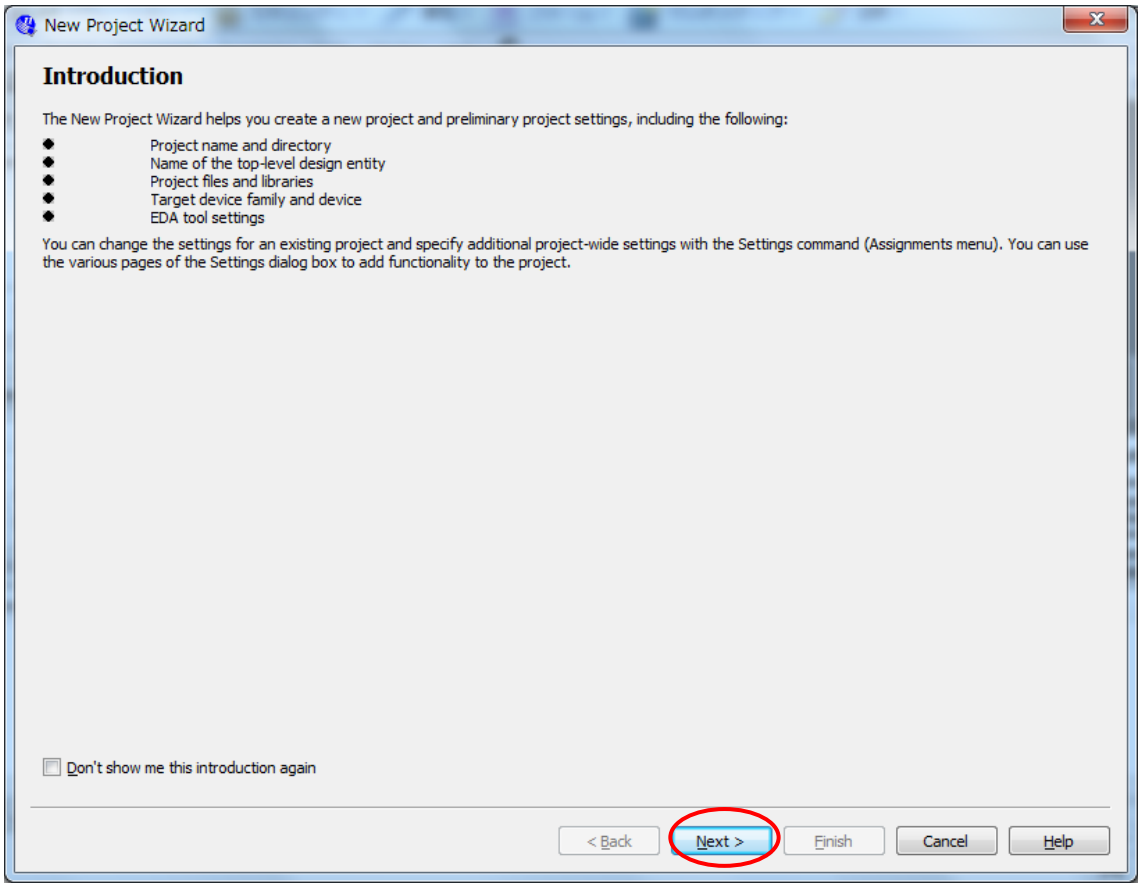
3.2 プロジェクト新規作成

①Quartus II 11.1 Web Edition を立ち上げ



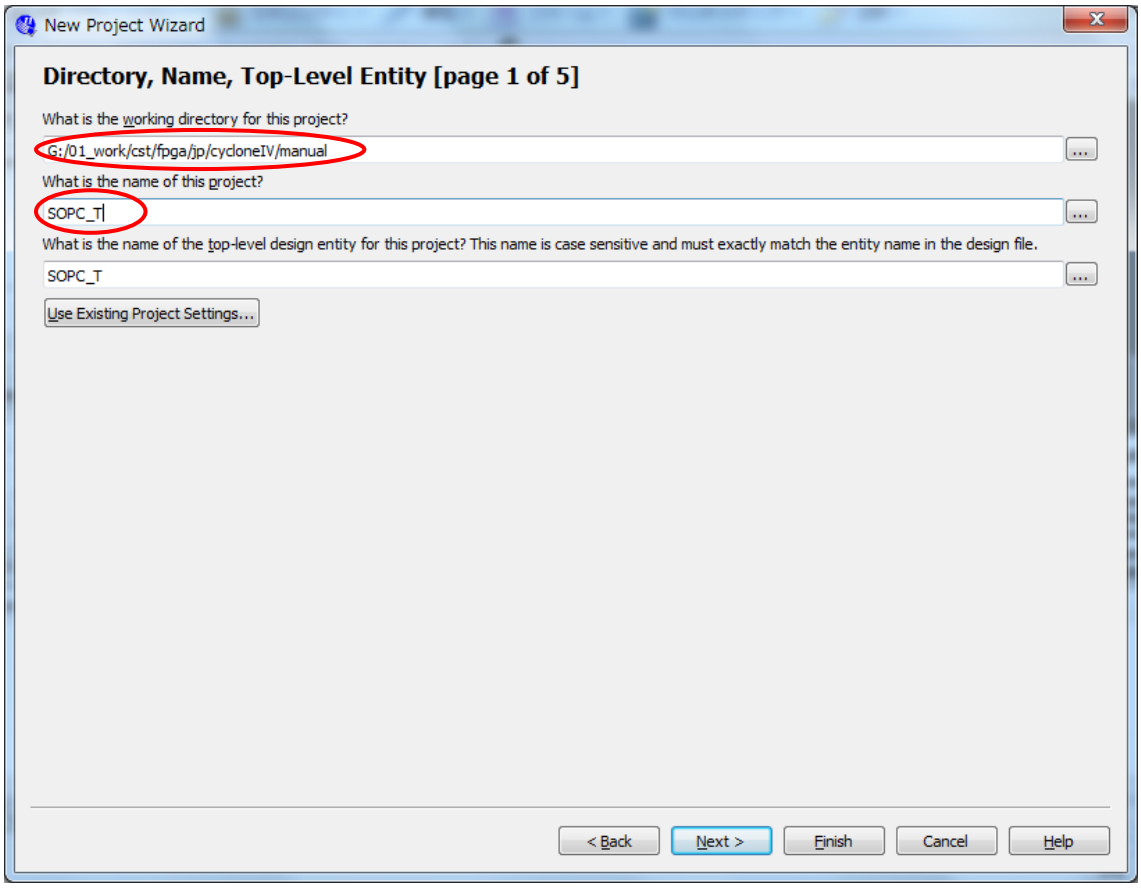
②File→New Project Wizard



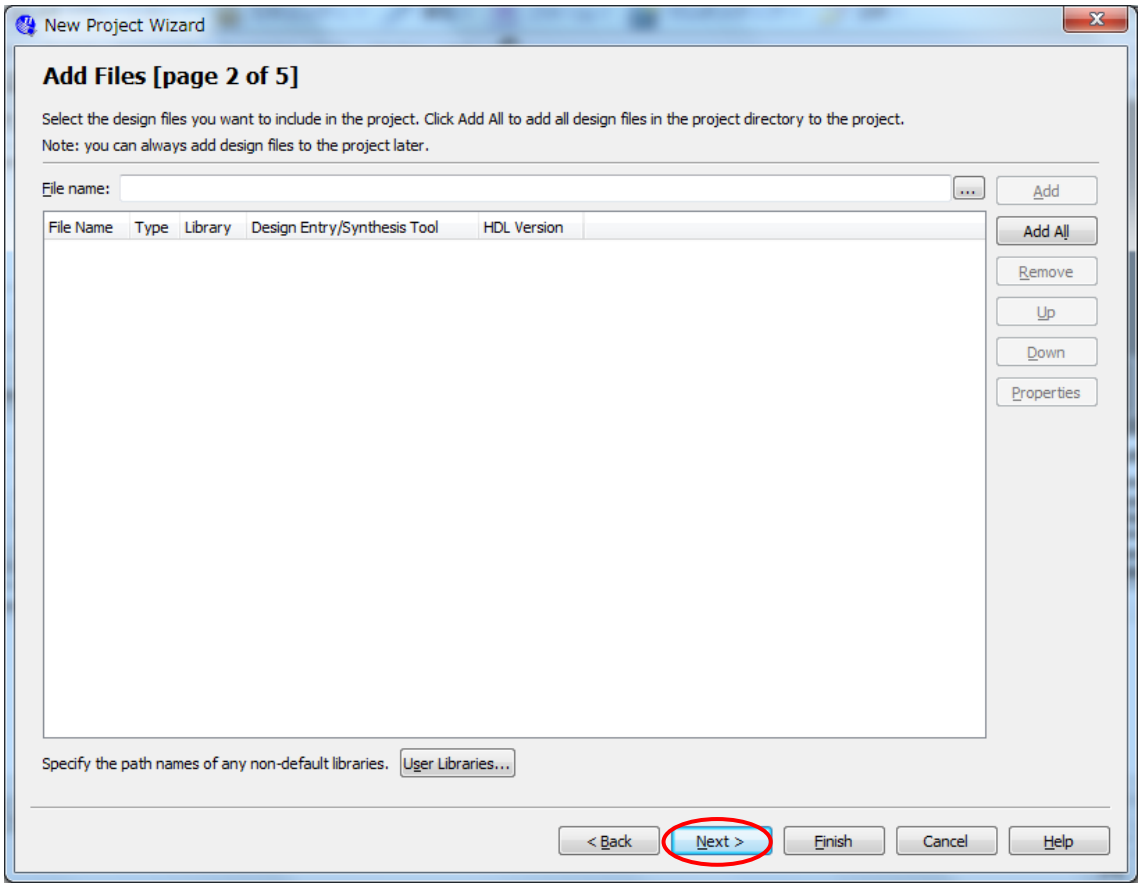


■プロジェクト保存場所を変更：「G:/01_work/cst/fpga/jp/cycloneIV/manual/example」
※お好きな場所を設定してください。

■プロジェクト名前を記入：「SOPC_T」



■ステップ２でそのまま「Next」ボタンをクリック



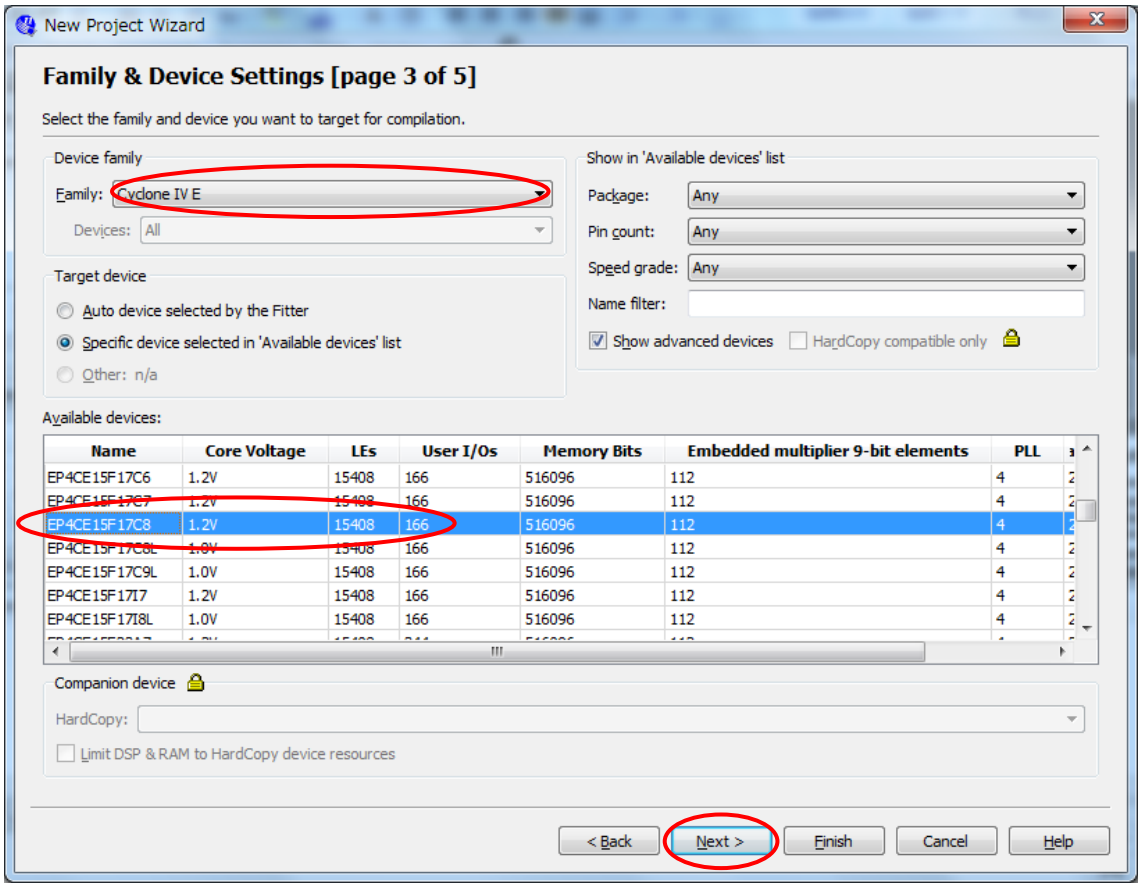
■Family：「CycloneIV E」を選べ

■Available devices：「EP4CE15F17C8」を選択

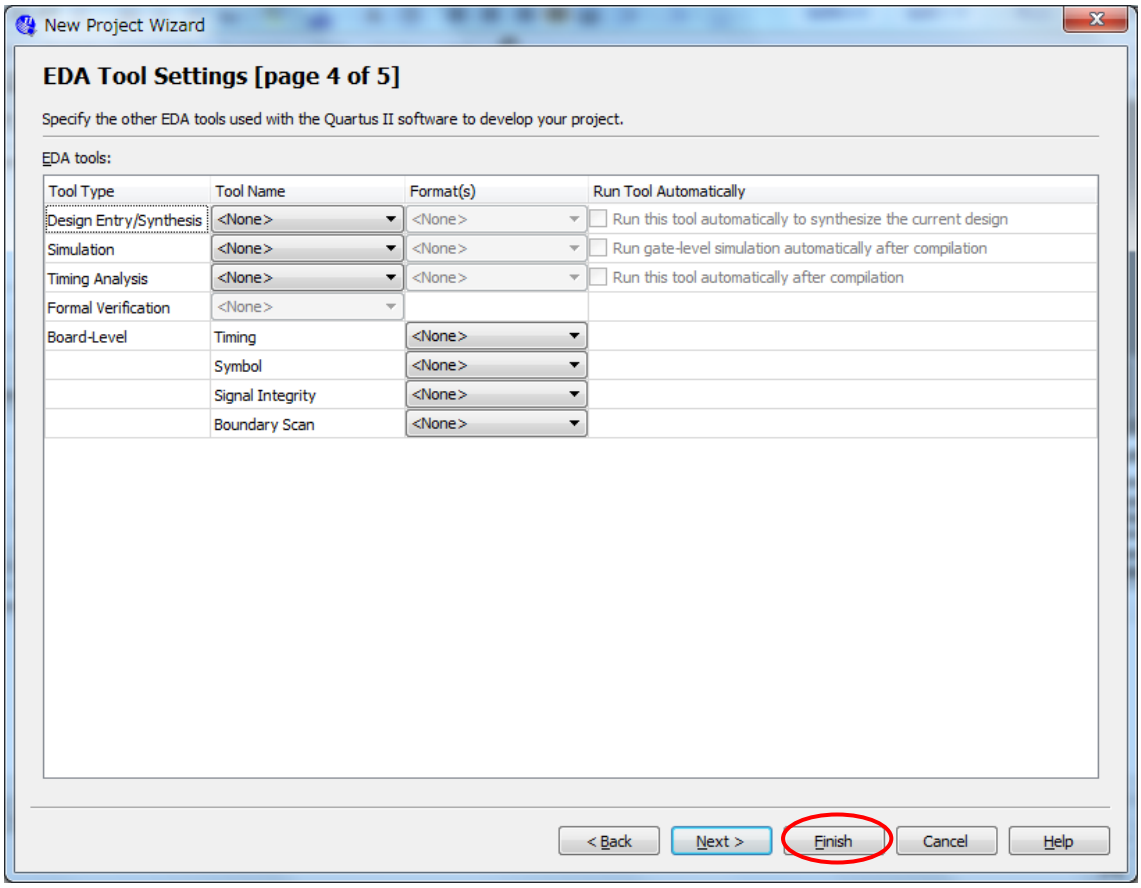
※上記内容は使われるボードにより適切なものを選択してください。

※CycloneII の場合、Family→「CycloneII」、Available devices→「EP2C8Q208C8」

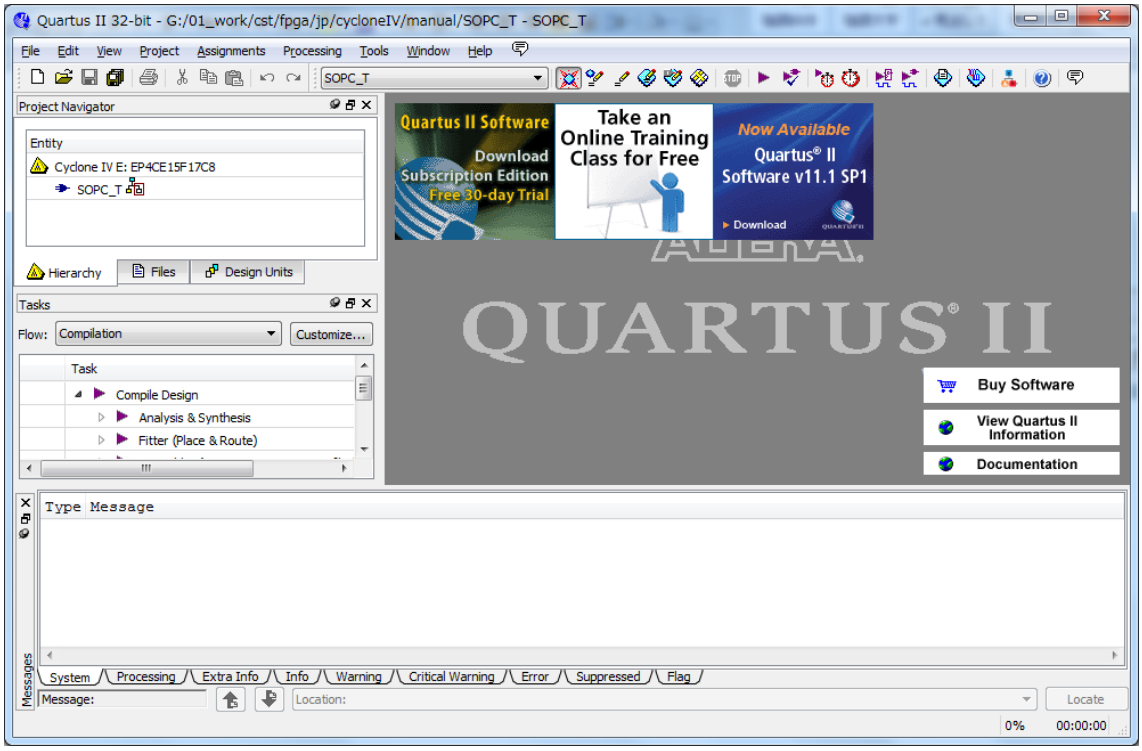
選択完了後、「Next」ボタンをクリック



■デフォルトのまま「Finish」ボタンをクリックして新規プロジェクト作成は完了させます。

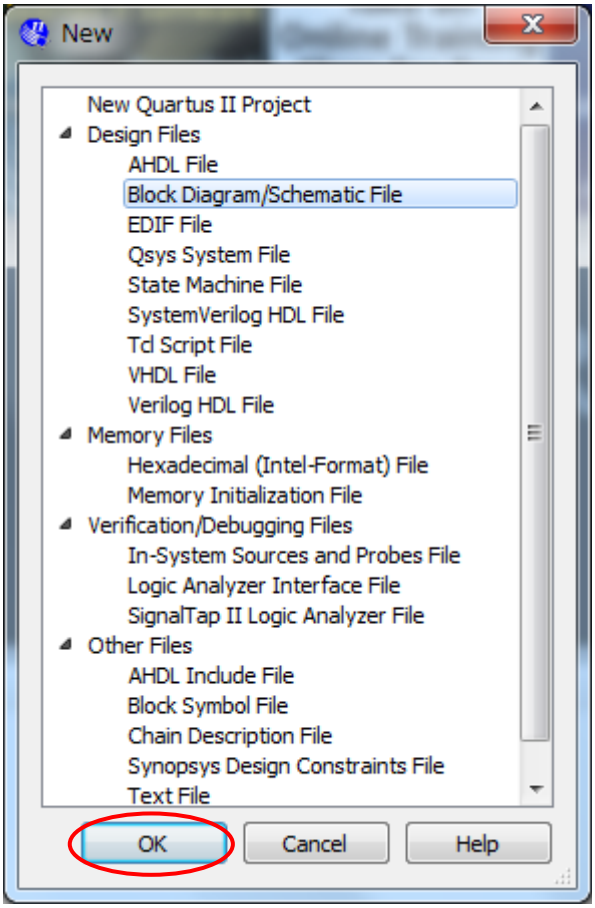

New Project Wizard
EDA Tool Settings [page 4 of 5]
Specify the other EDA tools used with the Quartus II software to develop your project.
EDA tools:
Tool Type
Tool Name
Format(s)
Run Tool Automatically
Design Entry/Synthesis
Simulation
Timing Analysis
Formal Verification
Board-Level
Timing
Symbol
Signal Integrity
Boundary Scan
<None>
Run this tool automatically to synthesize the current design
Run gate-level simulation automatically after compilation
Run this tool automatically after compilation
< Back
Next >
Finish
Cancel
Help

完了後の様子：

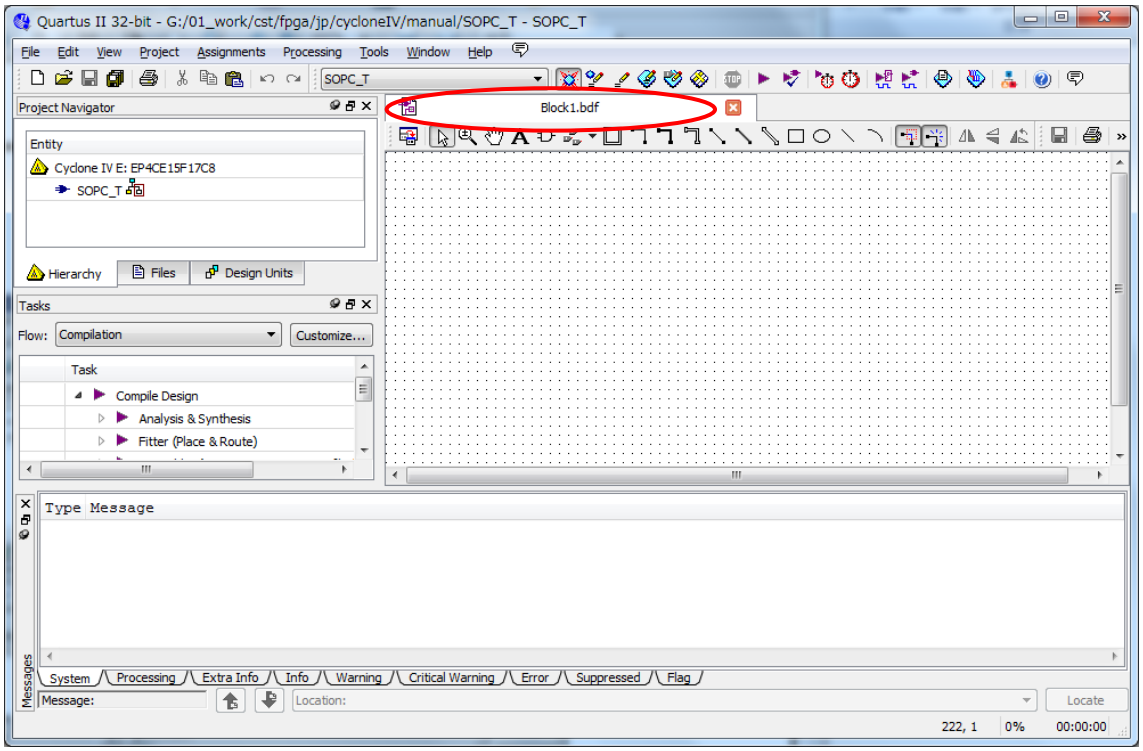

Quartus II 32-bit - G:/01_work/cst/fpga/jp/cycloneIV/manual/SOPC_T - SOPC_T
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Cyclone IV E: EP4CE15F17C8
SOPC_T
Hierarchy
Files
Design Units
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Quartus II Software
Download
Subscription Edition
Free 30-day Trial
Take an
Online Training
Class for Free
Now Available
Quartus® II
Software v11.1 SP1
Download
QUARTUS II
Buy Software
View Quartus II
Information
Documentation
Type Message
System Processing Extra Info Info Warning Critical Warning Error Suppressed Flag
Message:
Location:
Locate
0%
00:00:00

③Block Diagram/Schematic File を作成、File->New

「Block Diagram/Schematic File」を選択、「Ok」ボタンをクリック



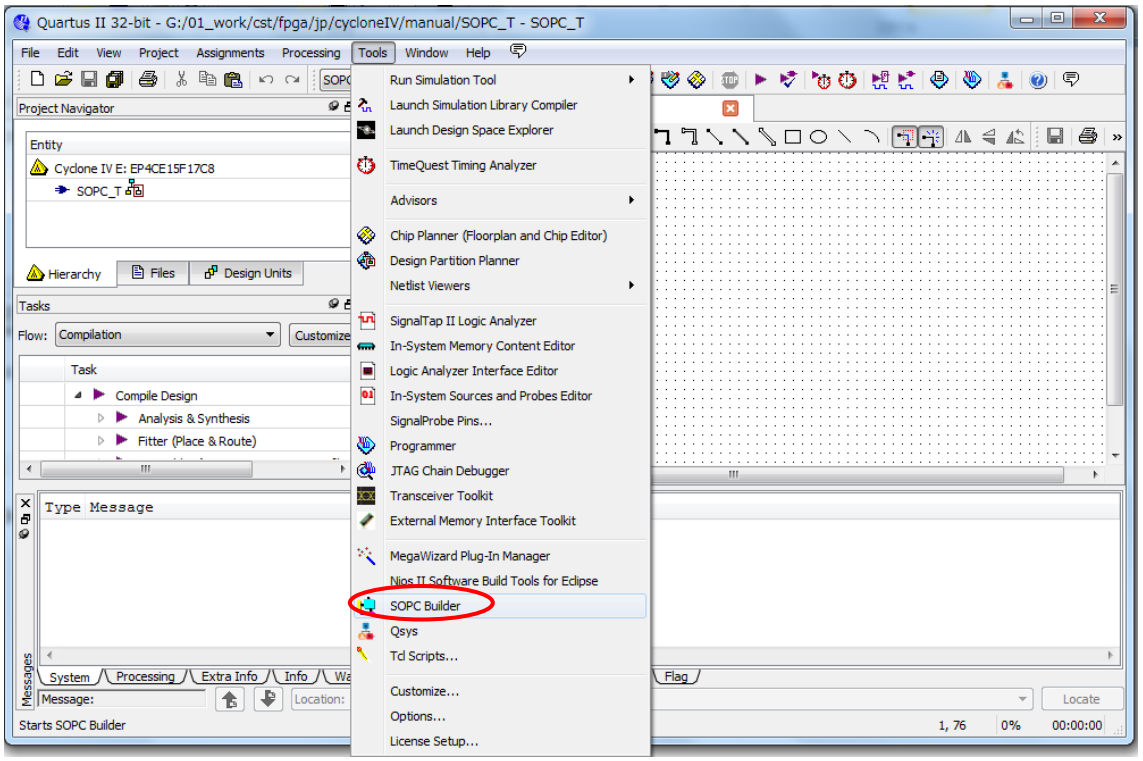
プロジェクトの中、「Block1.bdf」ファイルが出来ます。



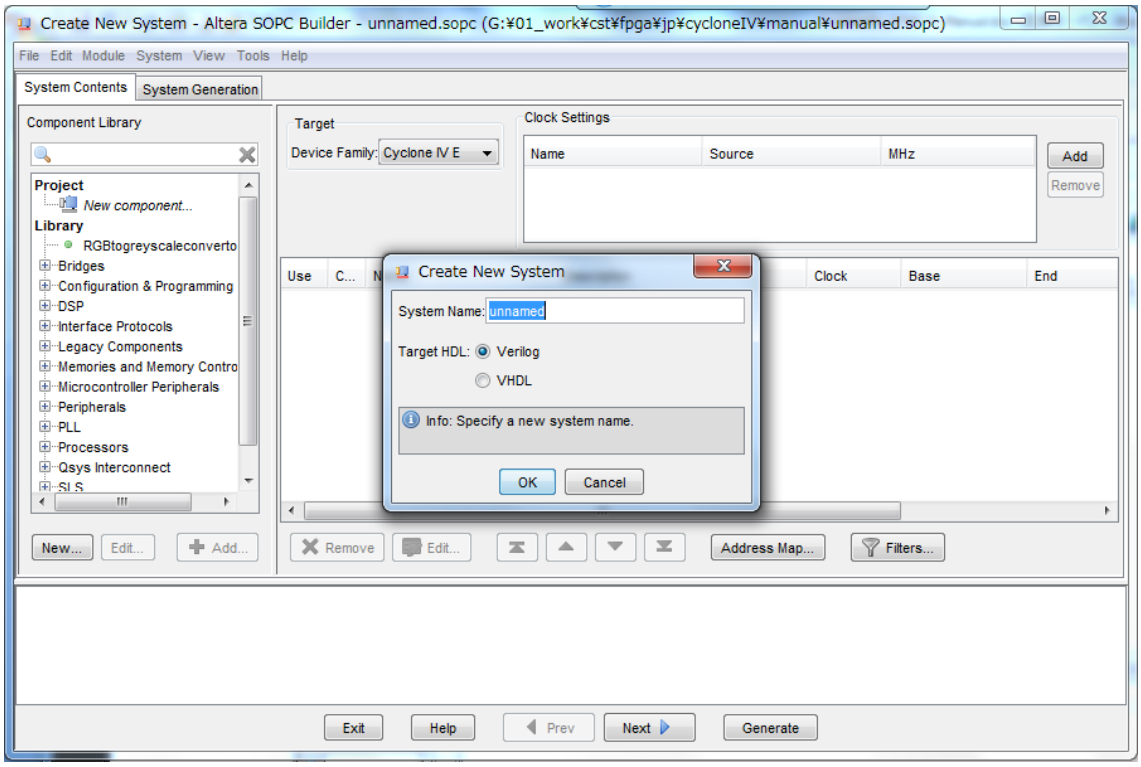
3.3 NIOS II ソフトマクロを作成

3.3.1 SPOC Builder 起動

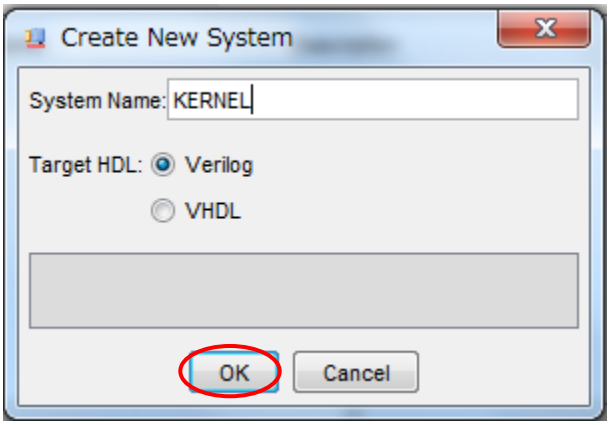
Tools->SOPC Builder



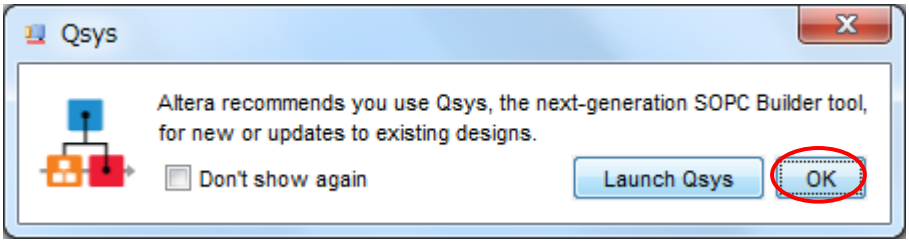
下記の画面が出ます。



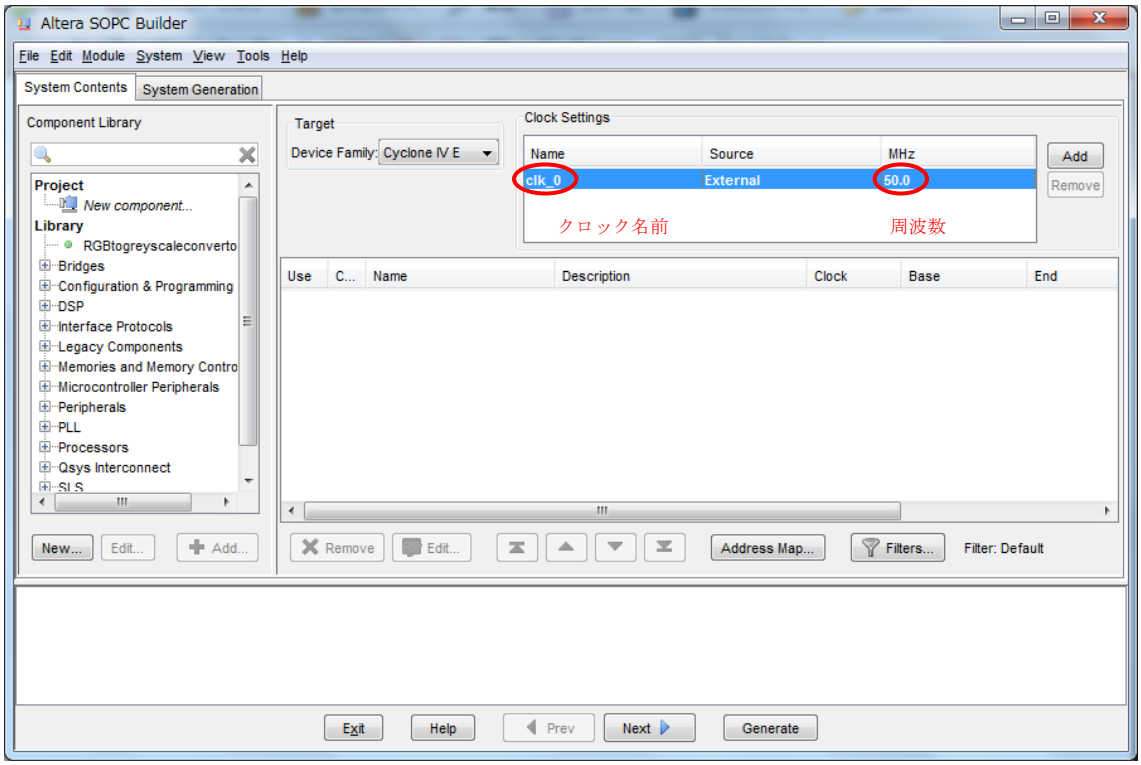
System Name に「KERNEL」を入力、「OK」ボタンをクリック



途中、以下のダイアログが出ますが、そのまま「OK」ボタンを押下



SOPC Builder 画面：

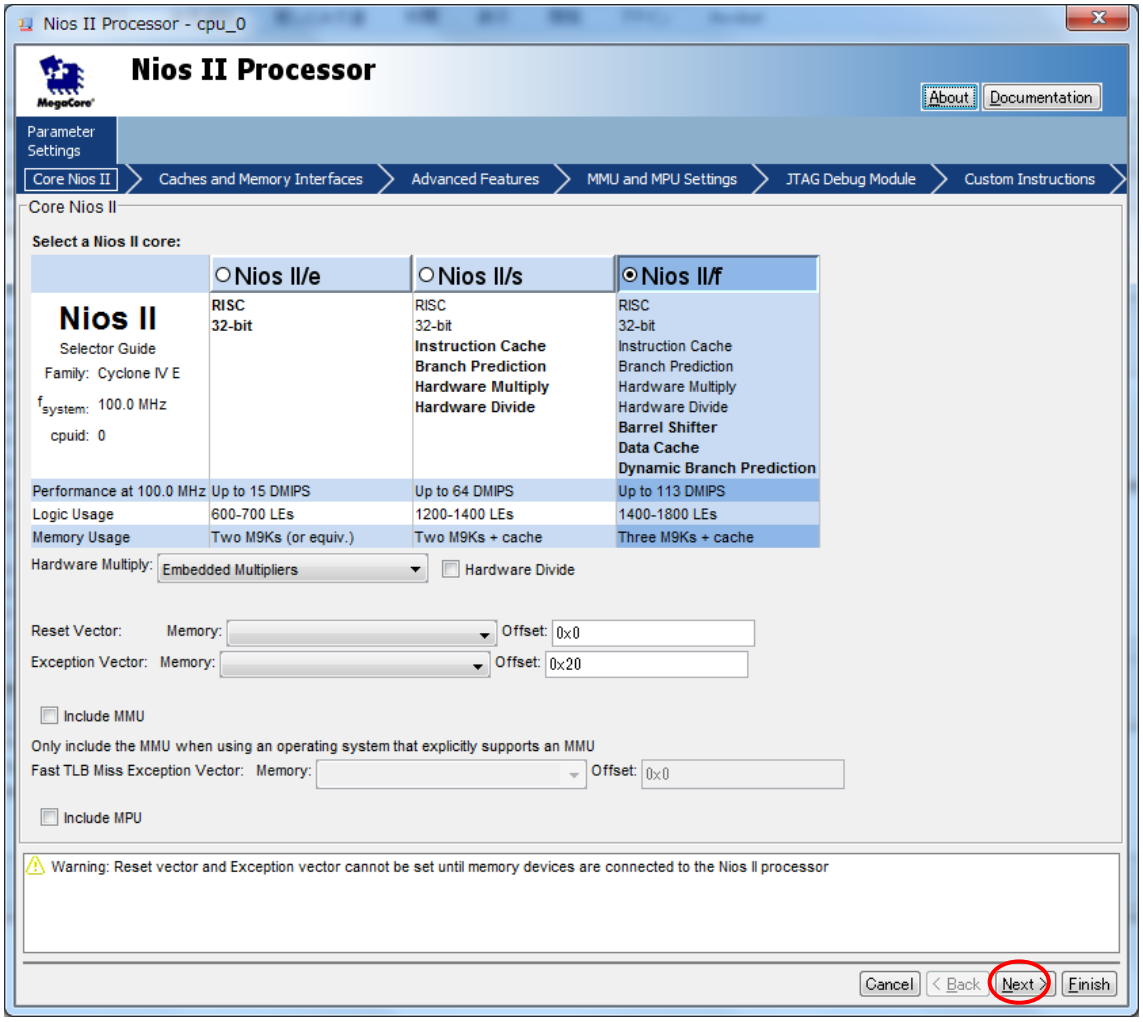


上図のように「clk_0」がクロック名前、「50.0」が周波数（単位：MHZ)。周波数を修正可能です、「50.0」をダブルクリックして「100.0」に修正しましょう。そうすると、ソフトマクロの周波数は 100MHZ になります。

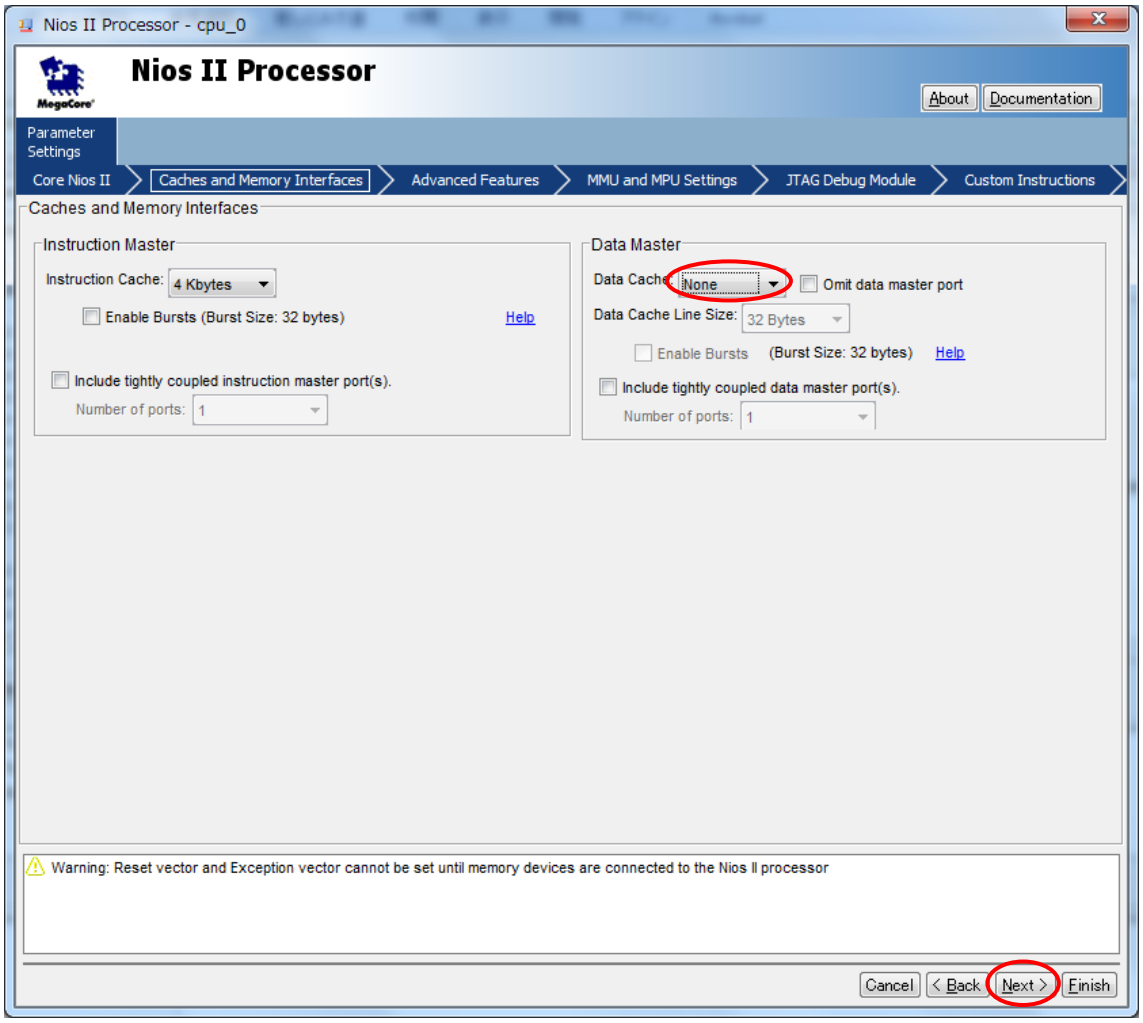
3.3.2 CPU プロセッサを作成

左側の「Processors」をクリック、「NIOS II Processor」が現れます。ダブルクリックすると、下図が表示されます。三種類プロセッサ：1)Nios II/e 2)Nios II/s 3)Nios II/f があります、Nios II/e がリソースの使用が一番少ないが、機能が少ない、スピードも一番遅い。Nios II/f がリソースの使用が一番多く、機能が一番多く、スピードも一番早いです。Nios II/s が e と f の間の位置を付けされます。何の種類のプロセッサを使うかにはニーズに応じチップのリソースにより決められることになります、ここ、Nios II/f を選べ、機能とスピードが両方も満たします。下の Reset Vector は起動後のメモリ種類とオフセットです、Exception Vector は異常の時にメモリ種類とオフセットです、SDRAM と Flash を設定出来た後、これらの設定を修正できます、現時点修正できませんが、とても重要の設定です。

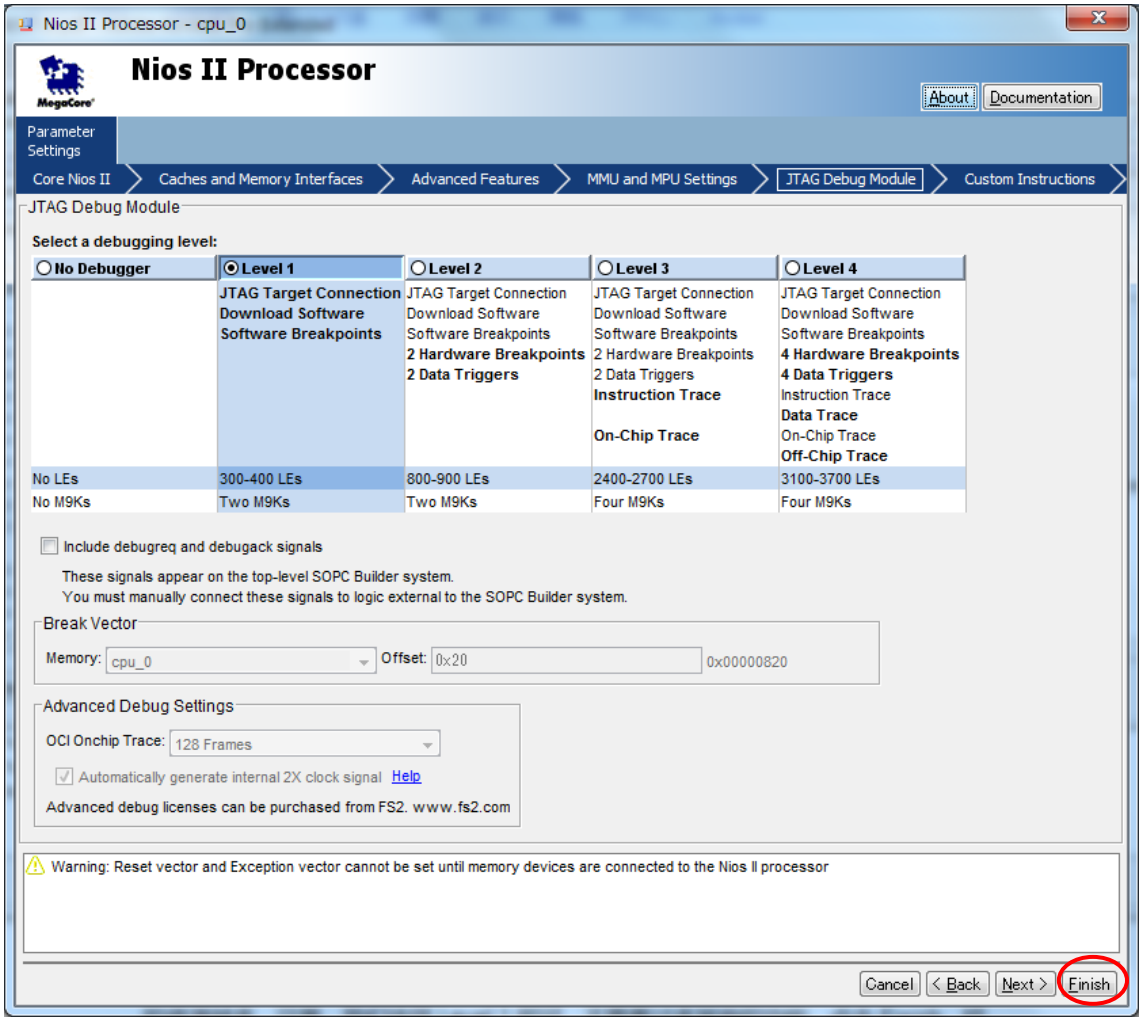
「Next」ボタンをクリック、次の設定に入ります。


Nios II Processor - cpu_0
Nios II Processor
About
Documentation
Parameter Settings
Core Nios II
Caches and Memory Interfaces
Advanced Features
MMU and MPU Settings
JTAG Debug Module
Custom Instructions
Core Nios II
Select a Nios II core:
Nios II/e
Nios II/s
Nios II/f
Nios II
Selector Guide
Family: Cyclone IV E
f system: 100.0 MHz
cpuid: 0
RISC
32-bit
Instruction Cache
Branch Prediction
Hardware Multiply
Hardware Divide
Barrel Shifter
Data Cache
Dynamic Branch Prediction
Performance at 100.0 MHz
Up to 15 DMIPS
Up to 64 DMIPS
Up to 113 DMIPS
Logic Usage
600-700 LEs
1200-1400 LEs
1400-1800 LEs
Memory Usage
Two M9Ks (or equiv.)
Two M9Ks + cache
Three M9Ks + cache
Hardware Multiply:
Embedded Multipliers
Hardware Divide
Reset Vector:
Memory:
Offset:
0x0
Exception Vector:
Memory:
Offset:
0x20
Include MMU
Only include the MMU when using an operating system that explicitly supports an MMU
Fast TLB Miss Exception Vector:
Memory:
Offset:
0x0
Include MPU
Warning: Reset vector and Exception vector cannot be set until memory devices are connected to the Nios II processor
Cancel
< Back
Next >
Finish

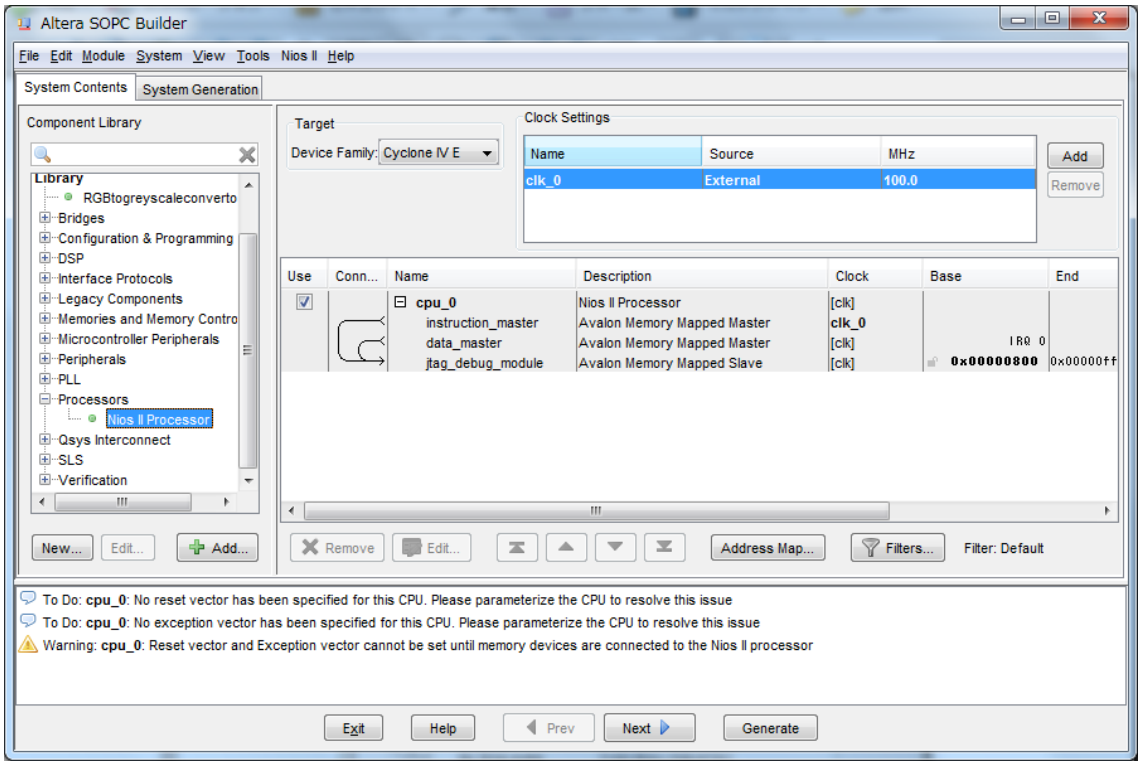
Data　Cache を「None」にし、「Next」ボタンをクリック
※Data Cache については Altera のウェブサイトをご参照ください。
http://www.altera.com/literature/hb/nios2/n2sw_nii52007.pdf
Data Cache を閉じる理由は付録の「Nios II でレジスタ方式で PIO を操作できない問題解析」を参照



残り手順は下図（JTAG Debug Module）までそのまま「Next」ボタンをクリック
下記の画面で JTAG Debug Module を設定できます、即ち JTAG デバッグ時に使われる機能モジュールの設定です、設定されたモジュールが多ければ、必要のリソースも多いです。ここ Level 1 のままで Ok です。「Finish」を押すと、CPU プロセッサの作成が完了します。

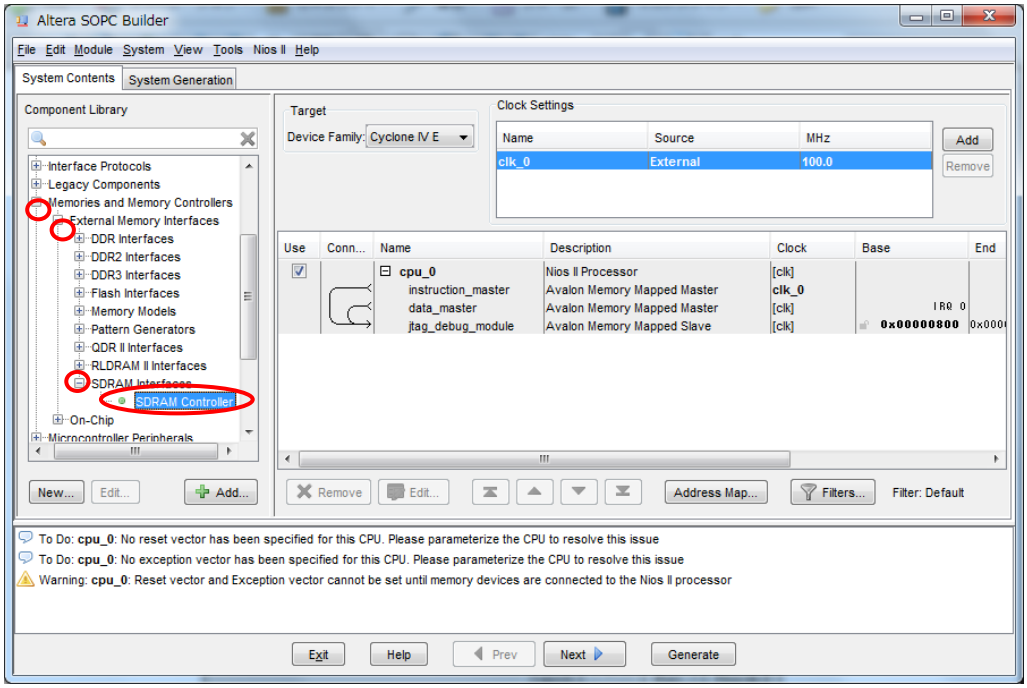


CPU プロセッサ作成完了の様子：



3.3.3 SDRAM モジュール作成

Altera SOPC Builder アプリメイン画面で左側の「Memories and Memory Controller」前の“+”を、「External Memory Interfaces」前の“+”を、「SDRAM Interfaces」前の“＋”をクリック（クリック後、“-”になる）

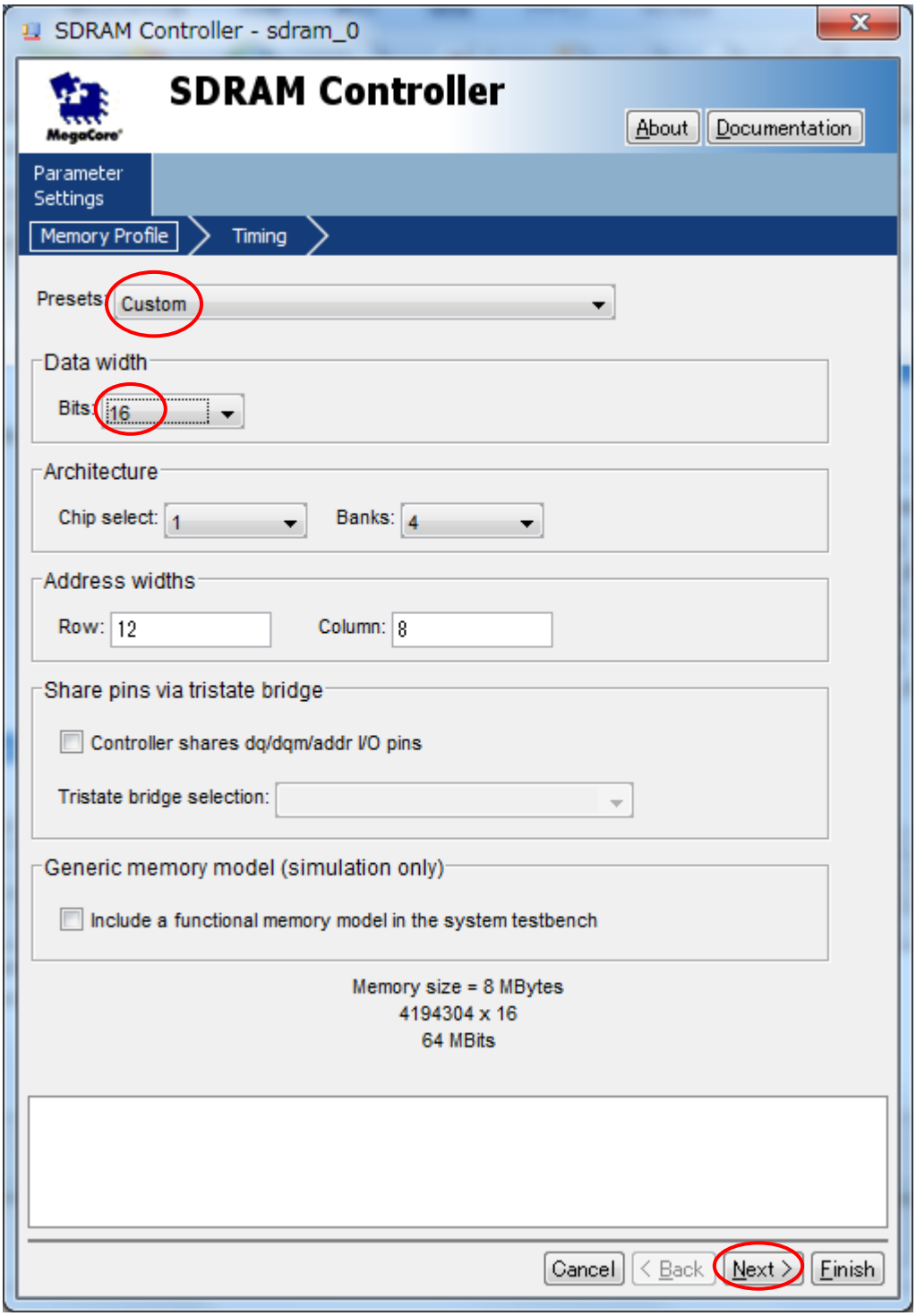


上図の「SDRAM Controller」をダブルクリックして下記の画面が表示されます。

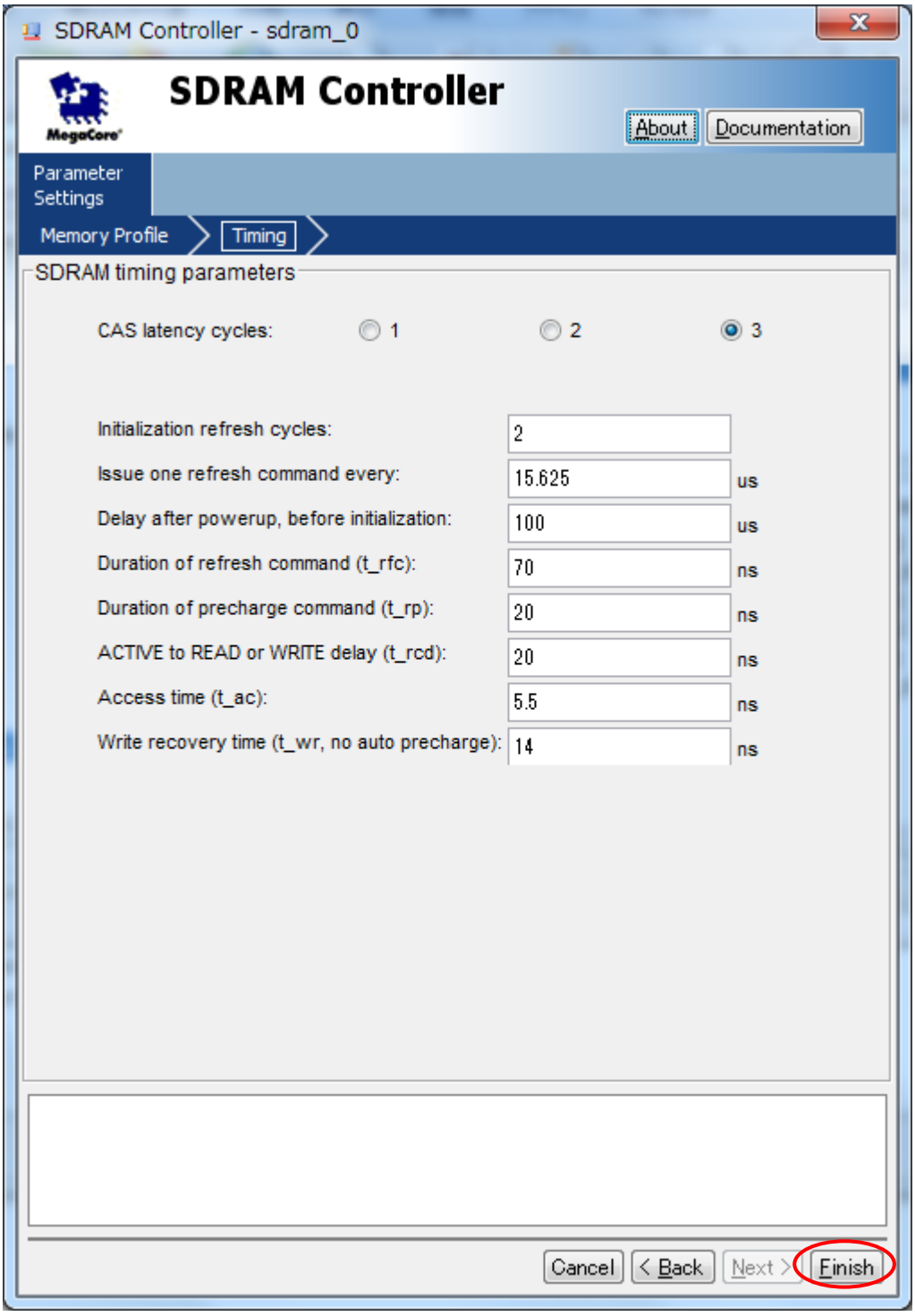
■Presets：「Custom」

■Bits：16

他にそのまま、「Next」ボタンをクリック



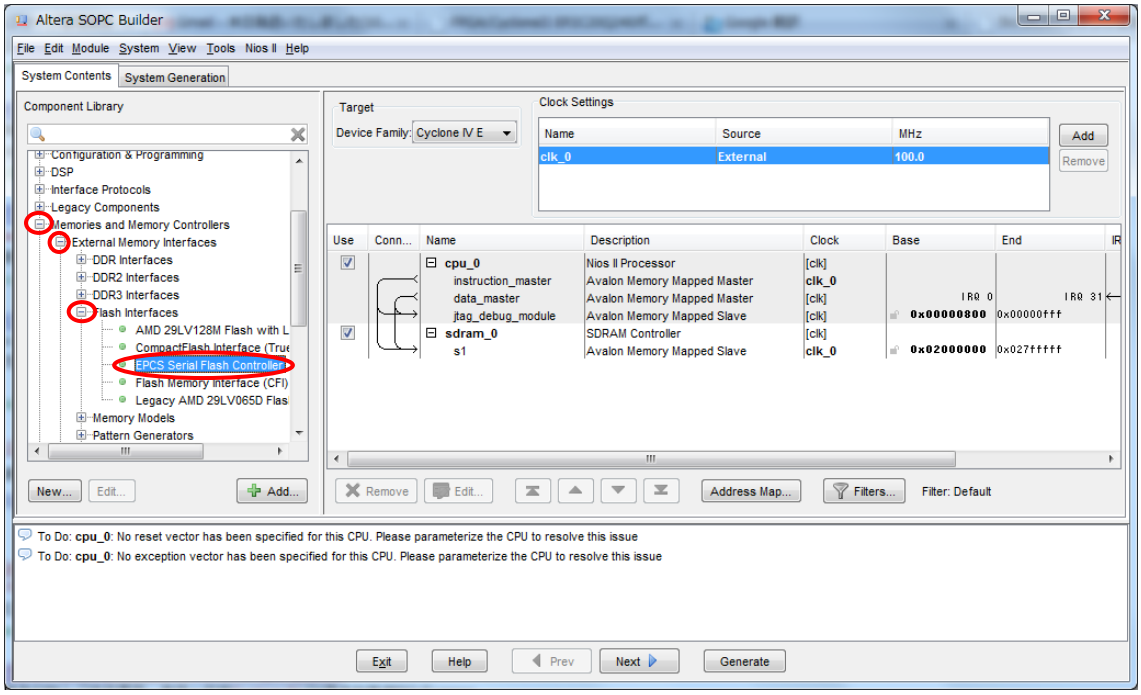
何にも変更せず「Finish」ボタンをクリックすると、SDRAM モジュールの作成が完了します。



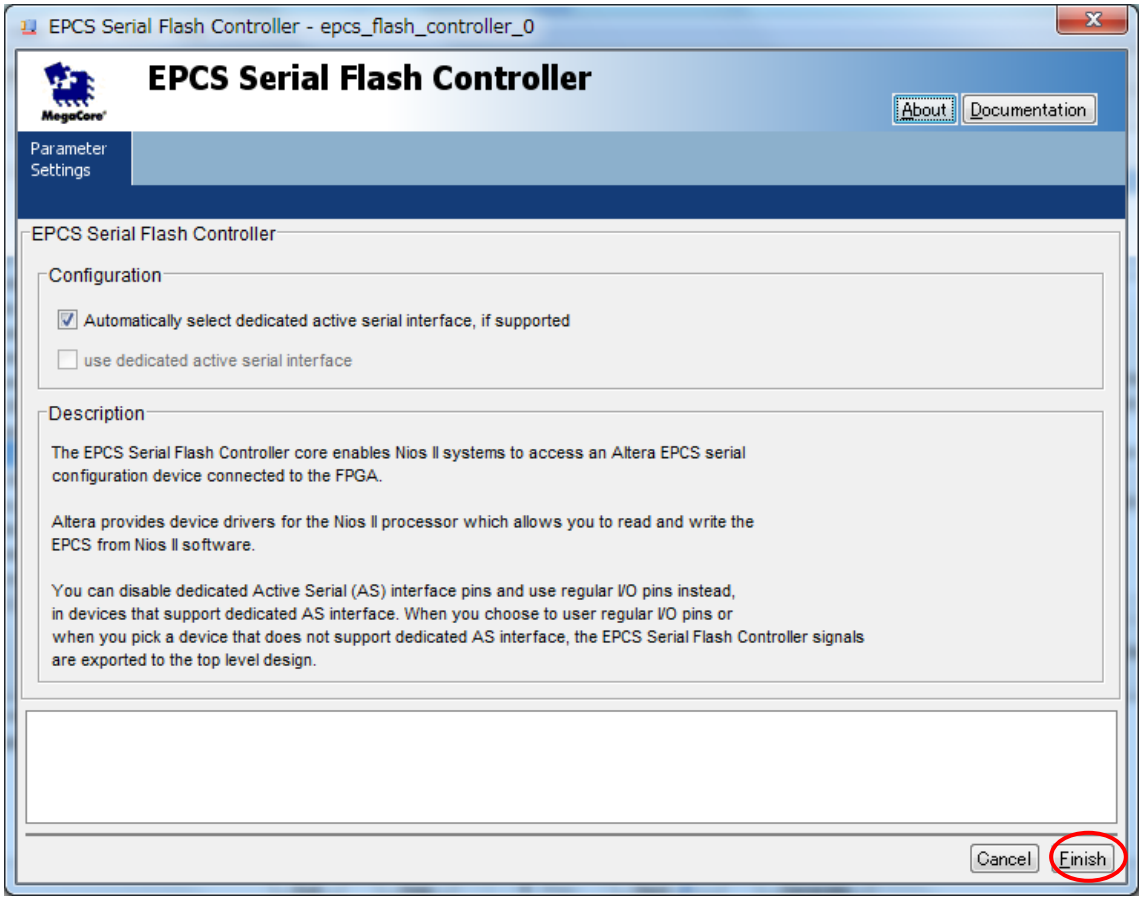
3.3.4 EPCS 作成

Altera SOPC Builder アプリメイン画面で左側の「Memories and Memory Controller」前の“+”を、「External Memory Interfaces」前の“+”を、「Flash Interfaces」前の“＋”をクリック（クリック後、“-”になる）（前から続ける場合、「SDRAM Interfaces」と同じ

レベルにある「Flash Interfaces」を直接クリックしても Ok)

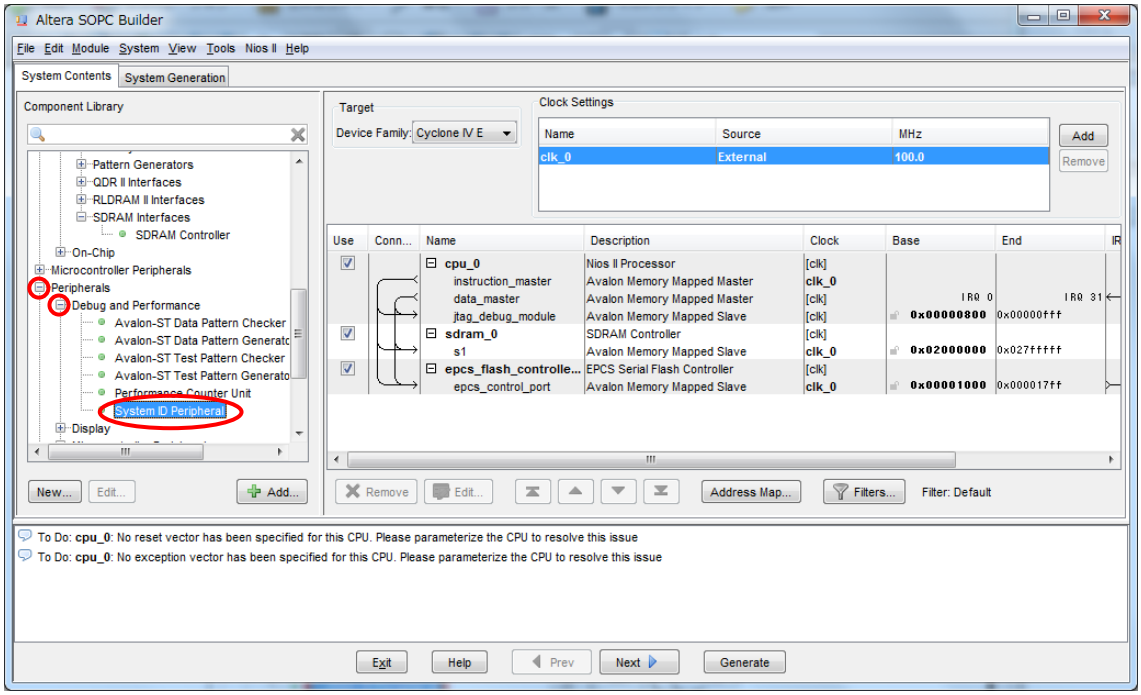


「EPCS Serial Flash Controller」をダブルクリックして変更せずそのまま「Finish」ボタンをクリックすれば、EPCS 作成が完了します。

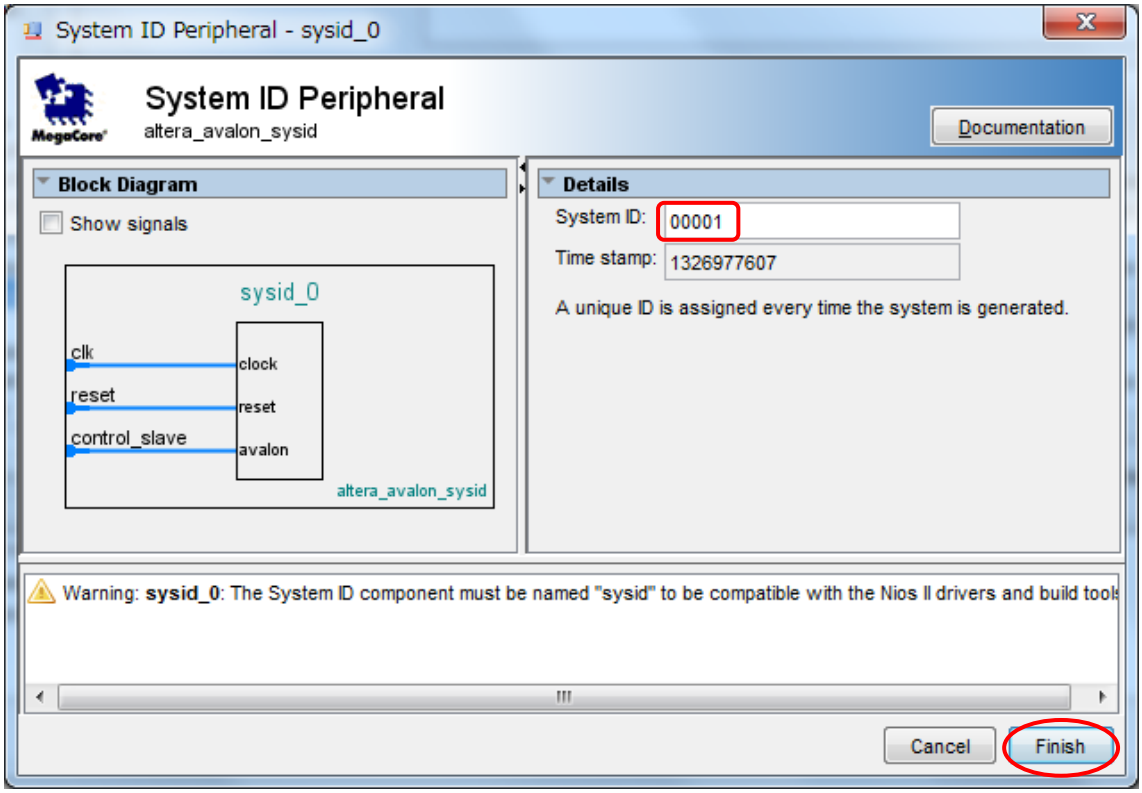


3.3.5 System ID 作成

Altera SOPC Builder アプリメイン画面で左側の「Peripherals」前の“+”を、「Debug and Performance」前の“+”をクリック（クリック後、“-”になる）



「System ID Peripheral」をダブルクリックして、ユニック System ID（例：00001）を入力、「Finish」ボタンを押せば System ID 作成は完了します。

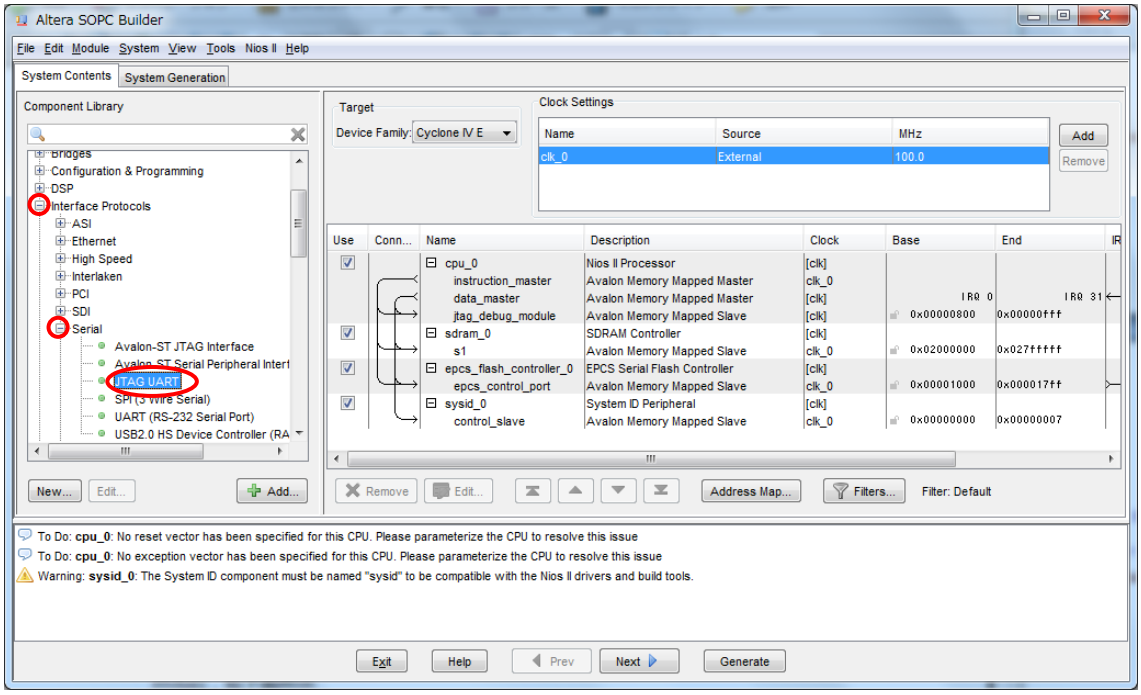


3.3.6 JTAG UART 作成

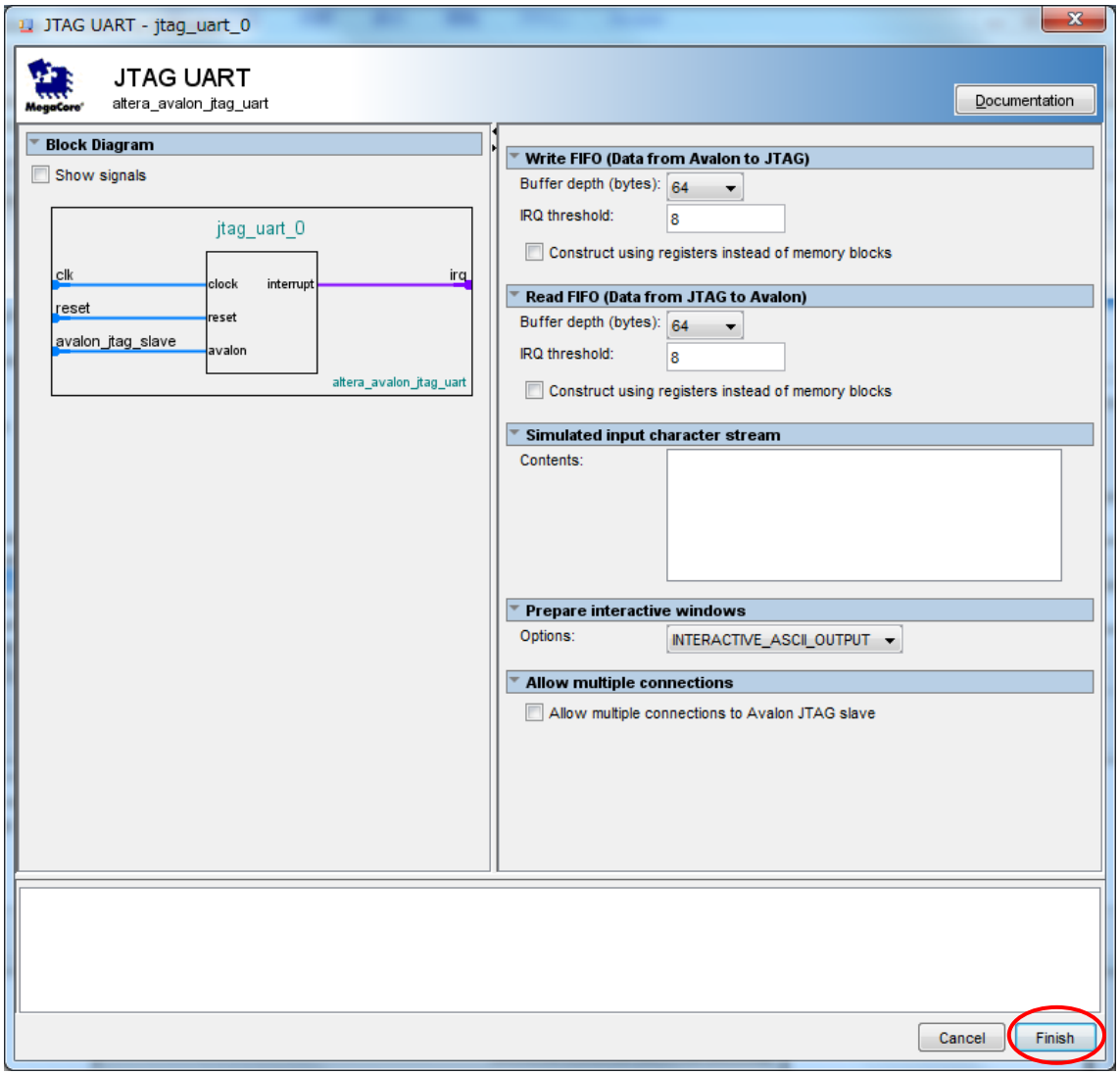
JTAG UART は PC と Nios II の間の通信を実現するシリアルポートインタフェースです、文

字を入出力できます。Nios II 開発で重要な役割を果たします。次は JTAG UART 作成方法を説明します。

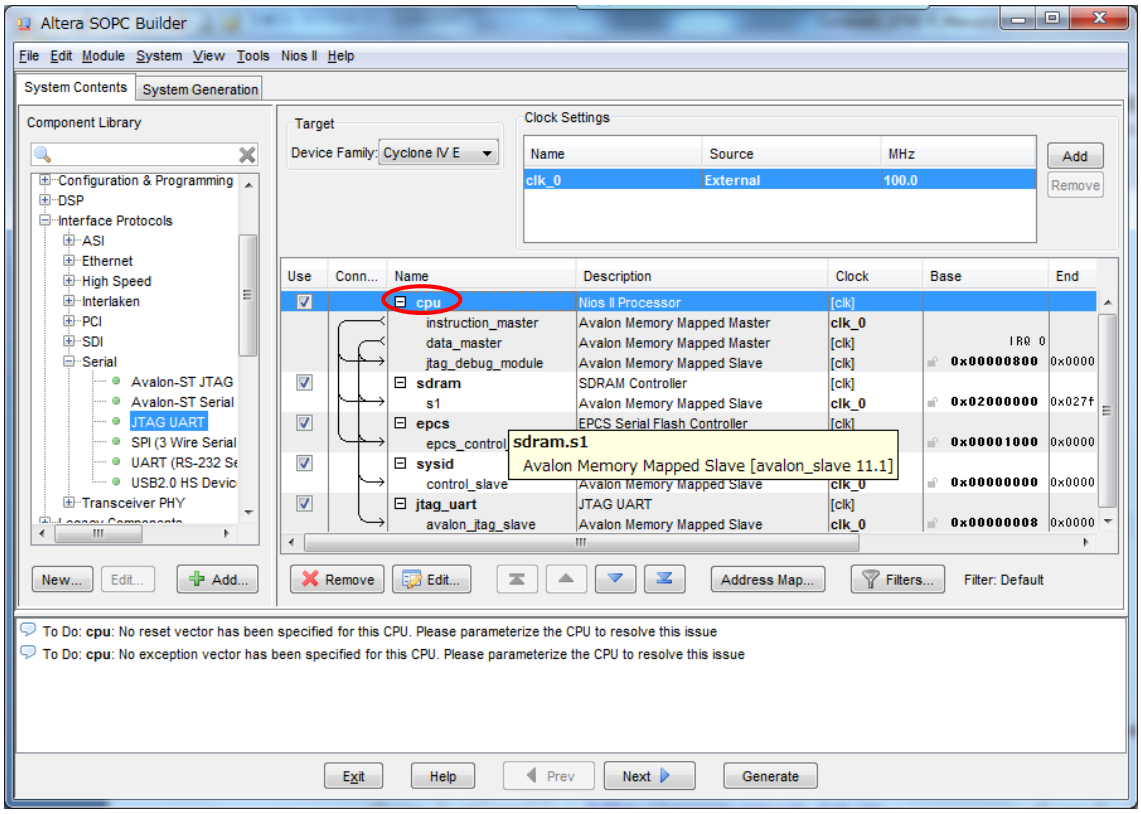
Altera SOPC Builder アプリメイン画面で左側の「Interface Protocols」前の“+”を、「Serial」前の “+” をクリック（クリック後、“-” になる）



JTAG UART をダブルクリック、下記画面に変更せず「Finish」ボタンをクリックすれば、JTAG UART モジュールを作成しました。



ここまで基本な NIOS システムモジュールを作成できました。（各モジュールの名前（_0 を削除）も修正してください。）



3.3.7 NIOS II の設定とコンパイル

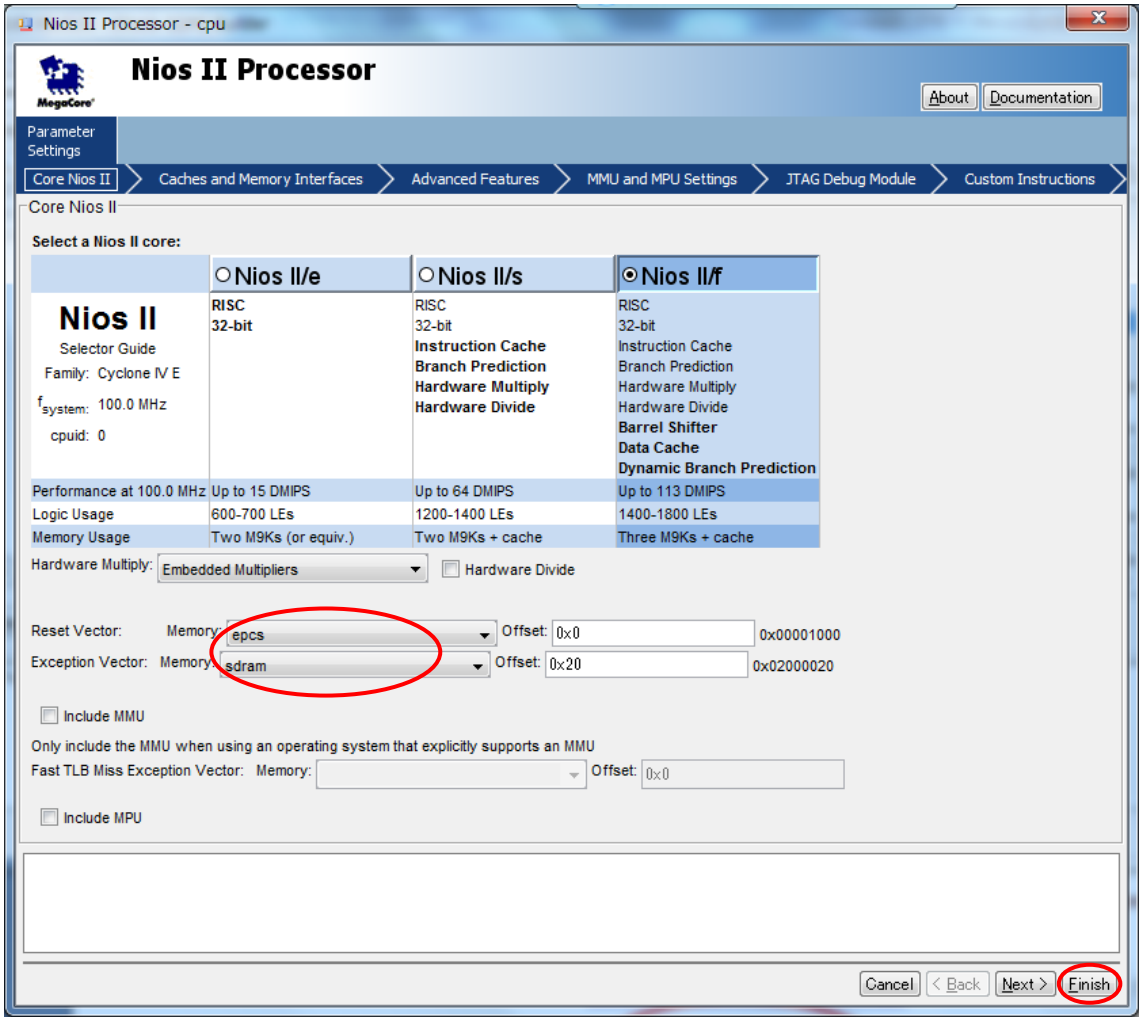
①CPU の設定

上図の「cpu」をダブルクリック、下記画面で

■Reset Vector：Memory：「epcs」を選択

■Exception Vector：Memory：「sdram」を選択

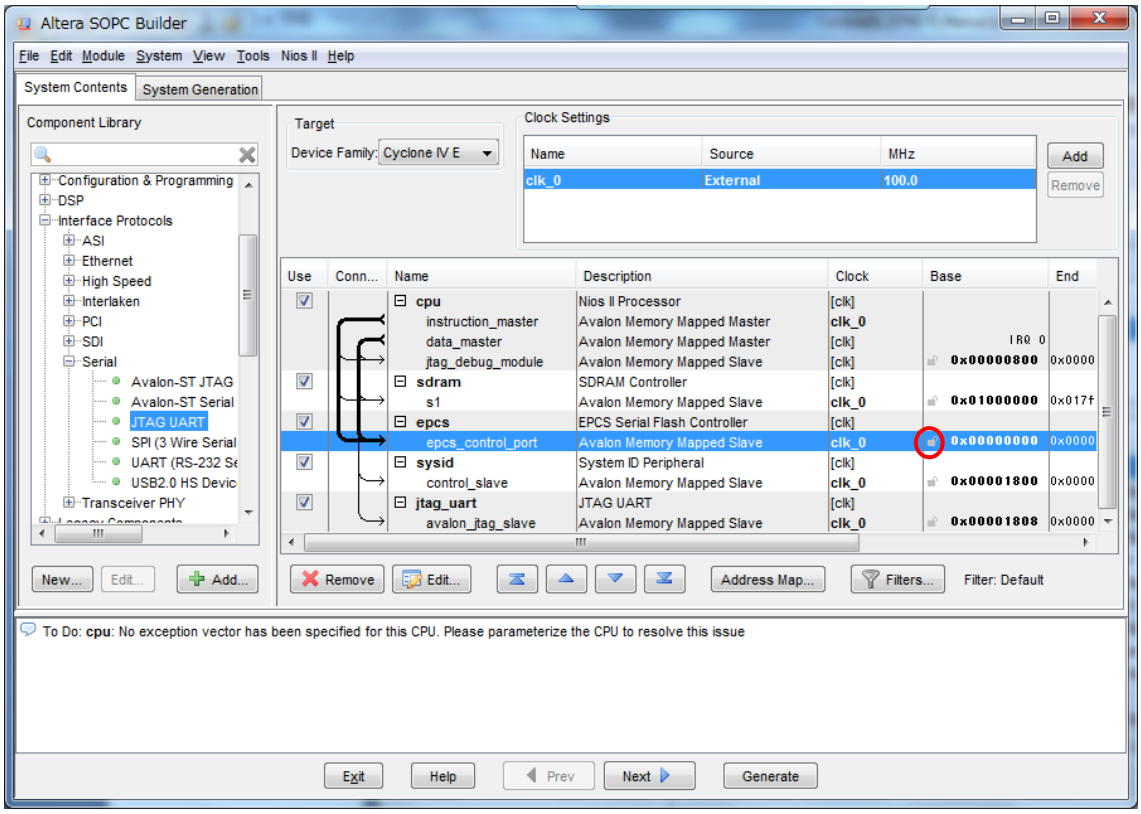
変更したら、「Finish」ボタンをクリック



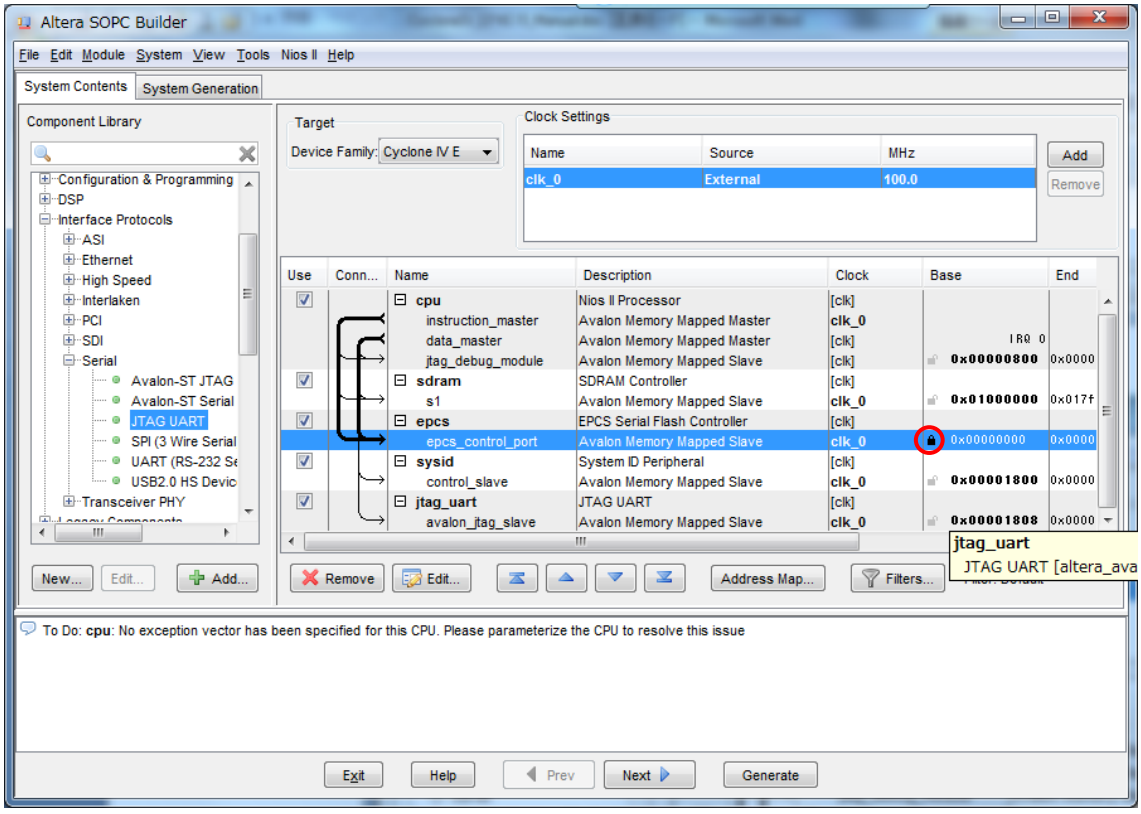
②Flash ベースアドレス設定

Flash 最初のアドレスに「0x00000000」を修正してからロックを掛けます、以下のように操作を実施してください。（ロックの所をクリック）

※Flash の最初アドレスに「0x00000000」を設定された場合、起動時アドレス「0x00000000」から起動します、これは一般のやり方です。



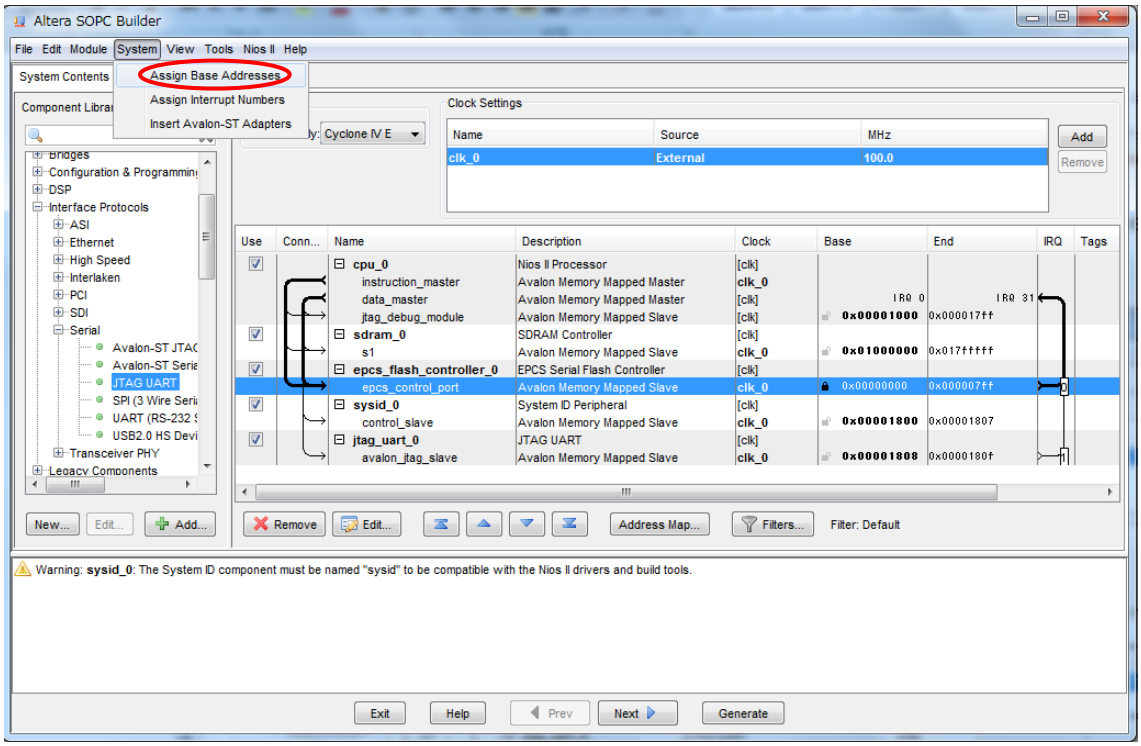
クリックしたら、ロックが締めた感じです。



③アドレスの自動割り当て

アドレスの自動割り当ての目的は無駄の容量を使わないためです。特別のニーズがあれば、手動で設定しても OK。

Altera SOPC Builder アプリメイン画面でメニュー「System」→「Assign Base Addresses」をクリック。自動割り当てられても Flash のアドレスが「0x00000000」になって、変わりません。(ロックが有効)



次は割り込みの割り当て。

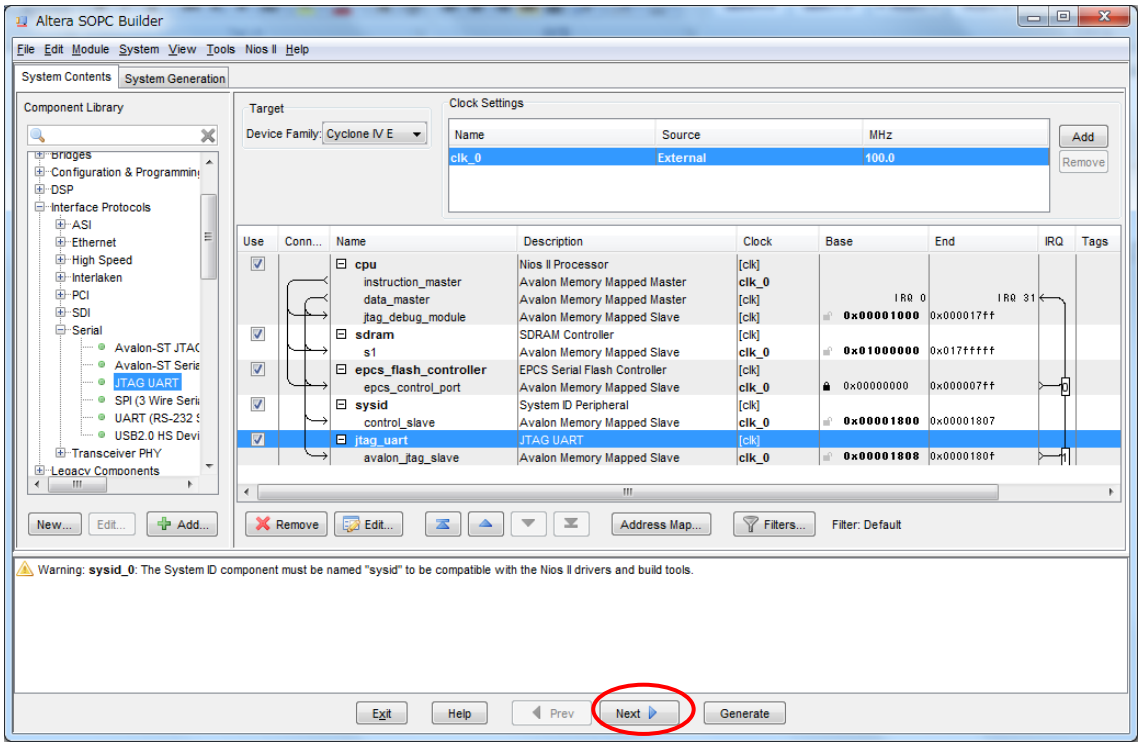
アドレス割り当てと同じように、「System」→「Assign Interrupt Numbers」

(基本的に Quartus II から自動的に割り当てられます)

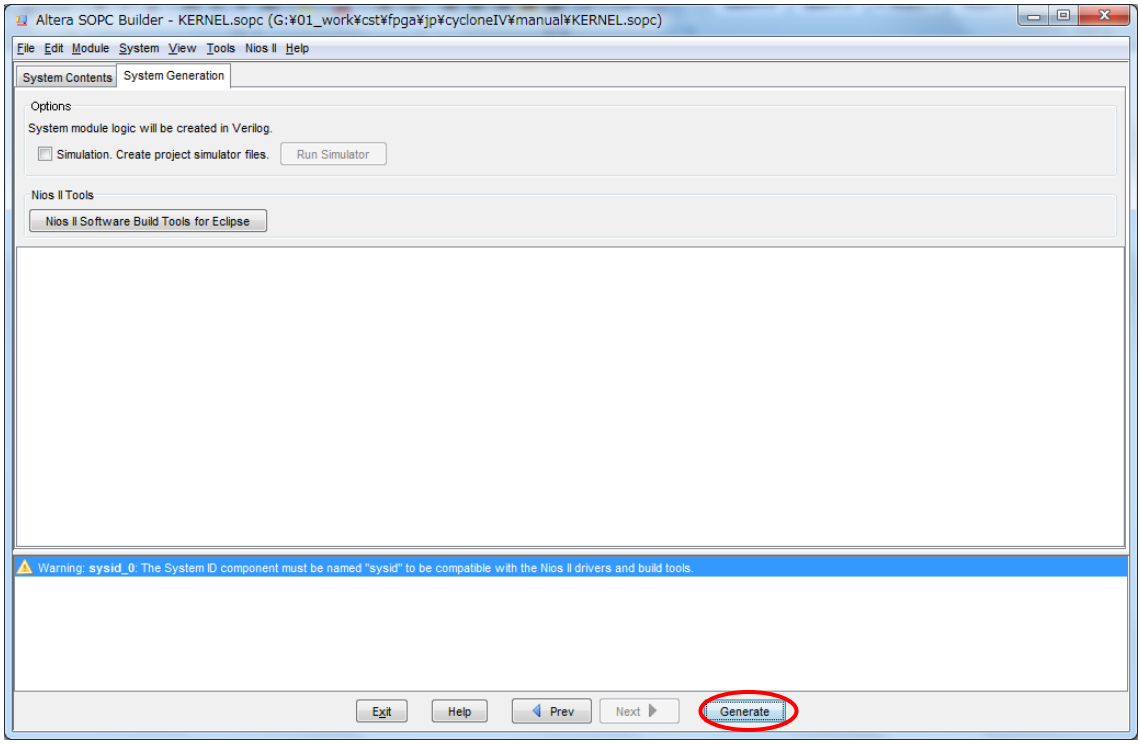
各モジュール名前を修正しましょう (_0 を外す)、特に System ID の名前を「sysid」に修正しないと警告メッセージも出てきます。

これからコンパイルして行きましょう。

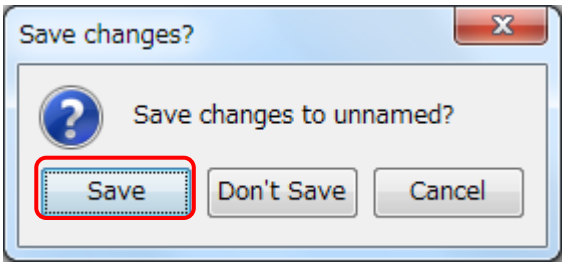
名前を修正したら、下記画面で「Next」ボタンをそのままクリック



「Next」ボタンをクリック

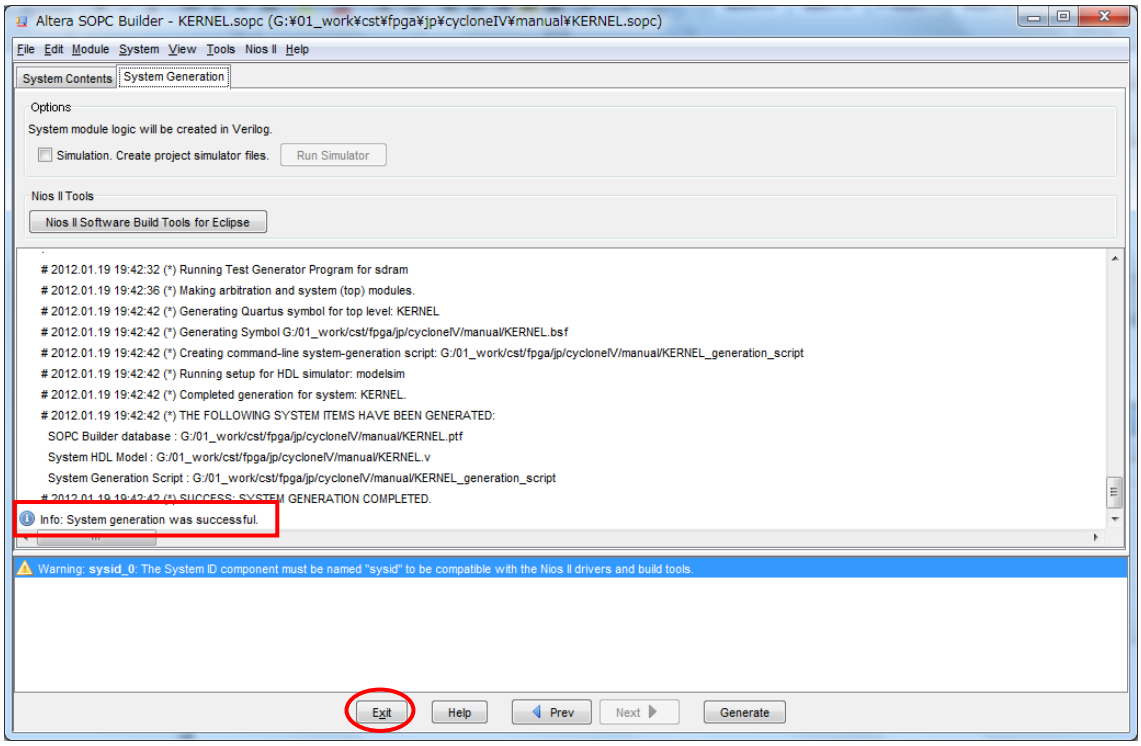


「Generate」をクリックすると、



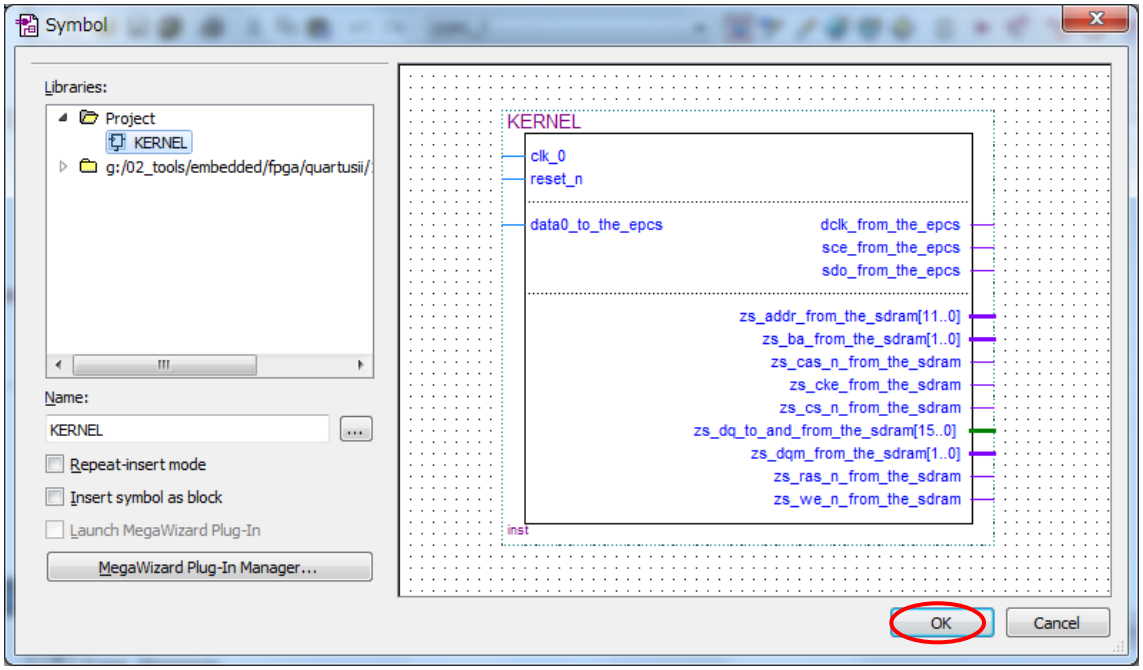
「KERNEL」として保存しましょう。保存したら、コンパイルを始めます。(休憩を取りましょう)

暫く待ってコンパイルが完了します、下記のような画面が出ます。
「Info：System generation was successful.」というメッセージがありましたら、
正常にコンパイルできた事を明らかにします。「Exit」ボタンをクリックして Quartus 画面に戻ります。

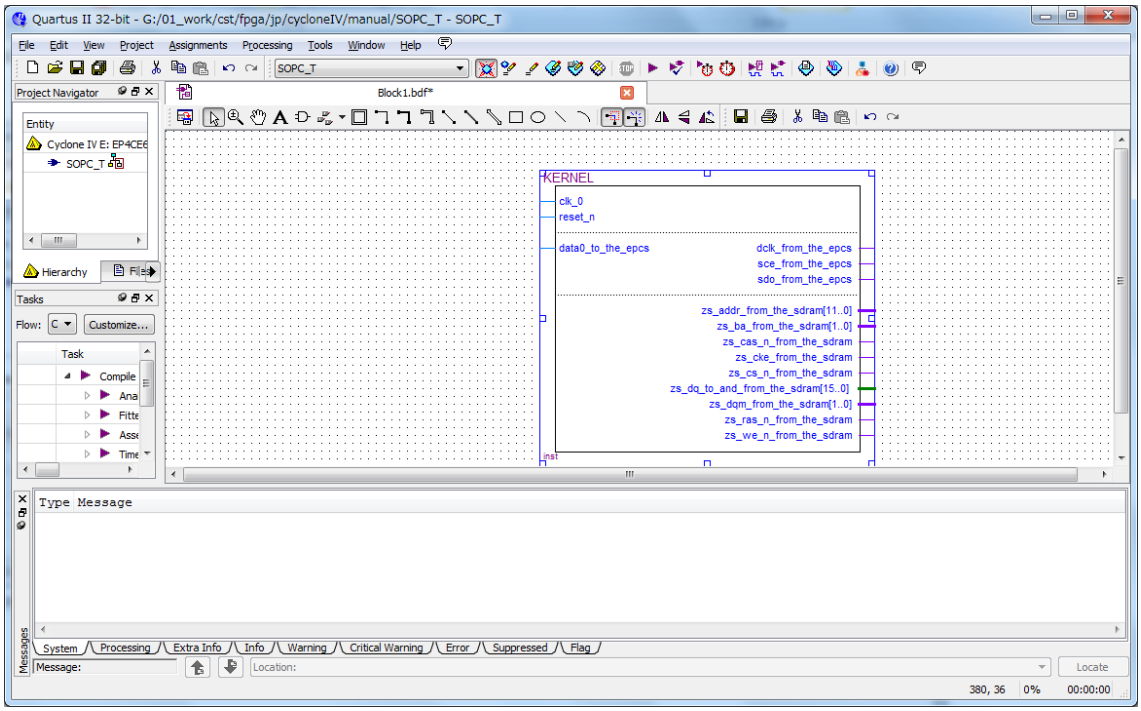


3.3.8 ピンの割り当て

Quartus メイン画面で Block1.bdf の空白所にダブルクリックすると、以下の画面が出ます。
「Project」をクリックして作った「KERNEL」を選択し「OK」ボタンを押下

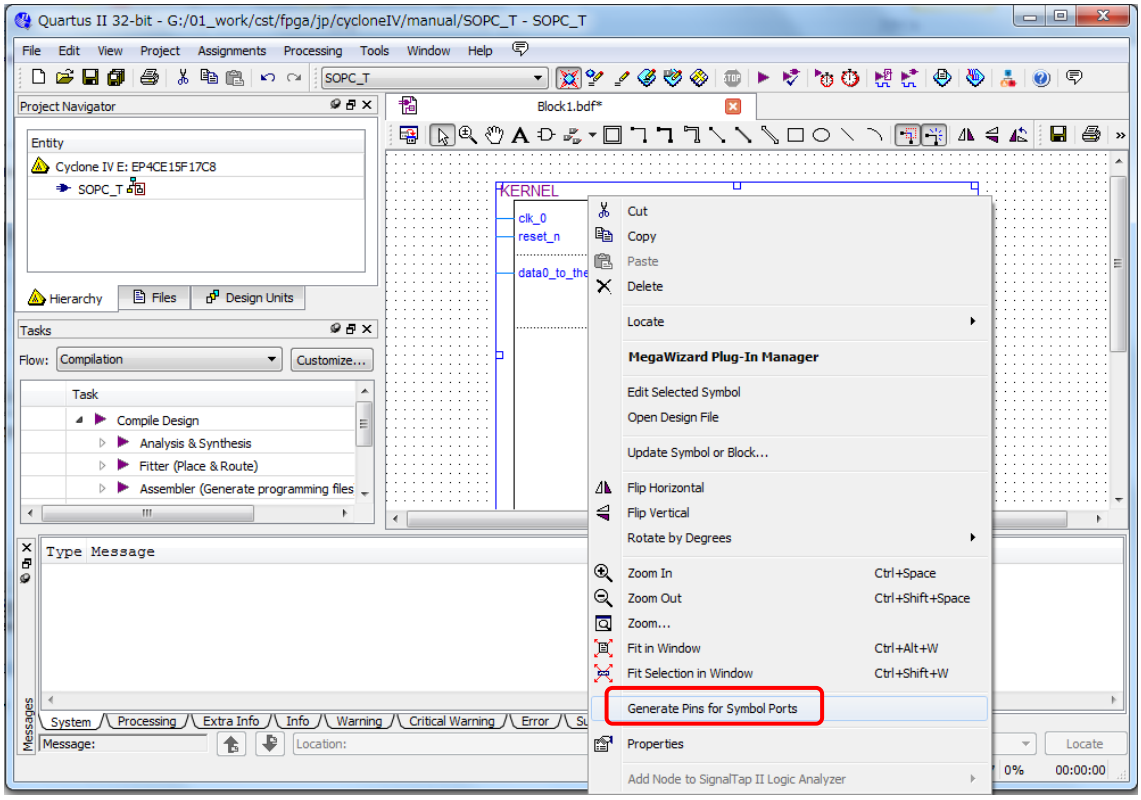


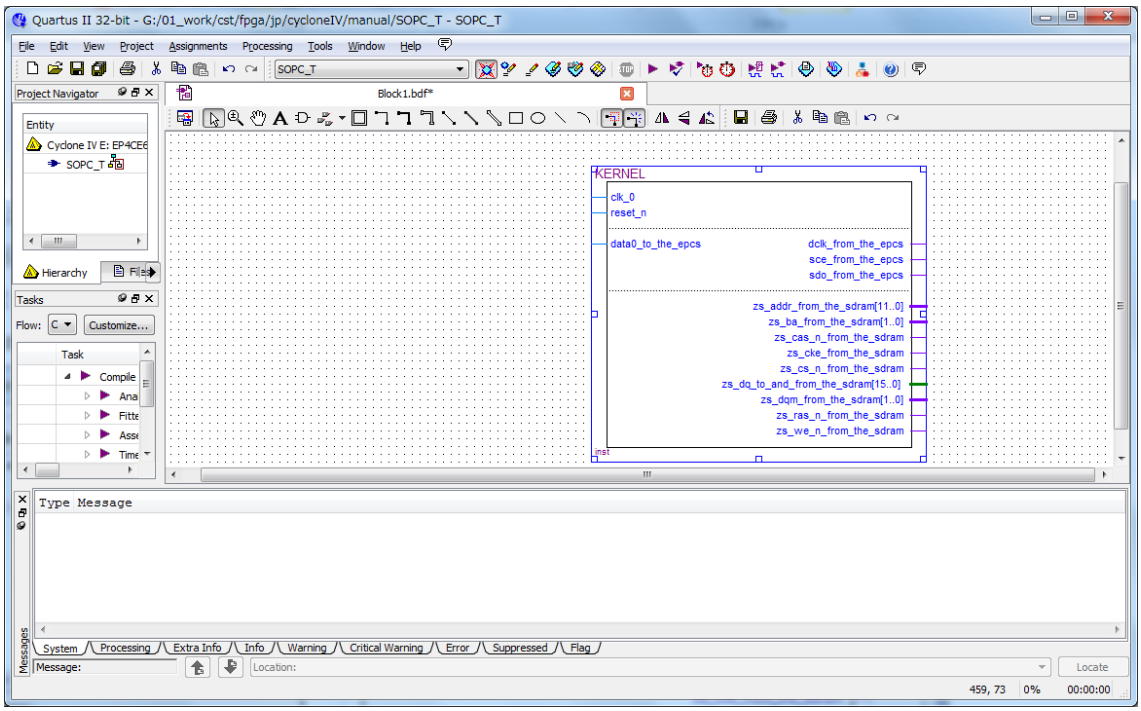
ブロックがマウスに粘着されます、下記画面のように Block1.dbf に真ん中に置きましょう。



ソフトマクロ「KERNEL」に右クリック、「Generate Pins for Symbol Ports」を選択

※これはピンの割り当てのことです。

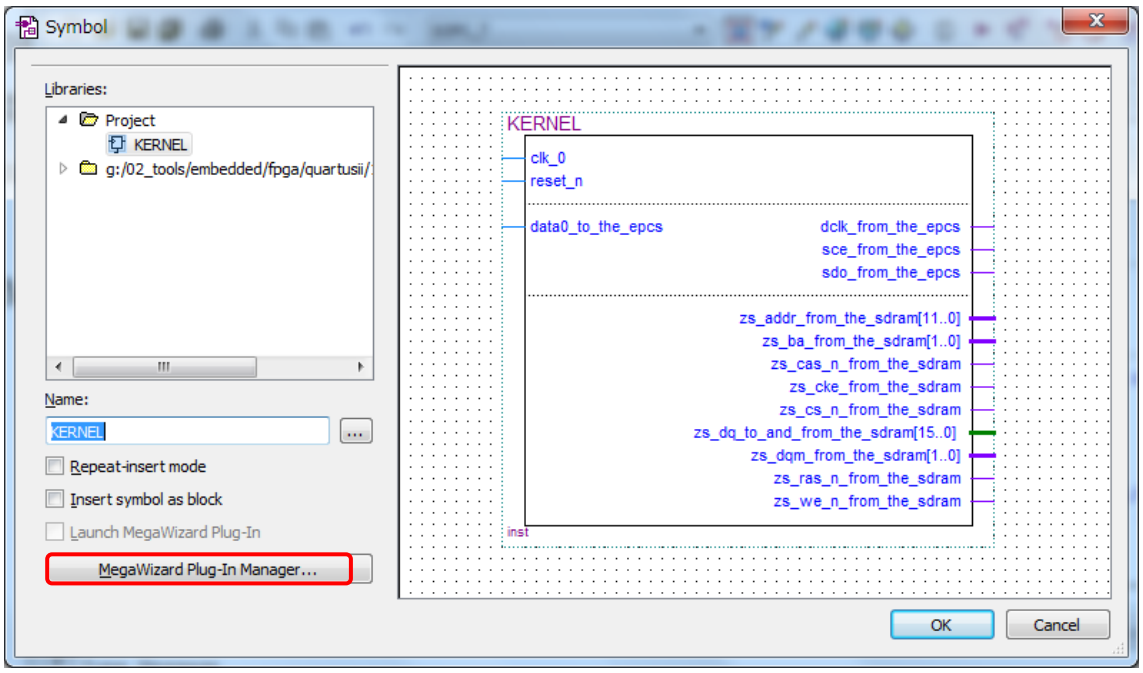

Quartus II 32-bit - G:/01_work/cst/fpga/jp/cycloneIV/manual/SOPC_T - SOPC_T
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Entity
Cyclone IV E: EP4CE15F17C8
SOPC_T
Hierarchy Files Design Units
Tasks
Flow: Compilation Customize...
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programming files
Block1.bdf*
KERNEL
clk_0
reset_n
data0_to_the
Cut
Copy
Paste
Delete
Locate
MegaWizard Plug-In Manager
Edit Selected Symbol
Open Design File
Update Symbol or Block...
Flip Horizontal
Flip Vertical
Rotate by Degrees
Zoom In Ctrl+Space
Zoom Out Ctrl+Shift+Space
Zoom...
Fit in Window Ctrl+Alt+W
Fit Selection in Window Ctrl+Shift+W
Generate Pins for Symbol Ports
Properties
Add Node to SignalTap II Logic Analyzer
Type Message
System Processing Extra Info Info Warning Critical Warning Error
Message:
Location:
Locate
0%
00:00:00



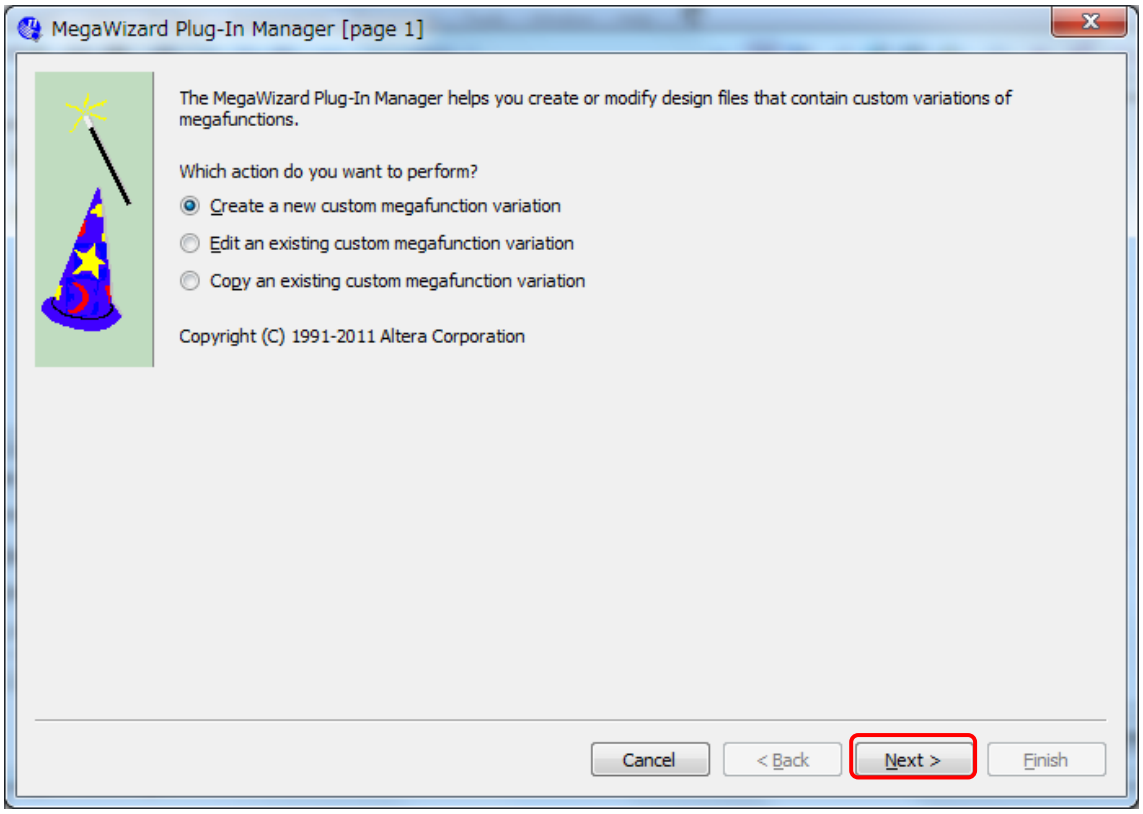
3.4 位相ロックループ PLL 作成

これから位相ロックループ PLL を作成して行きます。作成目的は、CycloneIV ボードの水晶発振器が 50MHZ となり、クロックの周波数を増加させ、100MHZ の Nois II ソフトマクロの周波数を満たします。それ以外、SDRAM の 100MHZ クロックも提供必要です。

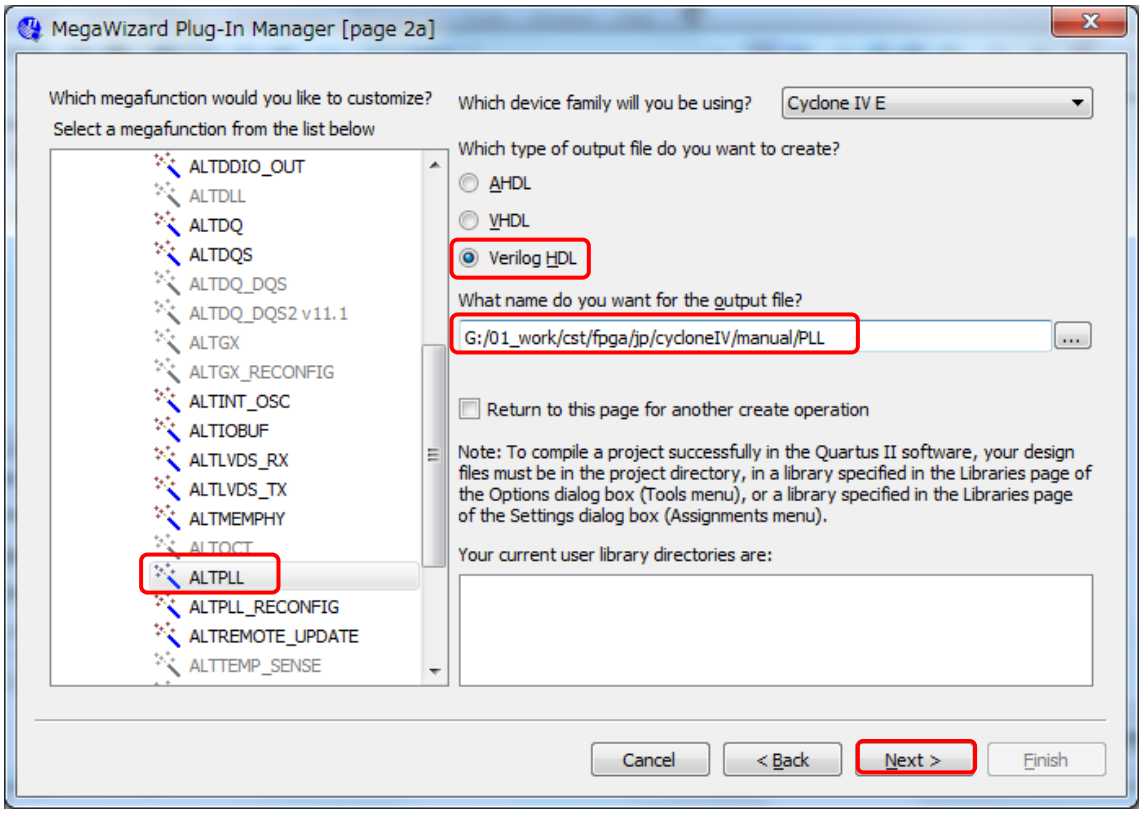
Quartus メイン画面で「Block1.bdf」の空白にダブルクリック、下記の画面が出ます。



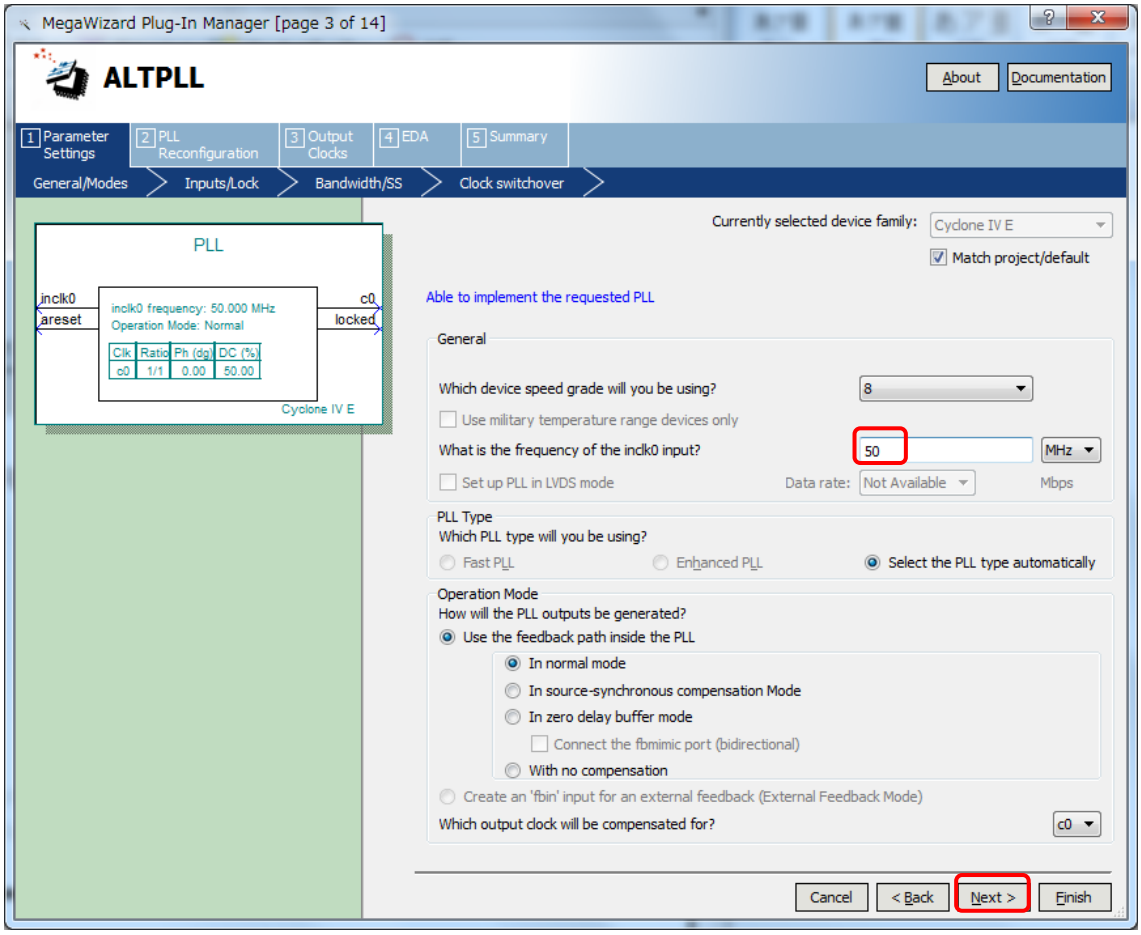
「MegaWizard Plug-In Manager...」ボタンをクリック



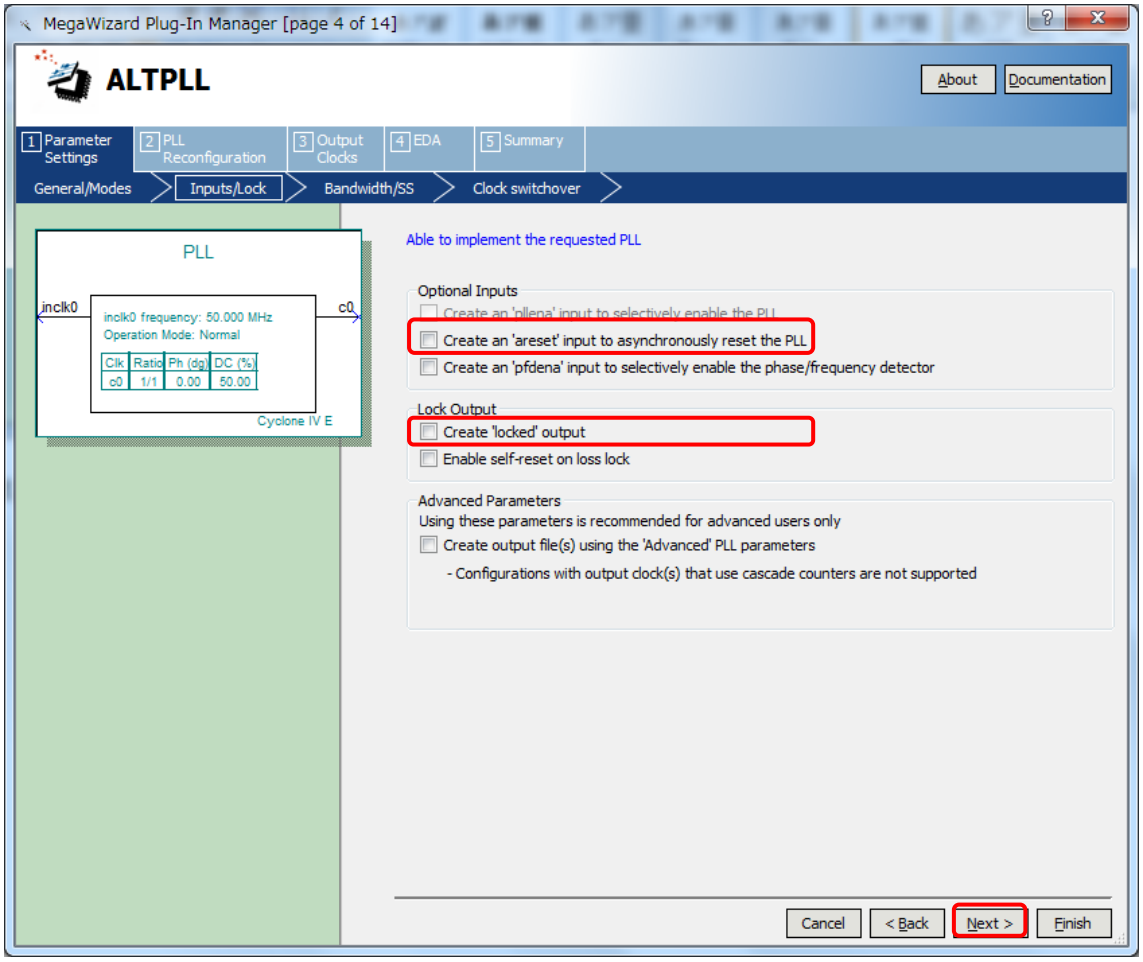
「Next」ボタンをクリック、下記のような画面が出ます。左側の I/O をダブルクリック、「ALTPLL」を選択、右側に下図のように設定（保存場所はお好きな所を変更）



変更が終わったら、「Next」ボタンを押下、下図が表示されます。
水晶発振器の周波数と合わせて「50MHZ」に修正しましょう。

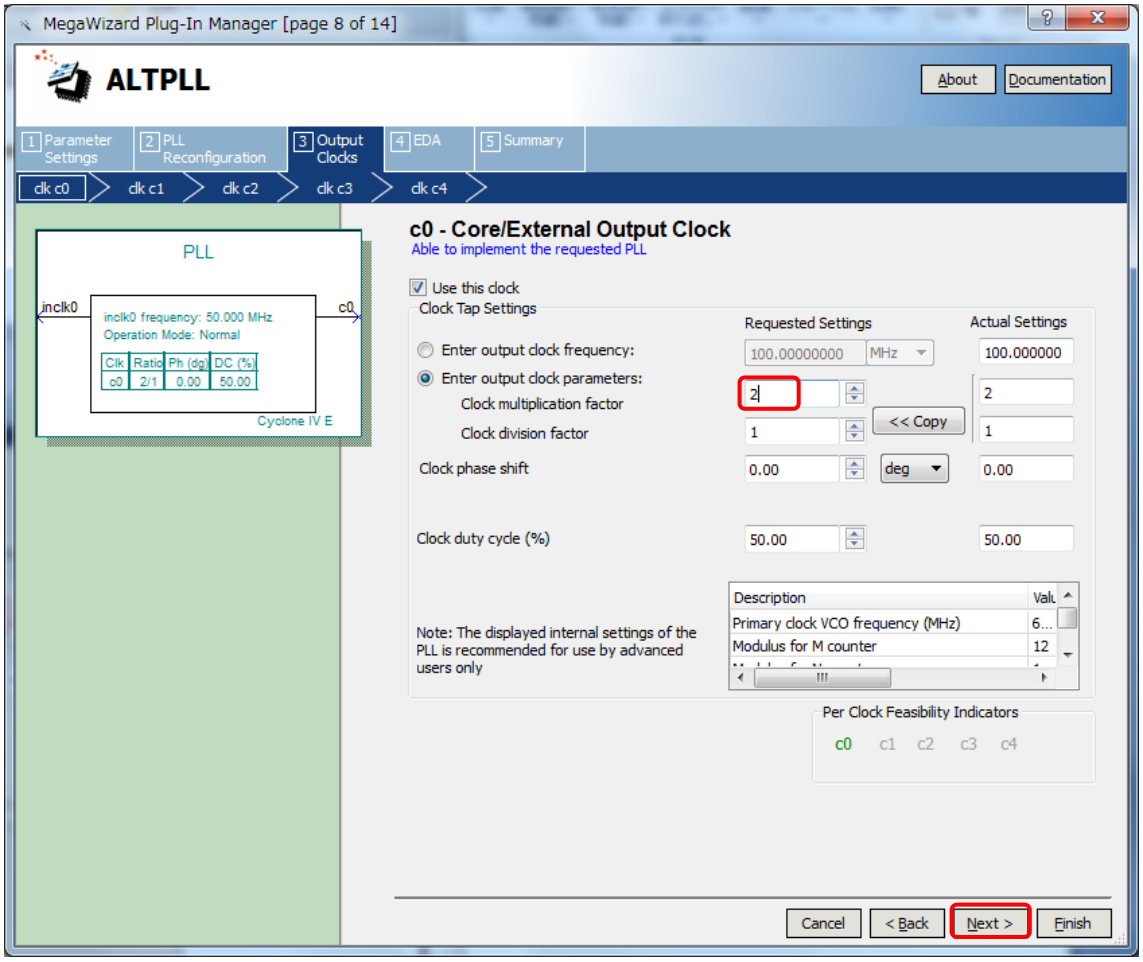


修正したら、「Next」ボタンをクリック、下図のようにチェックを外してください。



「Next」をクリック、下記画面が出るまでに次の画面でも「Next」をクリック

■Clock multiplication Factor:「2」に修正（Actual Settings が 100MHZ になるように）



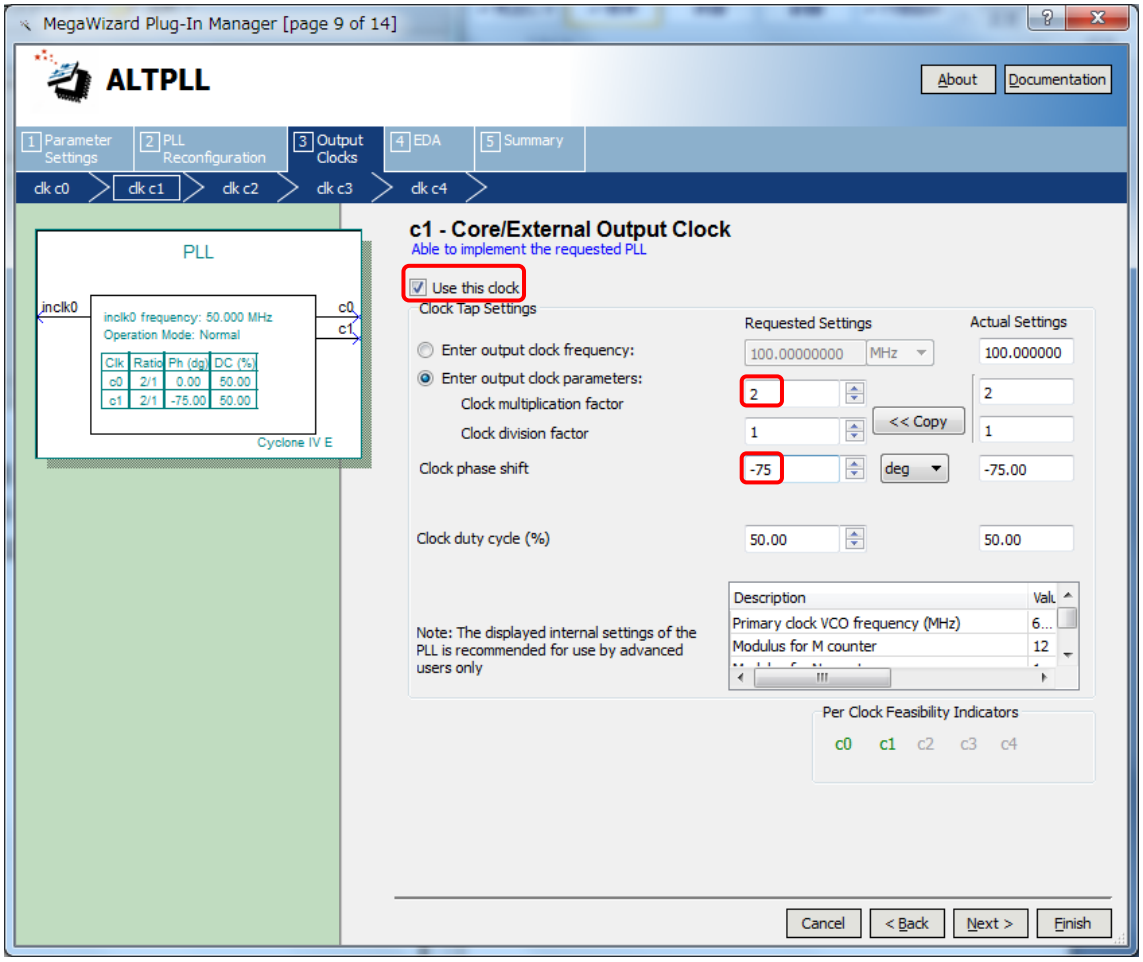
「Next」ボタンをクリック

■Use this clock：チェック

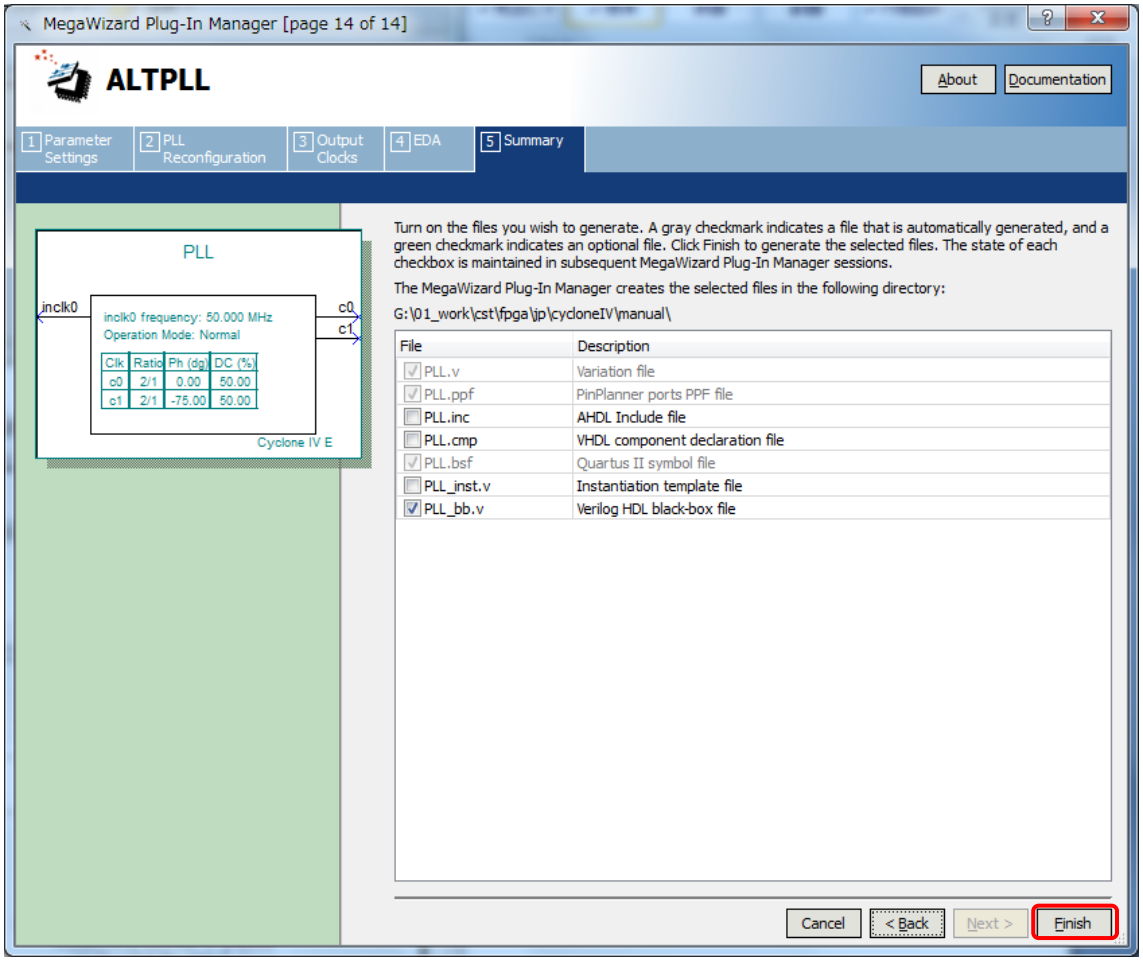
■Clock multiplication Factor:「2」に修正（Actual Settings が 100MHZ になるように）

■Clock phase shift:「-75」deg に修正

この部分は SDRAM にクロックを提供するための設定です。位相ロックループ PLL の c1 を利用し、クロックが 100MH z を設定にし、オフセットが-75deg に設定（この設定は SDRAM が正常に動くかに影響を及ぼします、後程、詳しく説明します。）

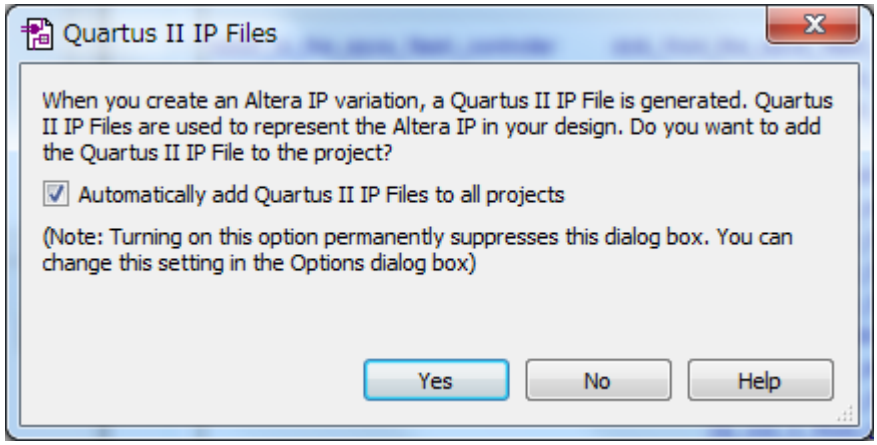


設定が完了後、繰り返して「Next」ボタンをクリックし、最後「Finish」ボタンを押して PLL の作成を完成させましょう。



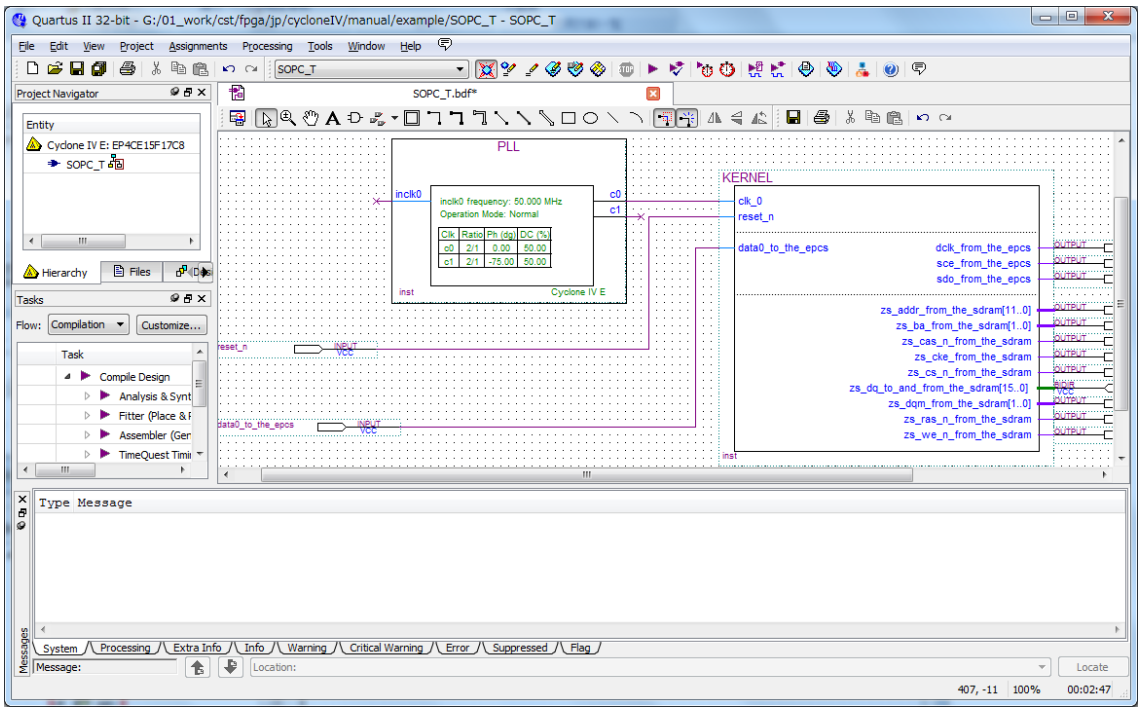
Finish ボタンを押した後、下記のダイアローグが出ます、

「Automatically add Quartus II IP Files to all projects」をチェックし、「Yes」ボタンをクリック（1 回実施したら、今後出ない）



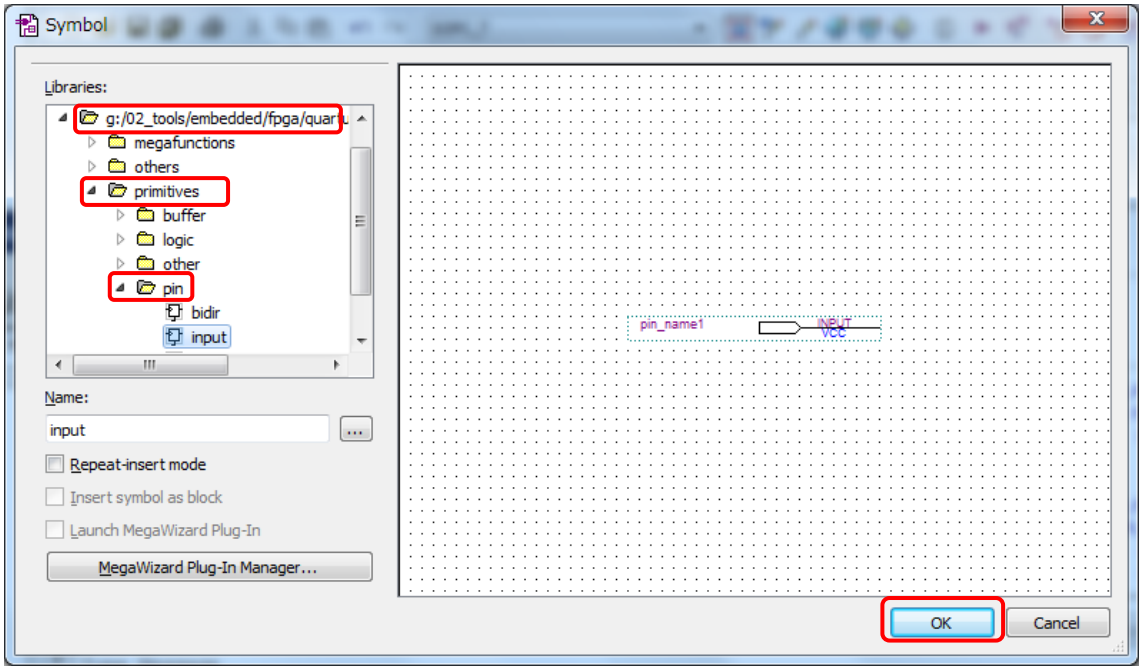
以下の画面が出る時「OK」をクリックして PLL を Block1.bdf に置きましょう。

※PLL のピンを NIOS II ソフトマクロと接続してください。



次に、inclk0:input、c1:output を追加しましょう。

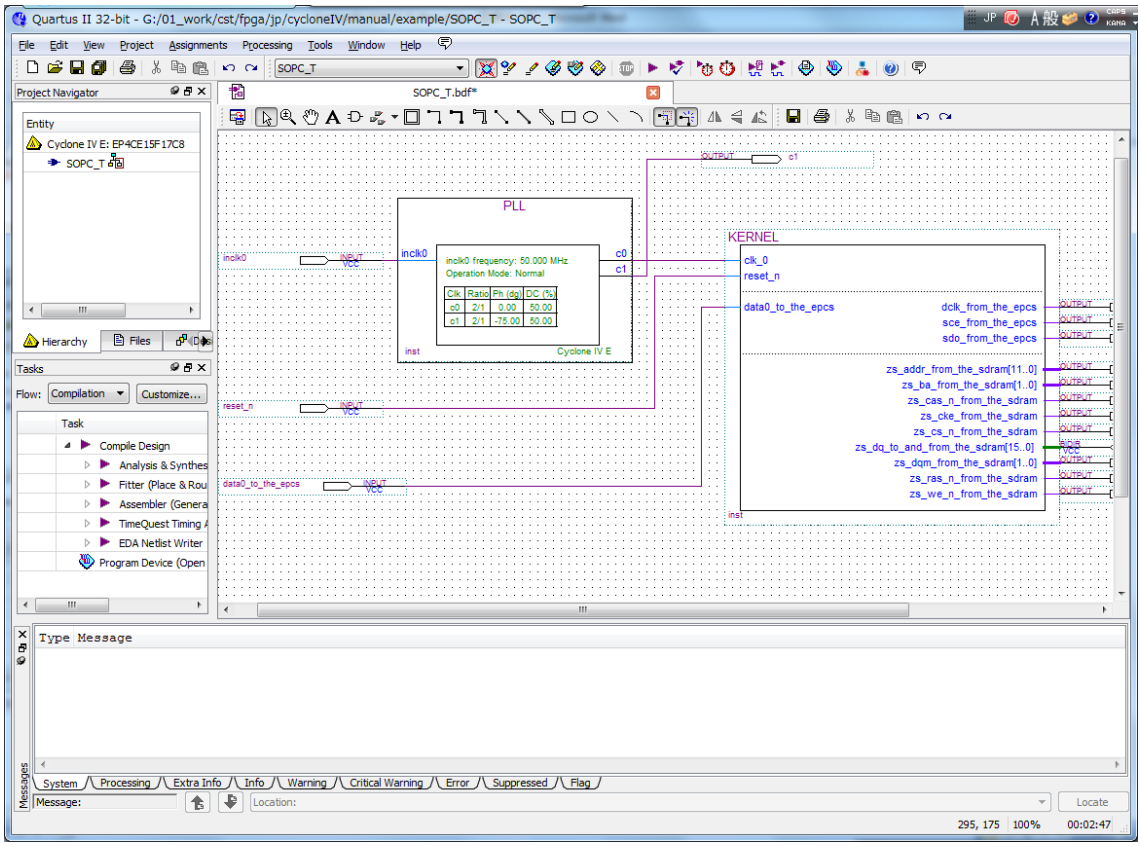
Block1.bdf の空白でダブルクリックして、下記の画面が出ます。以下のように操作して「Ok」をクリックすれば、input のピンを追加できます。



output も同じように追加できます。(追加時、上記画面の Name：output を入力あるいは pin の中の output を選択)

※ピンの名前を変更したい場合、名前の所をダブルクリックして変更できます。

※c1 が SDRAM のクロックにピンを割り当てる



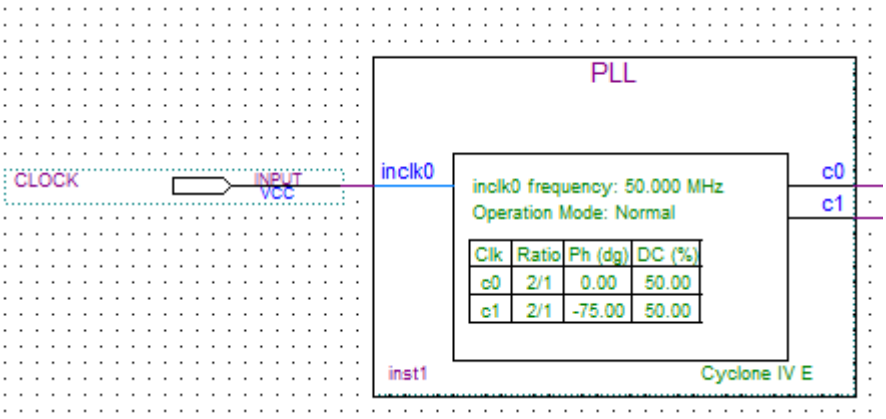
3.5 TCL スクリプトファイル

これから FPGA のピンを割り当て行きます。ピンの割り当てには二つ方法がありますが、ここ良い一つ方法：TCL スクリプトファイルの使用をお勧めします。このファイルは定めた記述ルールがありますので、具体的なピンの情報により修正すれば OK です。

下図示すのように(CycloneIV 用、CycloneII 用のものは CyclonII フォルダにある)

```
#------------------SDRAM---------------------#
set_location_assignment PIN_175 -to S_DB[0]
set_location_assignment PIN_173 -to S_DB[1]
set_location_assignment PIN_171 -to S_DB[2]
set_location_assignment PIN_170 -to S_DB[3]
set_location_assignment PIN_169 -to S_DB[4]
set_location_assignment PIN_168 -to S_DB[5]
set_location_assignment PIN_165 -to S_DB[6]
set_location_assignment PIN_164 -to S_DB[7]
set_location_assignment PIN_205 -to S_DB[8]
set_location_assignment PIN_203 -to S_DB[9]
set_location_assignment PIN_201 -to S_DB[10]
set_location_assignment PIN_200 -to S_DB[11]
set_location_assignment PIN_199 -to S_DB[12]
set_location_assignment PIN_198 -to S_DB[13]
set_location_assignment PIN_197 -to S_DB[14]
set_location_assignment PIN_195 -to S_DB[15]
```

我々の作業は後ろの名前を修正するだけです、PIN_*が FPGA ボード上のピンとなり、S_DB[*]が該当の名前です。

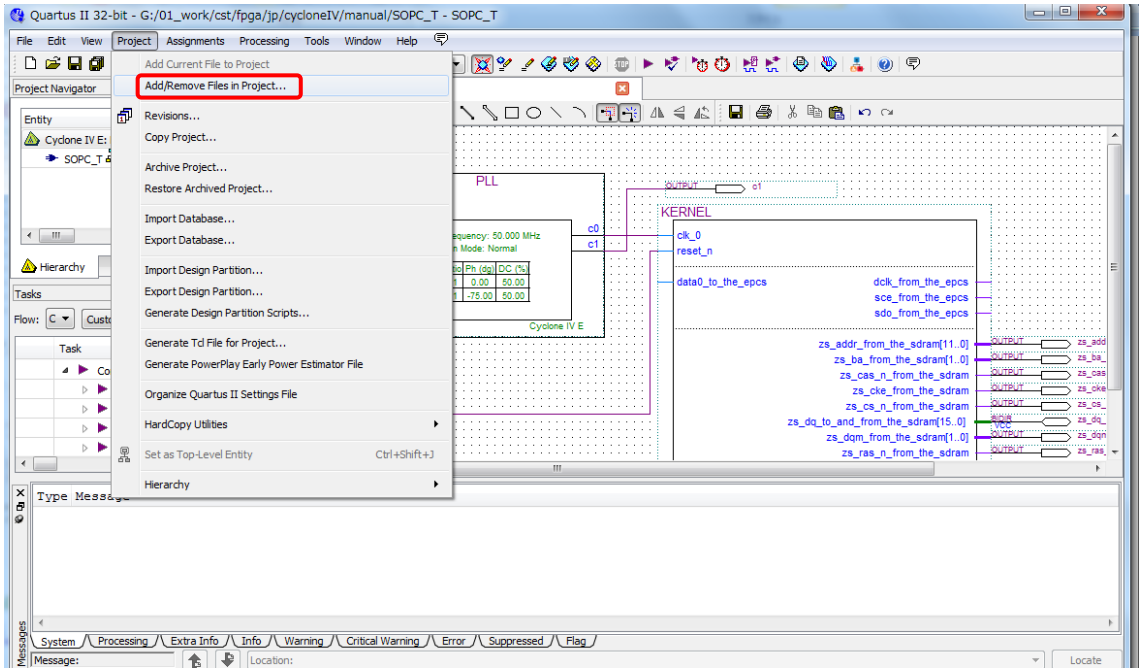


例をあげて説明しましょう。

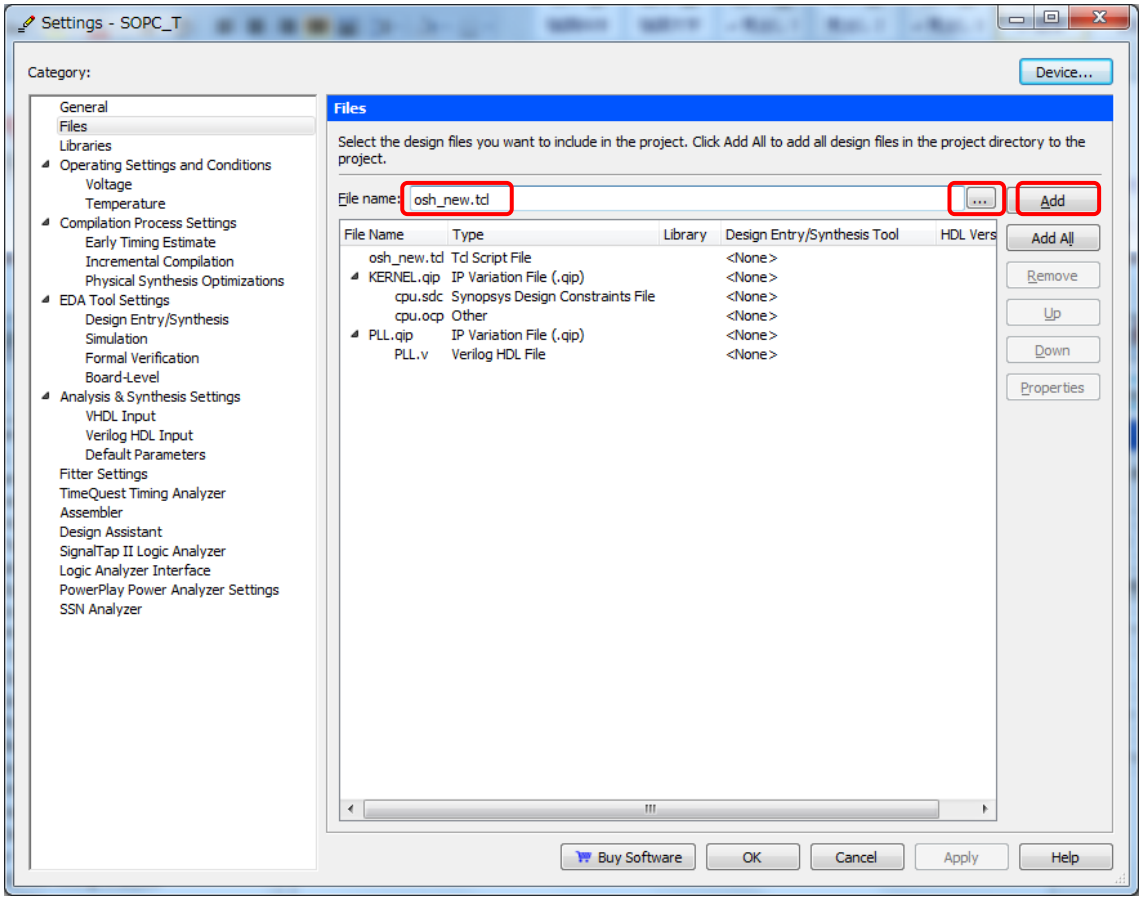
クロックピンの場合、まず、名前を「CLOCK」に修正、次は TCL スクリプトファイルを見ていきましょう。(弊社ウエブサイトから CycloneIV ボード用の TCL をダウンロードできます。)

TCL ファイルをプロジェクトにインポート

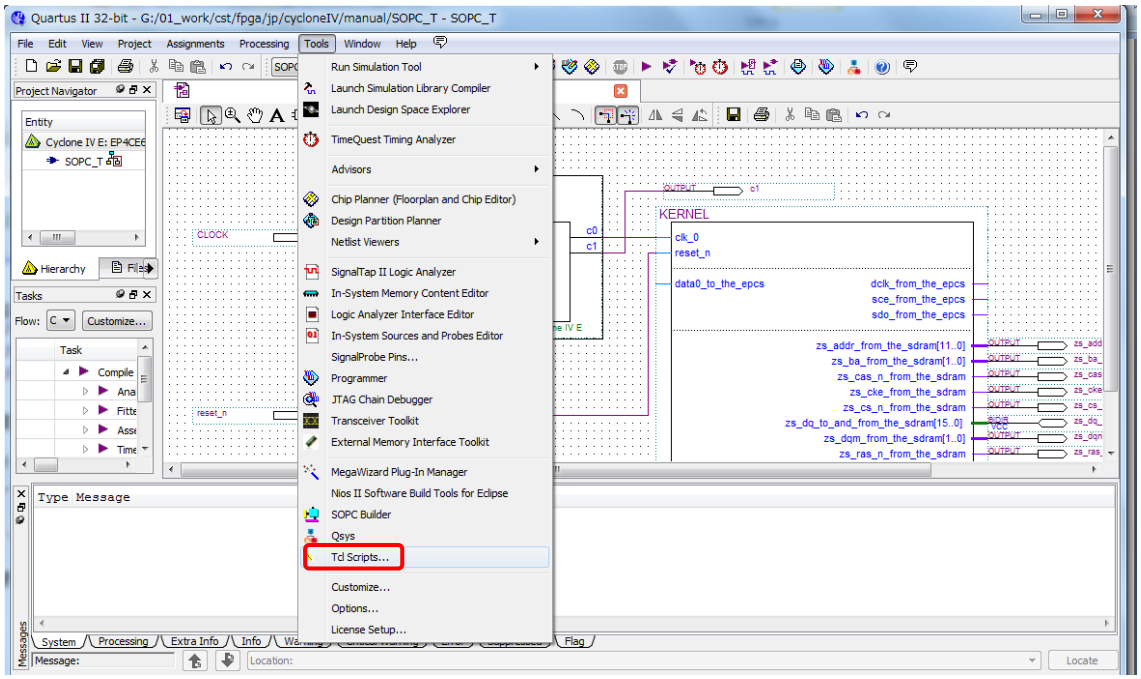
Project→Add/Remove Files in Project



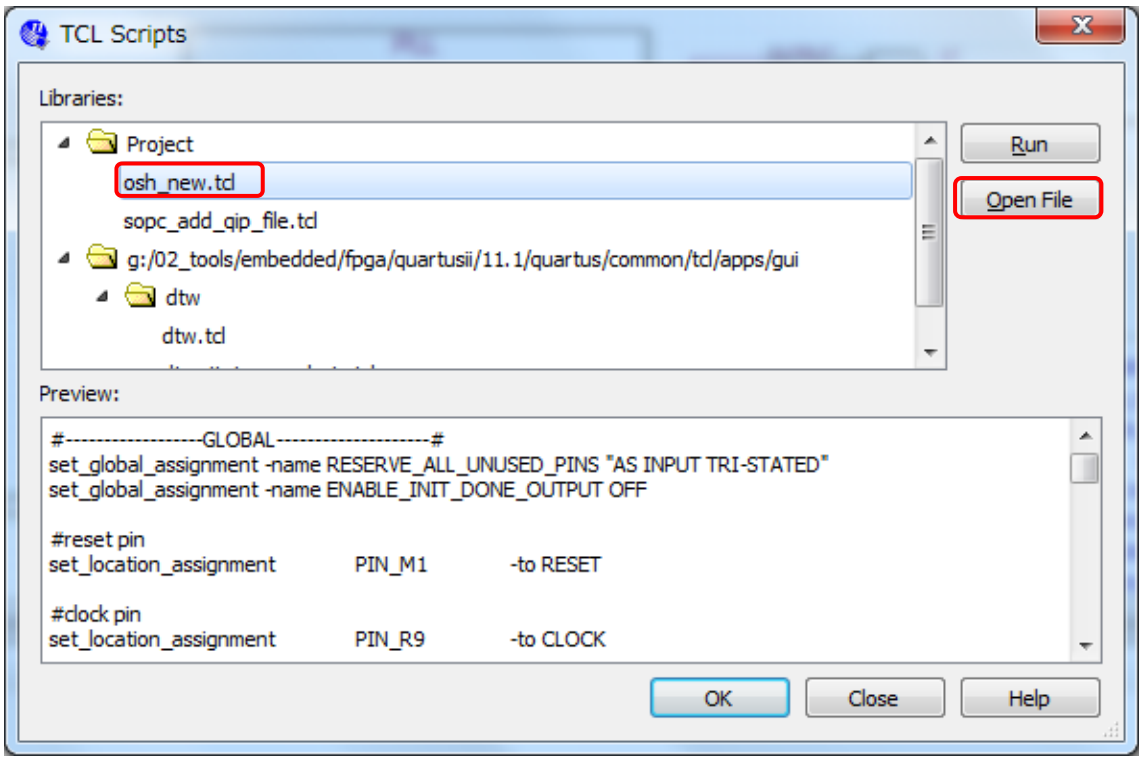
TCL ファイルを特定してから「Add」ボタンを押下



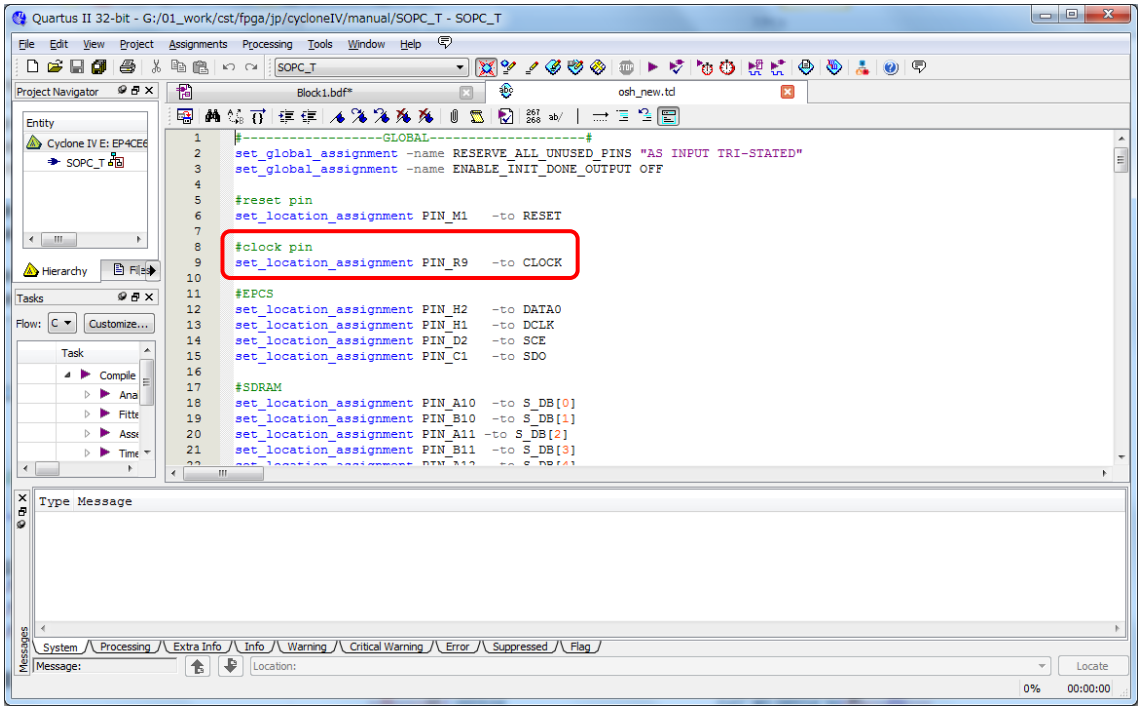
Tools->Tcl Scripts



TCL ファイルを選択して「Open File」ボタンをクリック



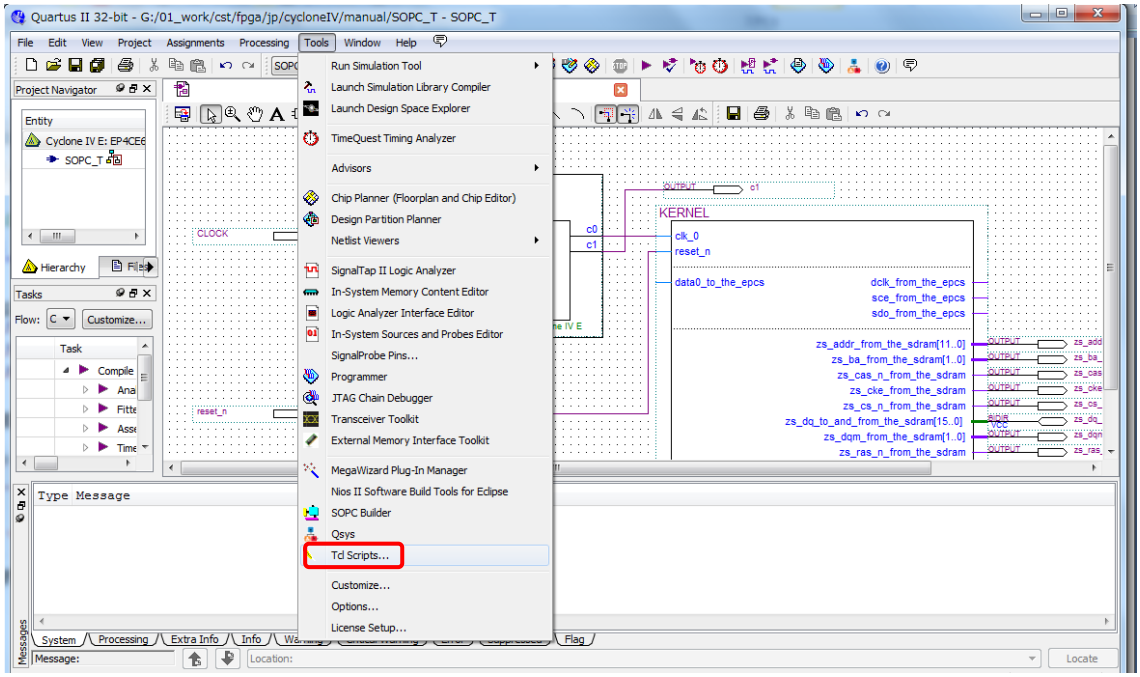
TCL ファイルを Quartus で見られます。



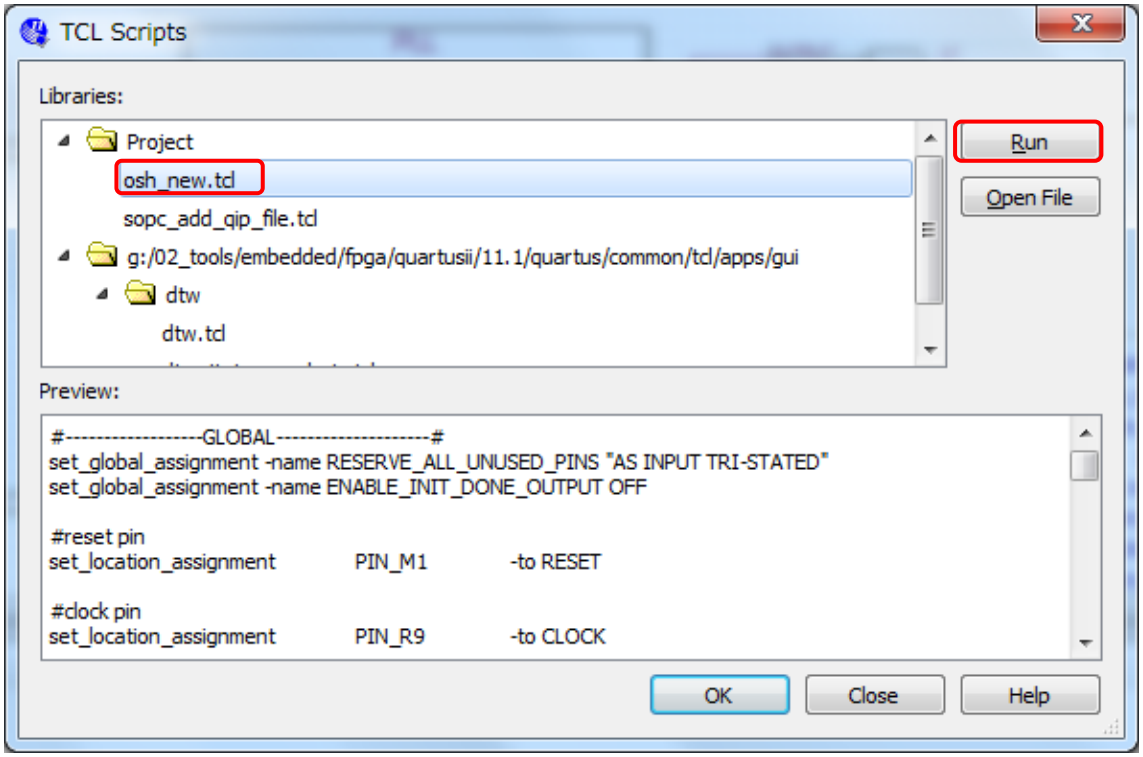
クロックのピンの名前は既に「CLOCK」に修正され、TCL ファイルと合っています。

では、TCL ファイルを実行してみましょう。

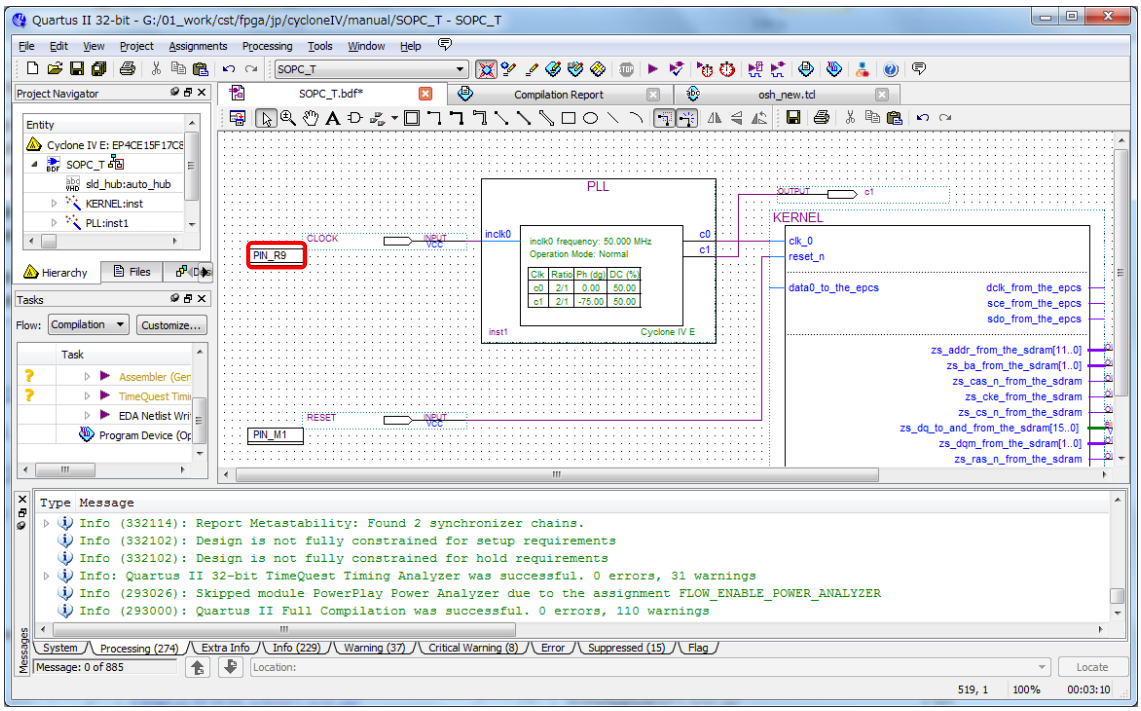
Tools->Tcl Scripts



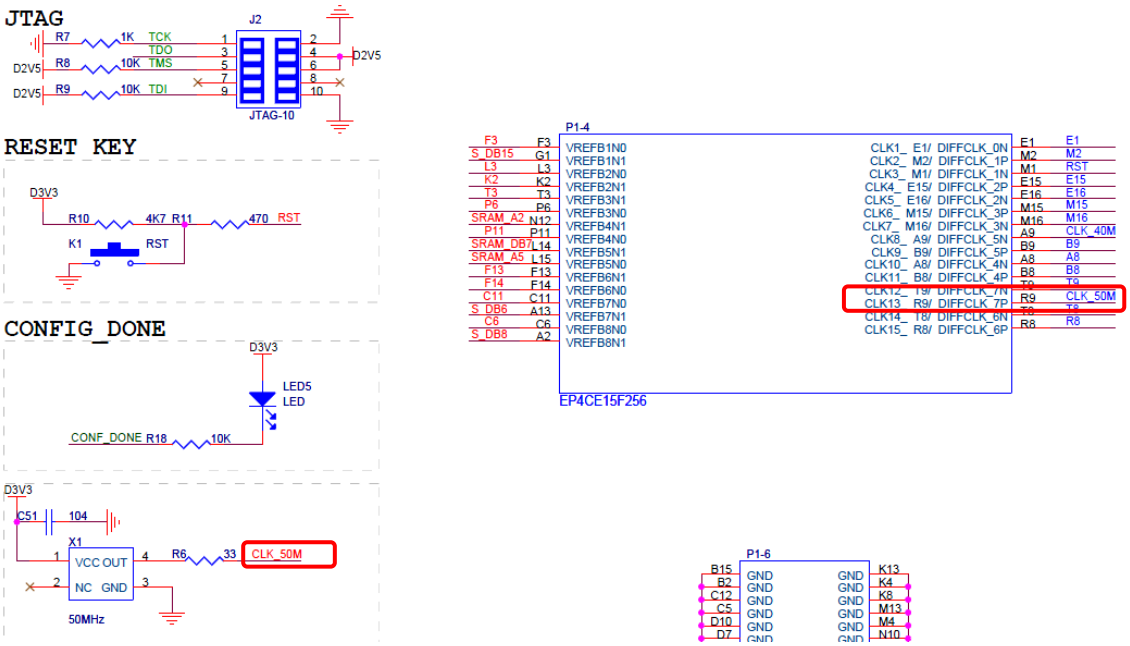
TCL ファイルを選択して「Run」ボタンをクリック



下図のようにクロックのピンが自動に「PIN_R9」を割り当てられます。



FPGA ボードの回路図により、50MHZ の水晶発振器が FPGA の R9 ピンと繋がります。



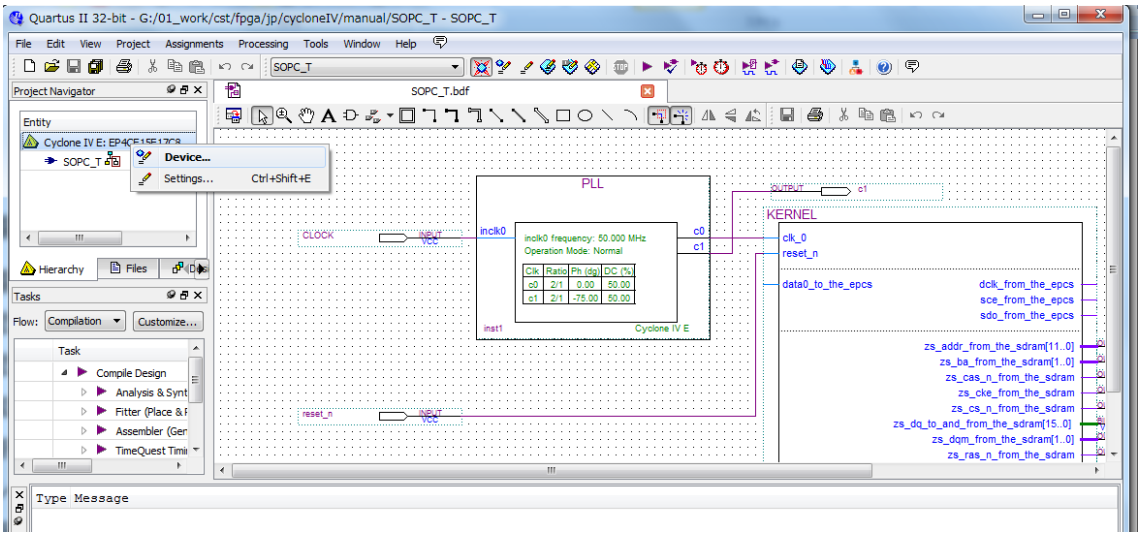
TCL ファイルを参照しながら、他のピンは全て同じ様に名前を修正してください。修正後、もう一度 TCL スクリプトファイルを実行しましょう。

弊社は全て FPGA ボードのピンに関わる TCL ファイルを提供しますので、簡単に FPGA ボードのピンを割り当てる事が出来ます。

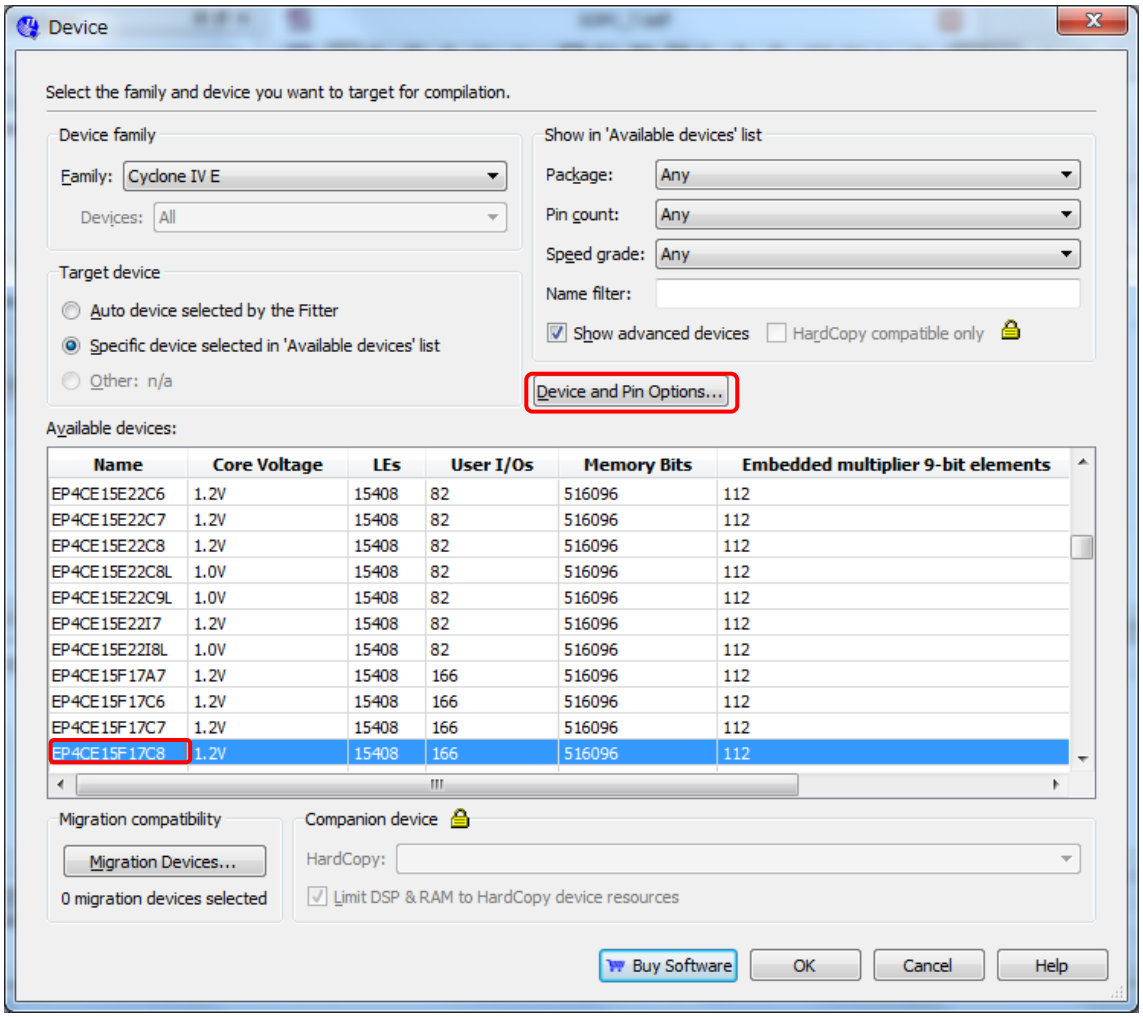
3.6 プロジェクトのコンフィグ

プロジェクトをコンフィグする前、プロジェクトの全てファイルを保存しましょう。

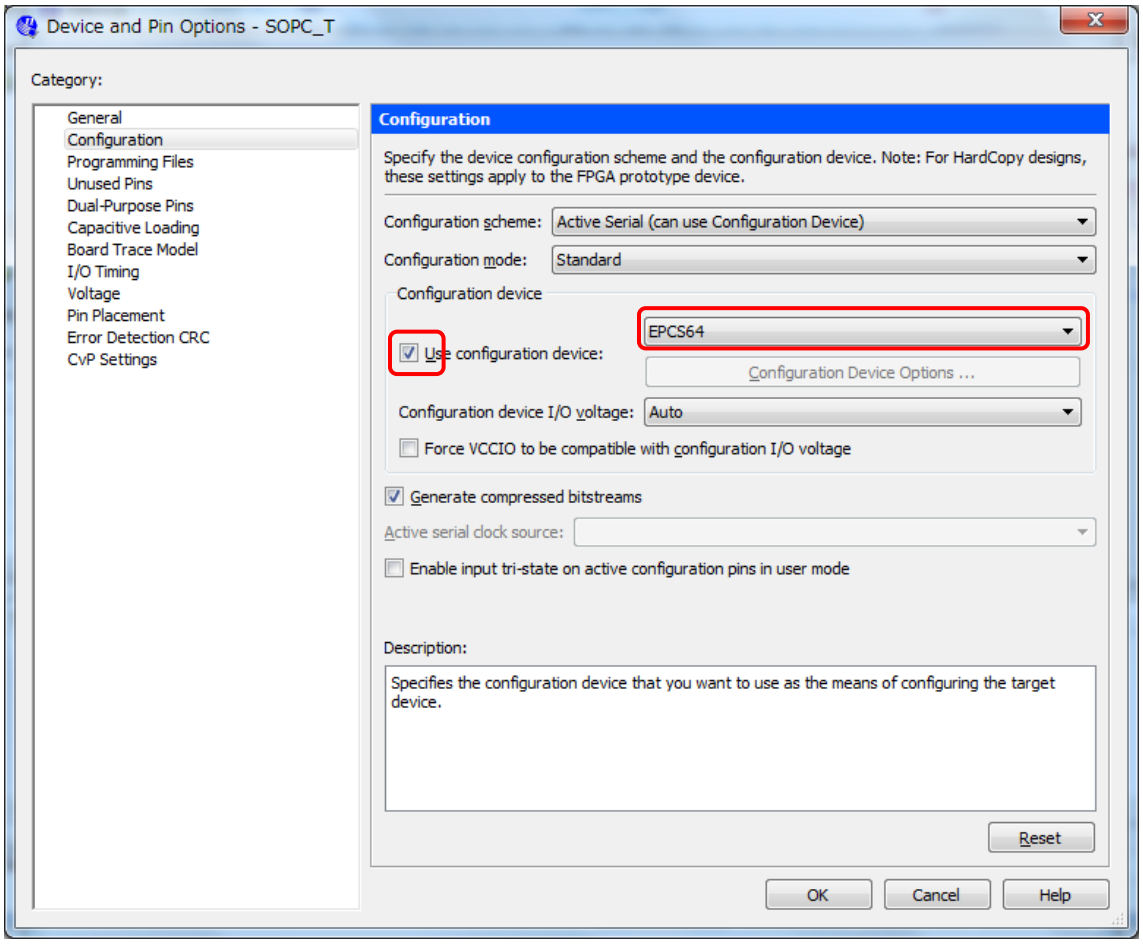
左側の「Project Navigator」ウィンドウに Cyclone IV E:EP4CE15F17C8 を選択し、右クリックして「Device」メニューを押下



「Device and Pin Options」をクリック



左側の「Configuration」を選択、「Use configuration device」をチェック、「EPCS64」を選択



次は左側の「Dual-Purpose Pins」を選択、

「DCLK」:「Use as regular I/O」

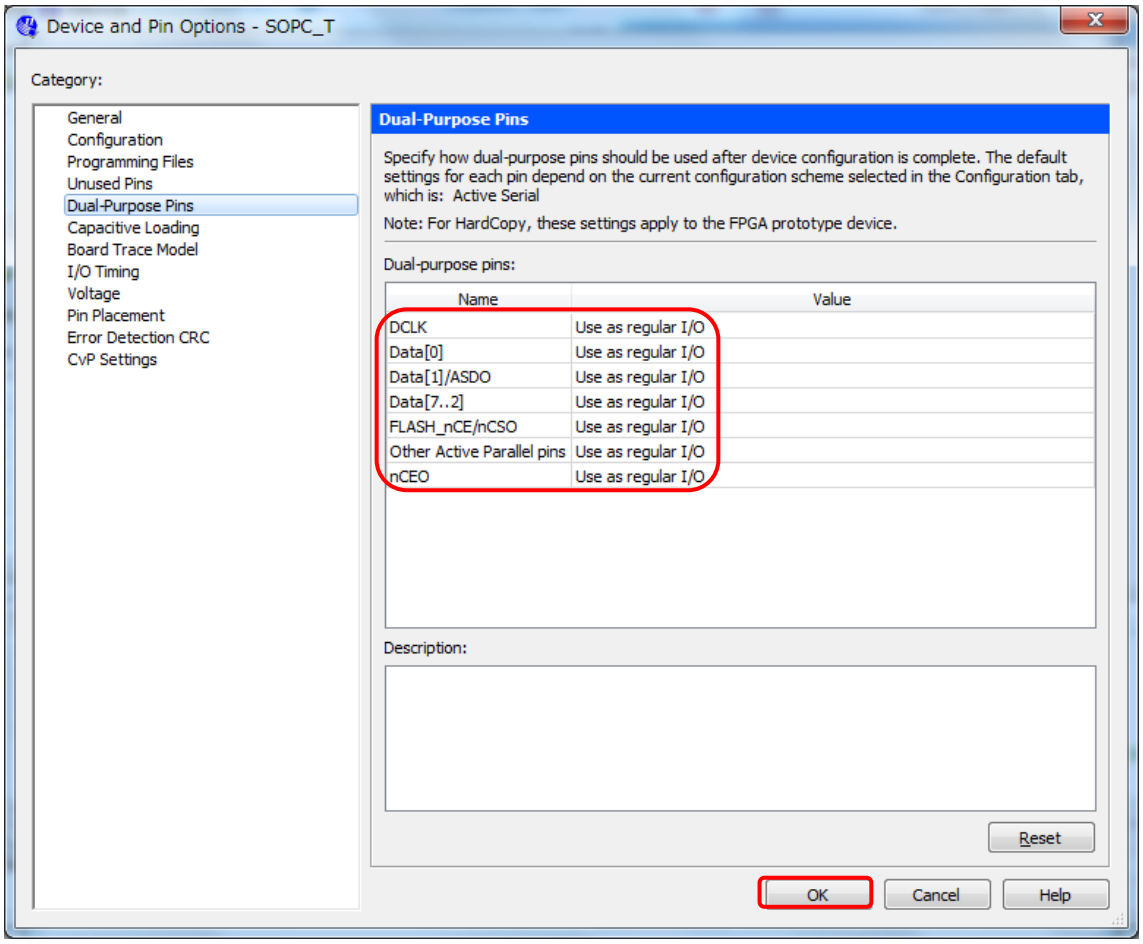
「Data[0]」:「Use as regular I/O」

「Data[1]/ADSO」:「Use as regular I/O」

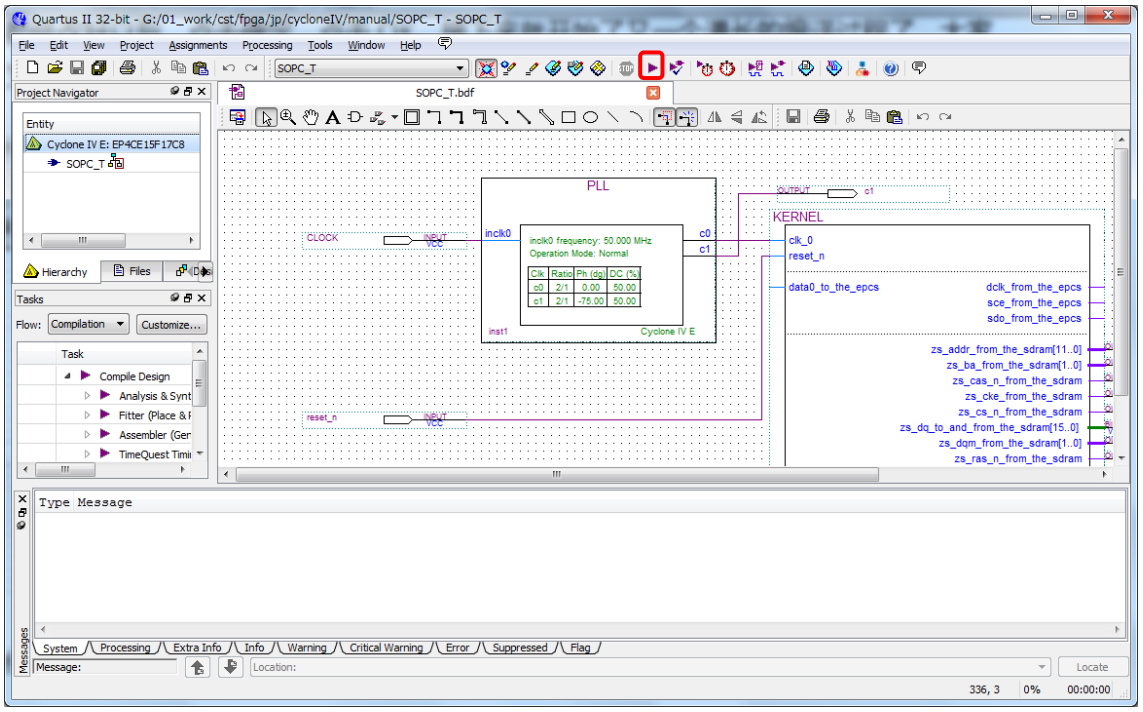
「Flash_nCE/nCSO」:「Use as regular I/O」

「nCEO」:「Use as regular I/O」を変更

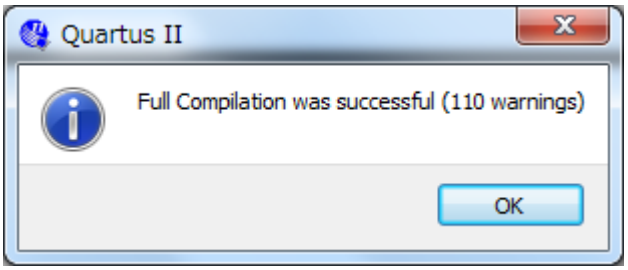
変更したら、連続二回の「OK」ボタンをクリックしてコンフィグレーションを完成させましょう。



コンフィグ出来たら、下図のボタンを押しプロジェクトをコンパイルしましょう。



長いコンパイル時間が掛ります、最後、下記のようなメッセージが出てきましたら、成功にコンパイルできた事です。「OK」ボタンをクリックしてコンパイルが完了しました。

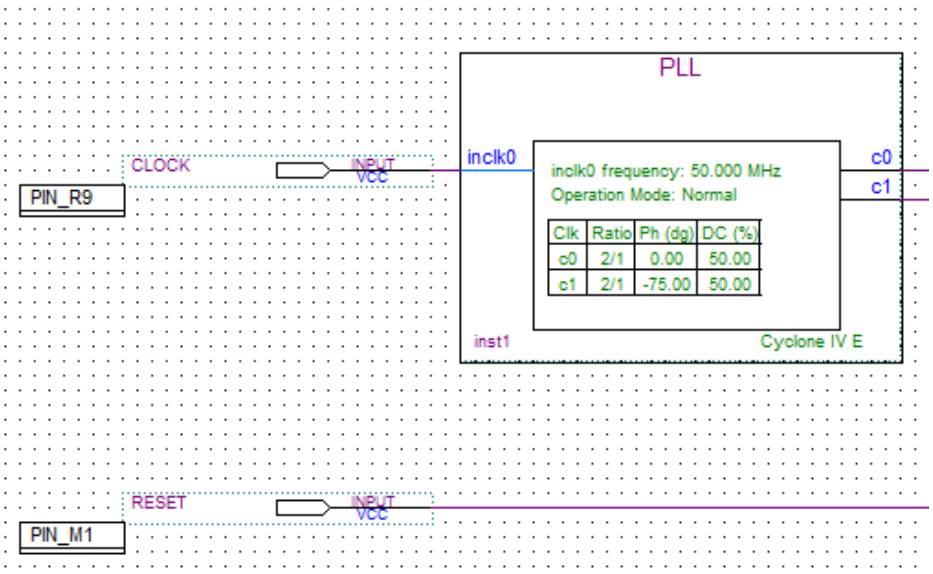


コンパイルレポートにより、どのぐらいリソースを使われるかを分かります。
22%の LE しか使われません。(まだ全然余裕がありますね)

Flow Summary

Flow Status	Successful - Thu Jan 19 23:54:17 2012
Quartus II 32-bit Version	11.1 Build 173 11/01/2011 SJ Web Edition
Revision Name	SOPC_T
Top-level Entity Name	SOPC_T
Family	Cyclone IV E
Device	EP4CE15F17C8
Timing Models	Final
Total logic elements	3,386 / 15,408 (22 %)
Total combinational functions	2,917 / 15,408 (19 %)
Dedicated logic registers	2,037 / 15,408 (13 %)
Total registers	2105
Total pins	40 / 166 (24 %)
Total virtual pins	0
Total memory bits	55,424 / 516,096 (11 %)
Embedded Multiplier 9-bit elements	4 / 112 (4 %)
Total PLLs	1 / 4 (25 %)

最後、全てのピンが正しく割り当てられるかどうかを確認してください。
SOP_T.bdf をクリックして確認しましょう。



上記ピンが割り当てられます、OK です。他のピンもチェックしてください。

ここまで Quartus II でハードウエア開発内容が全て終わりました。

第四章 ソフトウエア開発

第三章はハードウエア開発内容を説明しました、本章から NIOS II のソフトウエア開発を説明します。NIOS II IDE 11.1 でプロジェクトの作成からコンフィグ、コンパイル、ダウンロードまでの内容が含まれます、全て画像と合わせて説明しますので、NIOS II 開発経験がなくてもスムーズにソフトウエア開発を実施できるだろう。

参考資料：

■オンライン資料： Nios II プロセッサ

http://www.altera.co.jp/literature/lit-nio2.jsp

■NIOS II ソフトウェア開発ハンドブック

http://www.altera.co.jp/literature/hb/nios2/n2sw_nii5v2_j.pdf

4.1 概要

ハードウエア開発に基づき、まず、NIOS II 開発の流れを説明し、次は、NIOS II IDE の簡単な使い方も説明します、FPGA の始めた方としても分かりやすくにするようにします。

NIOS II IDE は Eclipse IDE フレームワークに基づき IDE 開発ツールです、たくさんの機能が含まれ、JAVA 開発の経験者としてよく知っていると思う。

NIOS II IDE 機能と特徴を簡単に紹介しましょう。

■GNU 開発ツール：Linux　OS を知っている人々はそれは見知らぬ世界ではない、それは標準の GCC コンパイラ、リンク、アセンブラ及び makefile ツールを含みます。

■GDB に基づきデバッグツール：エミュレーションとハードウエアデバッグ機能を含む

■ハードウェア抽象化レイヤ HAL（Hardware Abstraction Layer）を統合する

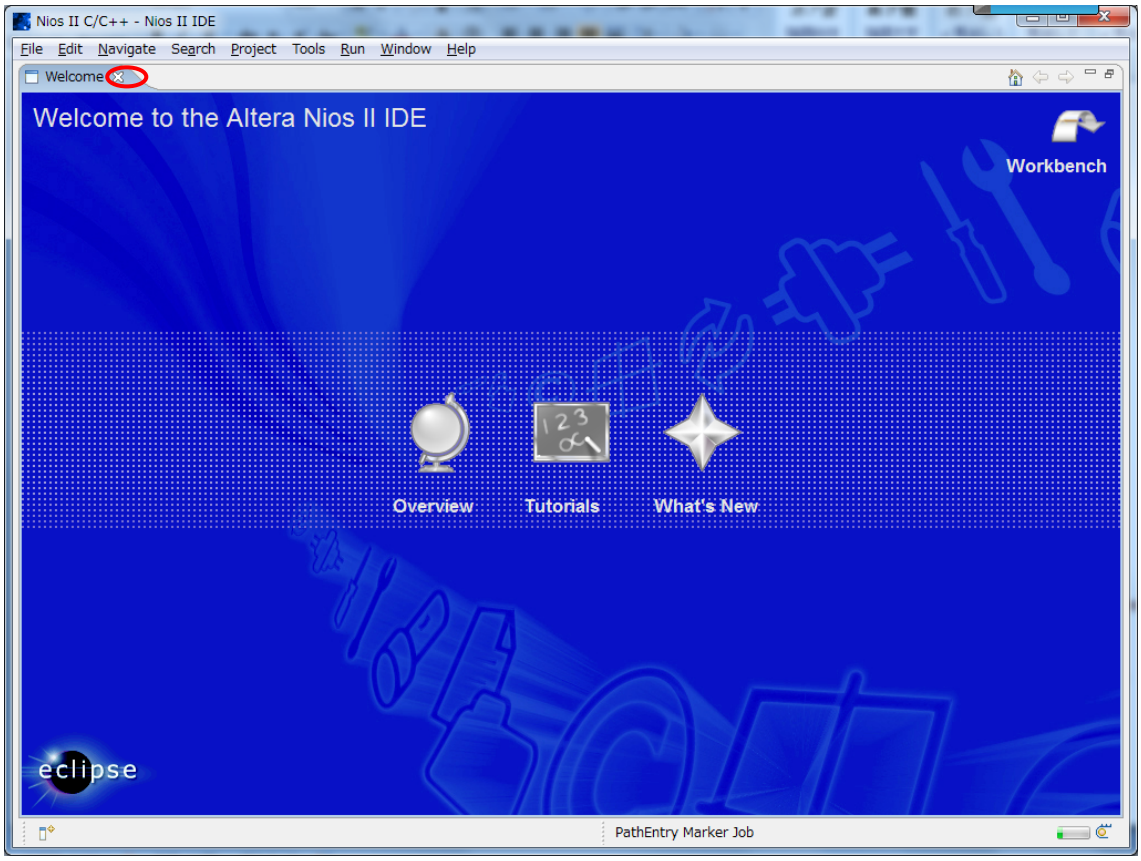
■MicroChip/OS II と LwTCP/IP のプロトコルスタックをサポート

■Flash ダウンロード（Flash Programmer 和 Quartus II Programmer）をサポート

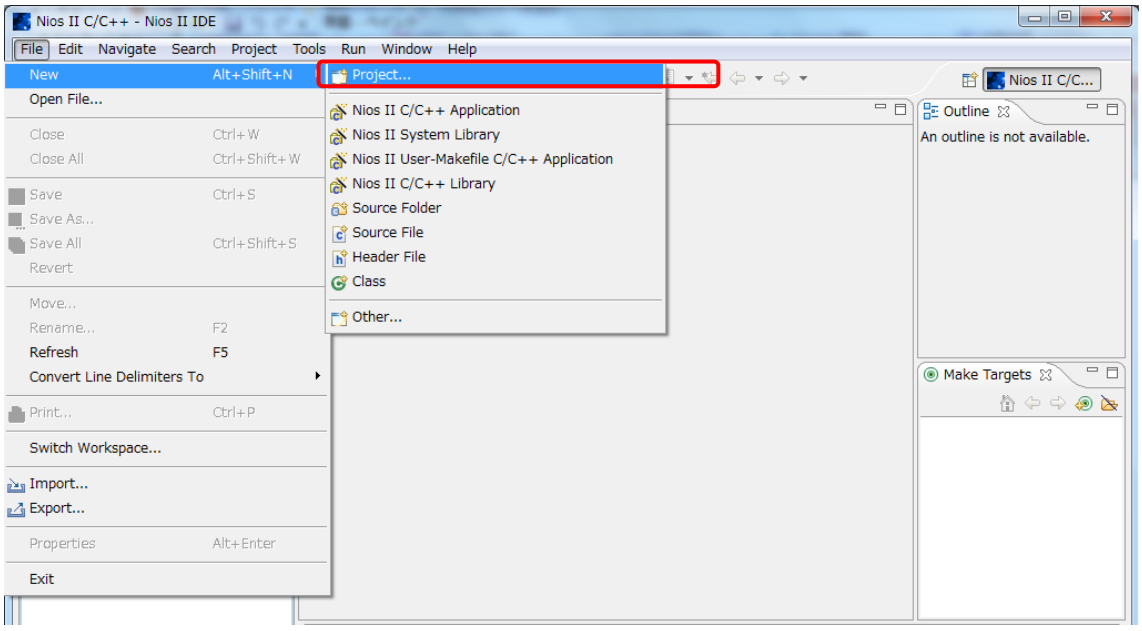
4.2 プロジェクト作成

NIOS II IDE11.1 を立ち上げ

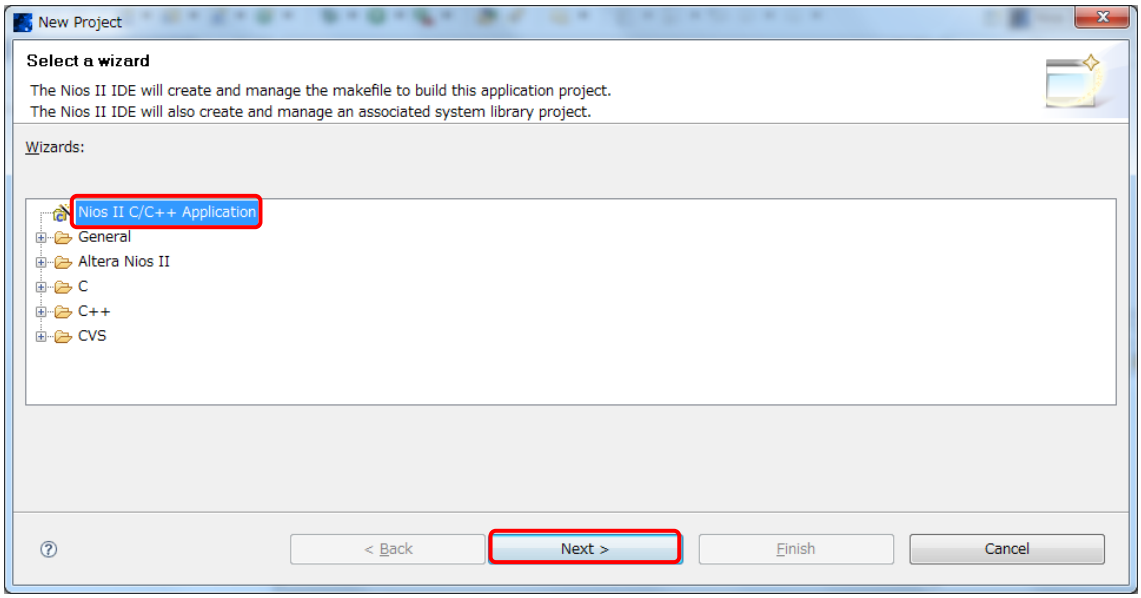
Welcome 画面を閉じプロジェクトを作成して行きましょう。



File->New->Project



「Nios II C/C++ Application」を選べ、「Next」をクリック

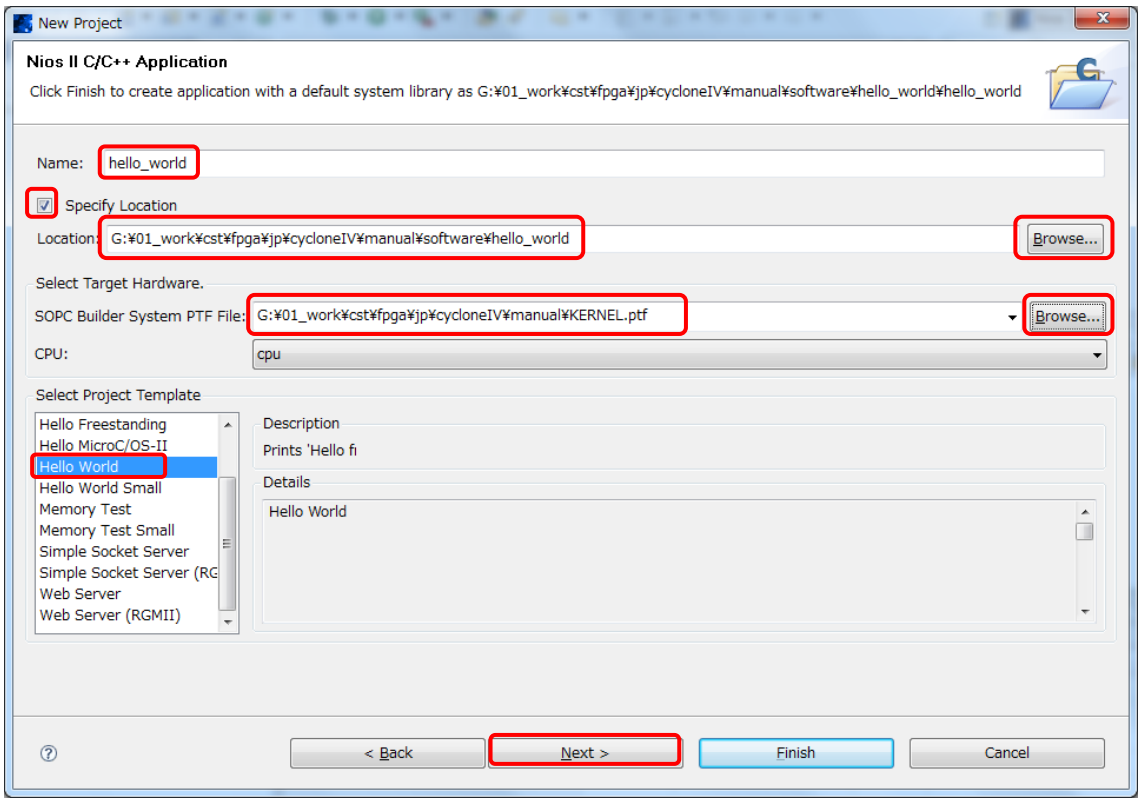


Project name：hello_world

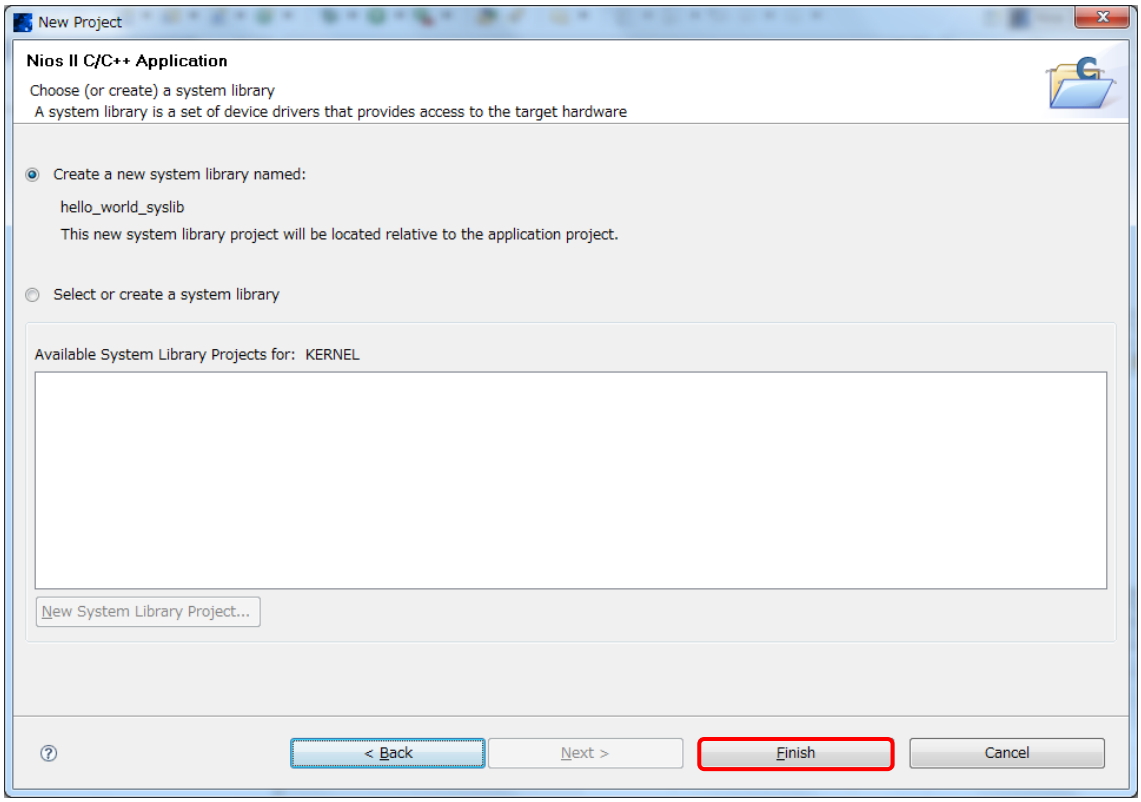
Location：お好きな場所を指定、変更されない場合、Nios II インストール先にワークスペースを置く

SOPC Information File name：第三章に作ったソフトマクロの場所を指定してください。

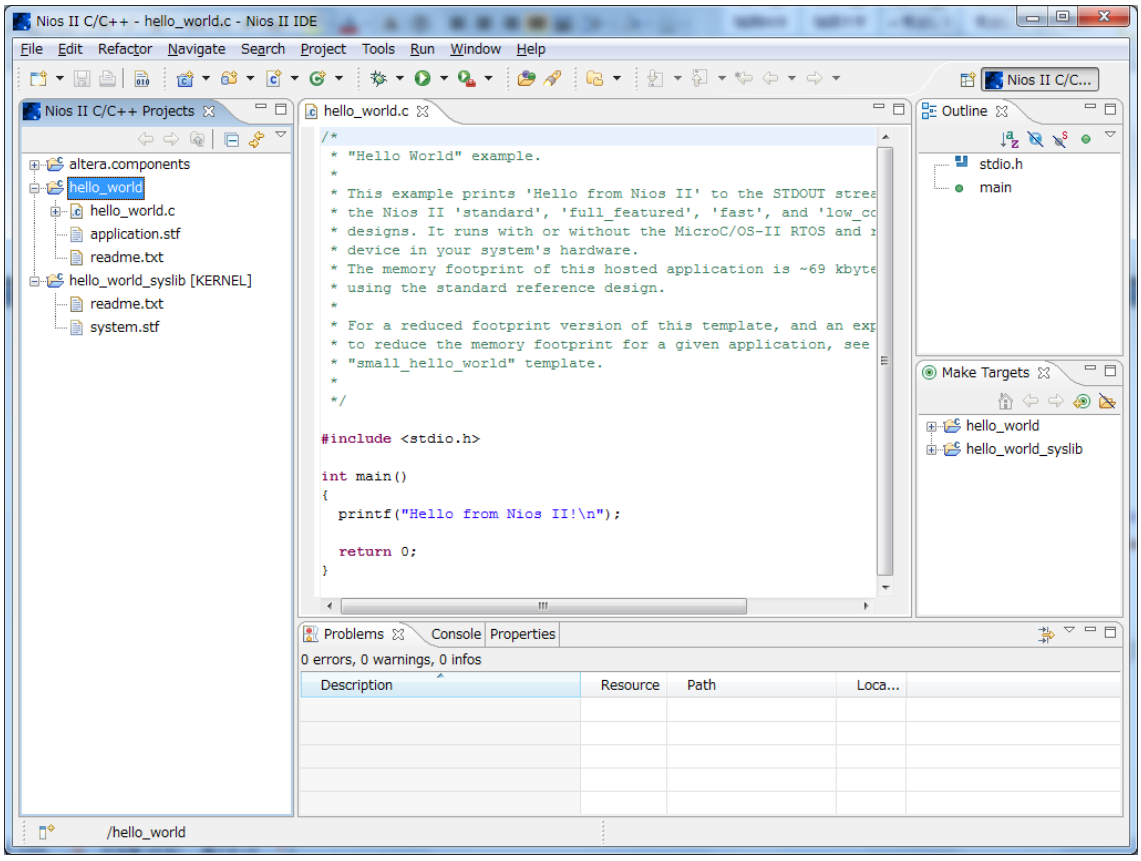
他のオプションはデフォルトのまま「Next」ボタンをクリック



デフォルトのまま「Next」をクリックして、暫く待ち、新規プロジェクトを作成完了。

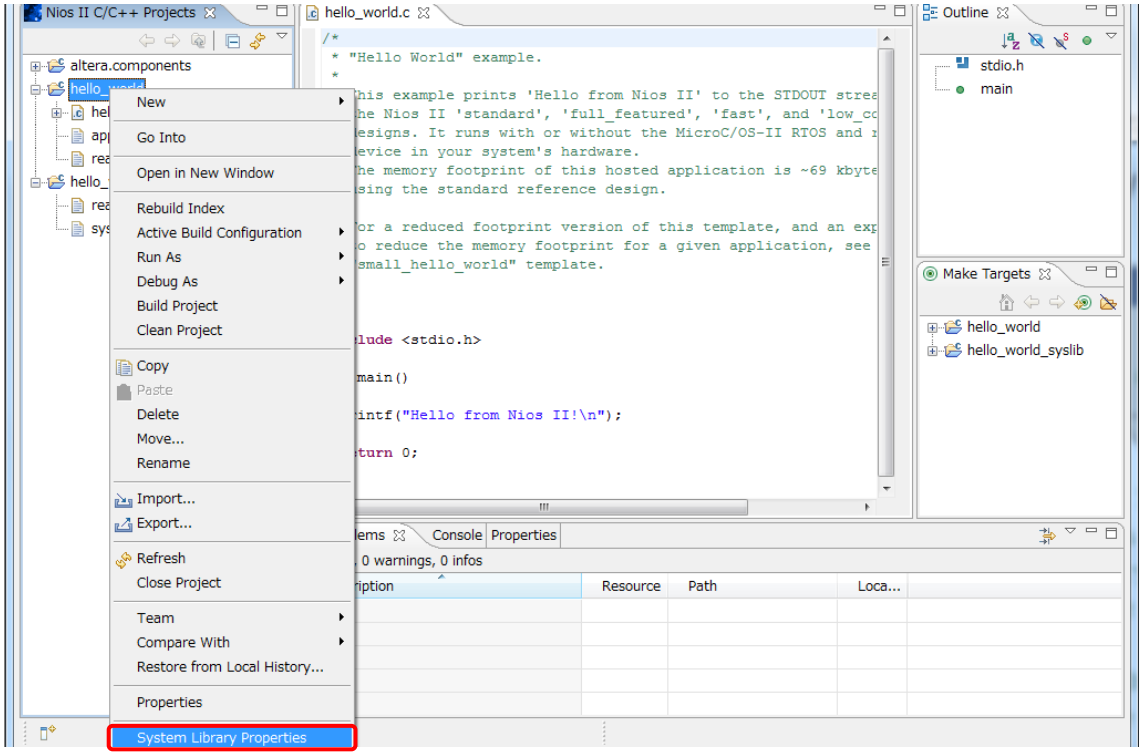

New Project
Nios II C/C++ Application
Choose (or create) a system library
A system library is a set of device drivers that provides access to the target hardware
Create a new system library named:
hello_world_syslib
This new system library project will be located relative to the application project.
Select or create a system library
Available System Library Projects for: KERNEL
New System Library Project...
< Back
Next >
Finish
Cancel

作成完了後の様子

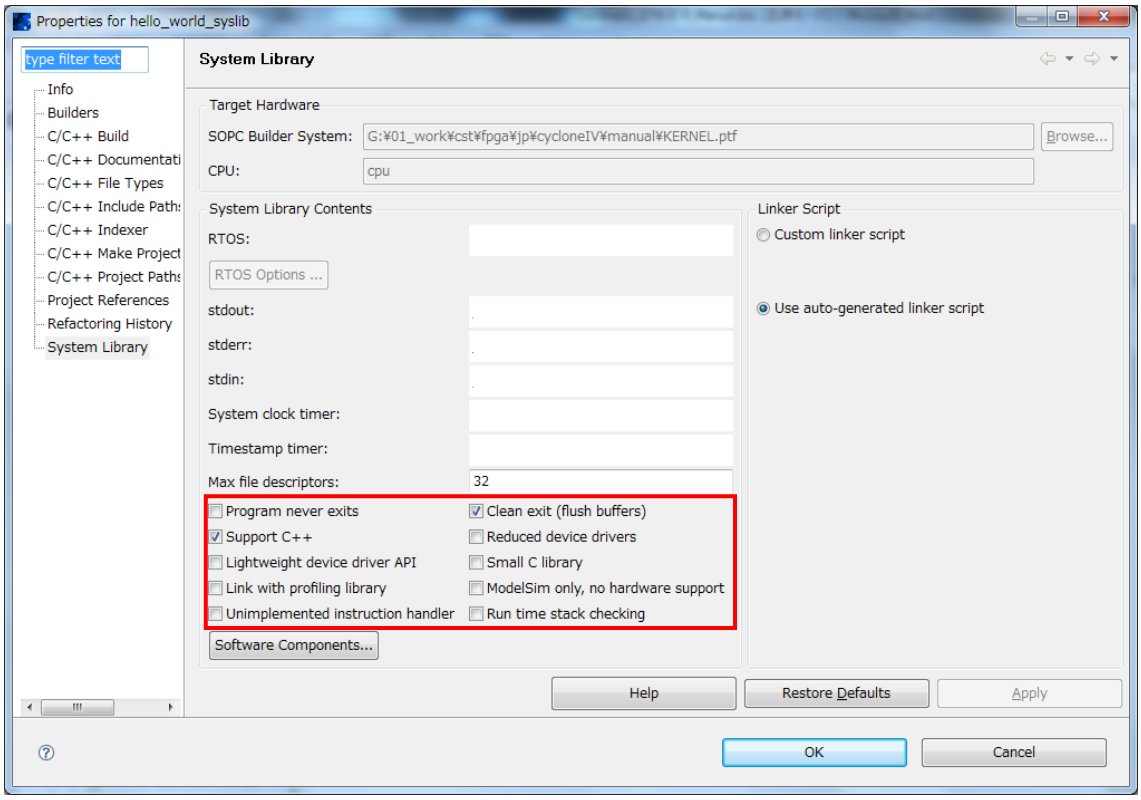

Nios II C/C++ - hello_world.c - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++ Projects
altera.components
hello_world
hello_world.c
application.stf
readme.txt
hello_world_syslib [KERNEL]
readme.txt
system.stf
hello_world.c
/*
* "Hello World" example.
*
* This example prints 'Hello from Nios II' to the STDOUT stre
* the Nios II 'standard', 'full_featured', 'fast', and 'low_co
* designs. It runs with or without the MicroC/OS-II RTOS and
* device in your system's hardware.
* The memory footprint of this hosted application is ~69 kbyte
* using the standard reference design.
*
* For a reduced footprint version of this template, and an exp
* to reduce the memory footprint for a given application, see
* "small_hello_world" template.
*
*/
#include <stdio.h>
int main()
{
printf("Hello from Nios II!\n");
return 0;
}
Outline
stdio.h
main
Make Targets
hello_world
hello_world_syslib
Problems
Console
Properties
0 errors, 0 warnings, 0 infos
Description
Resource
Path
Loca...
/hello_world

プロジェクトコンフィグ：

Nios II メイン画面に左側の「Project Explorer」→「hello_world」を右クリック→「System Library Properties」をクリック



プロパティ内容を確認しましょう。（デフォルトのままで OK）



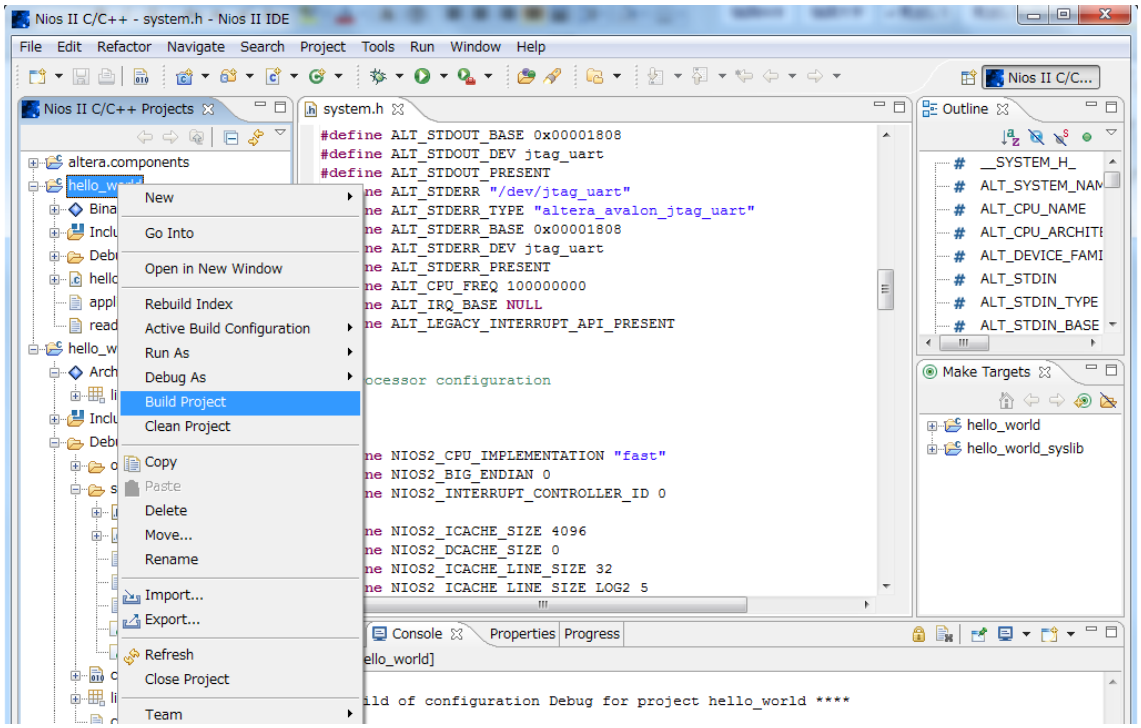
Nios II メイン画面に左側の「Project Explorer」→「hello_world_syslib」を右クリック→「System Library Properties」をクリック

4.3 コンパイル

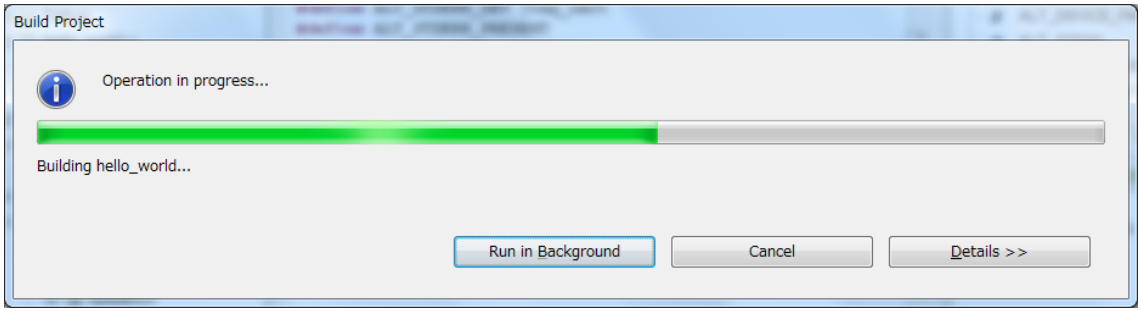
これからコンパイルしましょう。コンパイル途中で非常に重要なファイル「system.h」を生成されますため、最初のコンパイルは時間が掛ります。生成されたファイルはこの前に作ったソフトマクロにより作成されたものです、即ち、「system.h」の内容はソフトマクロと一致するべきです、ソフトマクロが変わったら、再度コンパイル必要です、新しい「system.h」も生成されます。

「hello_world」プロジェクト名に右クリック、「Build Project」を押下、コンパイルは始まります。

（或いはコンパイルショットキー：Ctrl+b を同時押下）

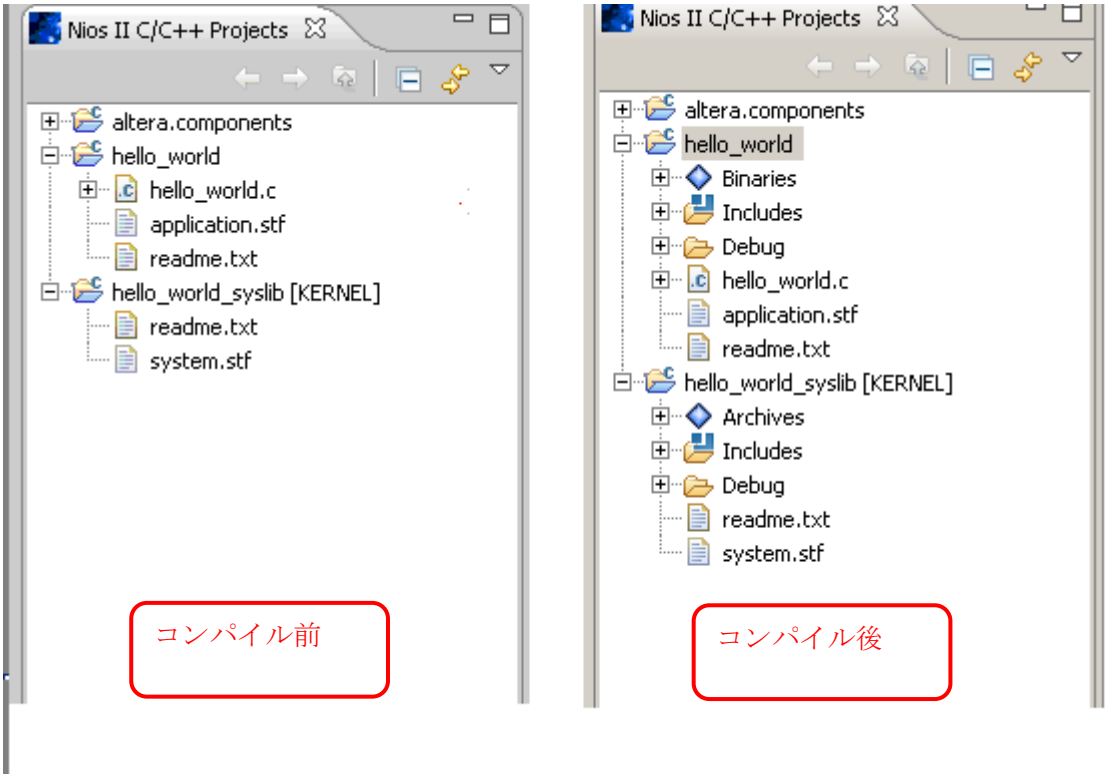


下記の様画面が出来ます、暫く待ち



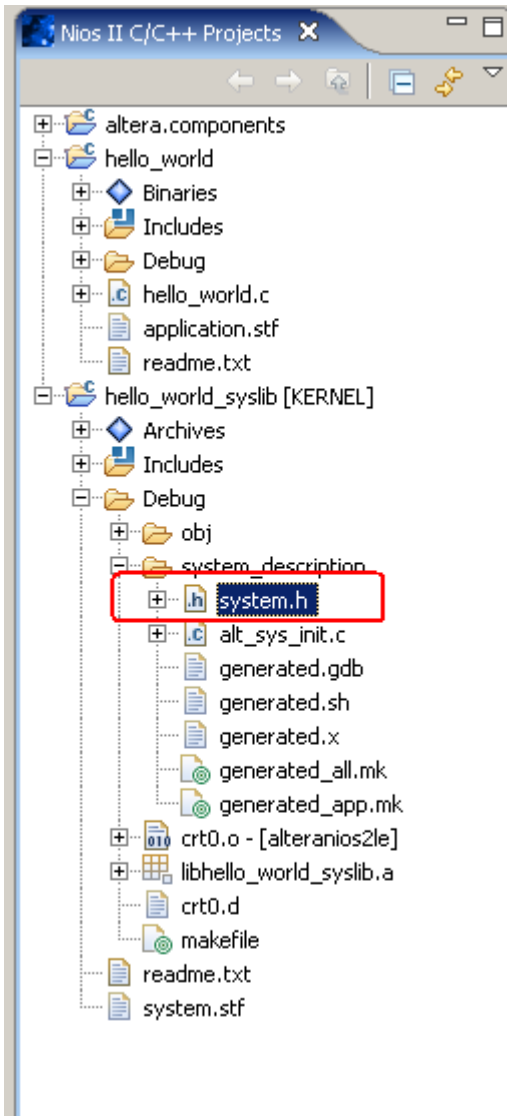
コンパイルが完了したら、以下の画面が出ます。

コンパイル前後の違いを比較



「system.h」ファイルを開き内容を見ましょう。

（hello_world_syslib/Debug/system_description/system.h）



JTAG を例として簡単に説明します。(system.h ファイルから抜粋)
注意を頂きたい所は JTAG_UART_BASE と JTAG_UART_IRQ です、
JTAG_UART_BASE：JTAG のベースアドレス
JTAG_UART_IRQ：割り込み番号
他にはコンフィグ情報です、後程説明しますので、今省略です。それ以外、SDRAM、EPCS 等にもアドレスがあります､これらのアドレスを使って IP コアのレジスタを操作します｡NIOS の強さはここで見えるでしょう、何か足りない場合、設計者から IP コアを追加・修正できます、さらにハードウエアモジュールも作成できます。

```
/*
 * jtag_uart configuration
 *
 */
```

```
#define JTAG_UART_NAME "/dev/jtag_uart"
#define JTAG_UART_TYPE "altera_avalon_jtag_uart"
#define JTAG_UART_BASE 0x00001808
#define JTAG_UART_SPAN 8
#define JTAG_UART_IRQ 1
#define JTAG_UART_IRQ_INTERRUPT_CONTROLLER_ID 0
#define JTAG_UART_WRITE_DEPTH 64
#define JTAG_UART_READ_DEPTH 64
#define JTAG_UART_WRITE_THRESHOLD 8
#define JTAG_UART_READ_THRESHOLD 8
#define JTAG_UART_READ_CHAR_STREAM ""
#define JTAG_UART_SHOWASCII 1
#define JTAG_UART_RELATIVEPATH 1
#define JTAG_UART_READ_LE 0
#define JTAG_UART_WRITE_LE 0
#define JTAG_UART_ALTERA_SHOW_UNRELEASED_JTAG_UART_FEATURES 1
#define ALT_MODULE_CLASS_jtag_uart altera_avalon_jtag_uart
```

4.4 実行

NIOS II IDE で二つ方法から実行できます、一つは直接ハードウエアでオンラインエミュレーション、もう一つはソフトウエアエミュレーションです。

１．ハードウエアでのオンラインエミュレーション（デバッグもできます）

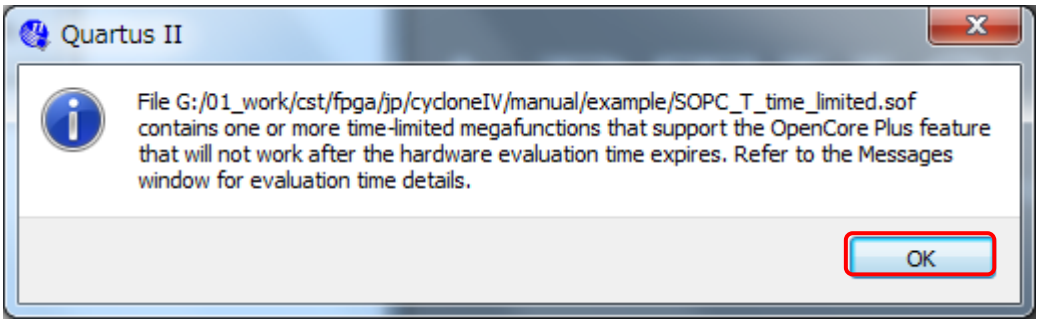
この場合、開発ボードと USB ダウンロードケーブルというハードウエア必要です。まず、USB ダウンロードケーブルは開発ボードの JTAG インタフェースと接続にし、次はボードの電源を入れます。電源を入れてからプログラムを JTAG でボードにダウンロード必要です、詳しいダウンロード方法は「第五章 プログラムダウンロード」をご参照ください。

■プログラムダウンロード：

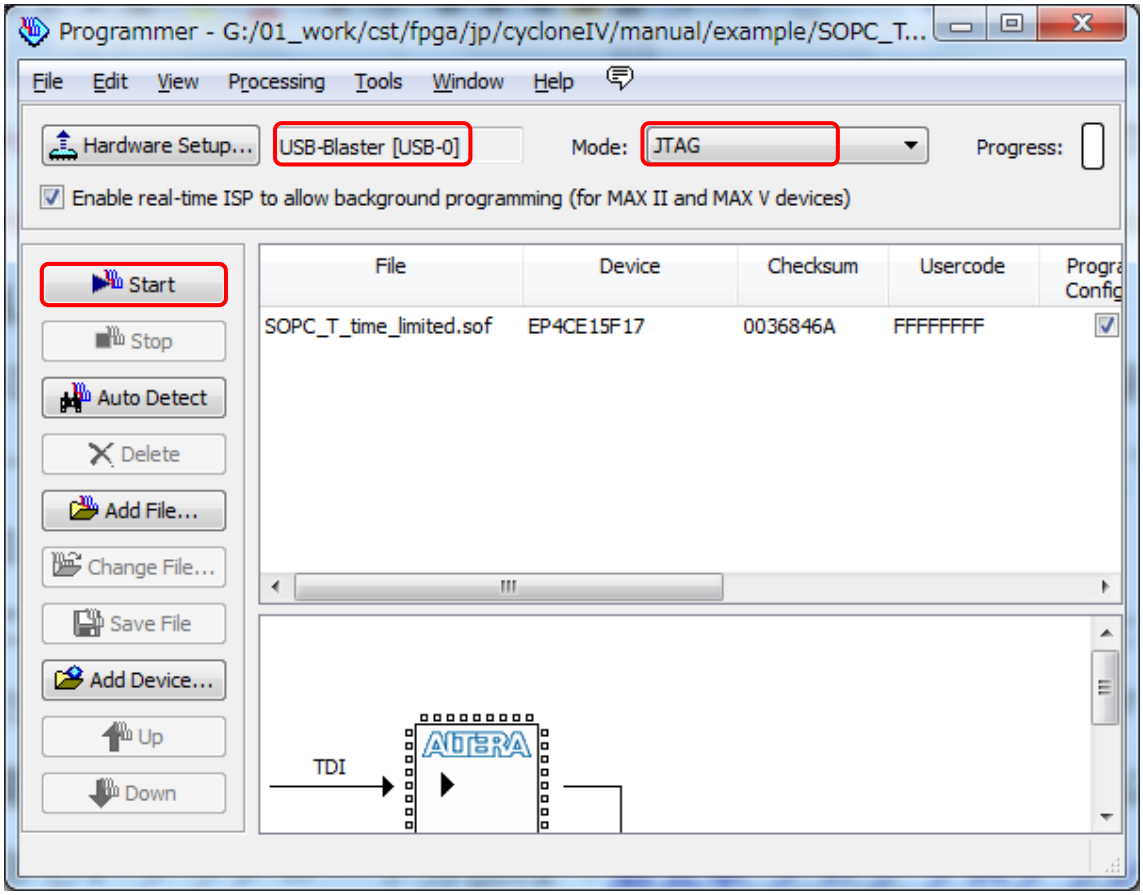
Nios II IDE からメニュー「Tools」→「Quartus II Programmer」をクリック

※Quartus II を起動させ、前章作ったプロジェクト「SOPC_T.qpf」を開き、メニュー「Tools」→「Programmer」をクリックしても同じです。

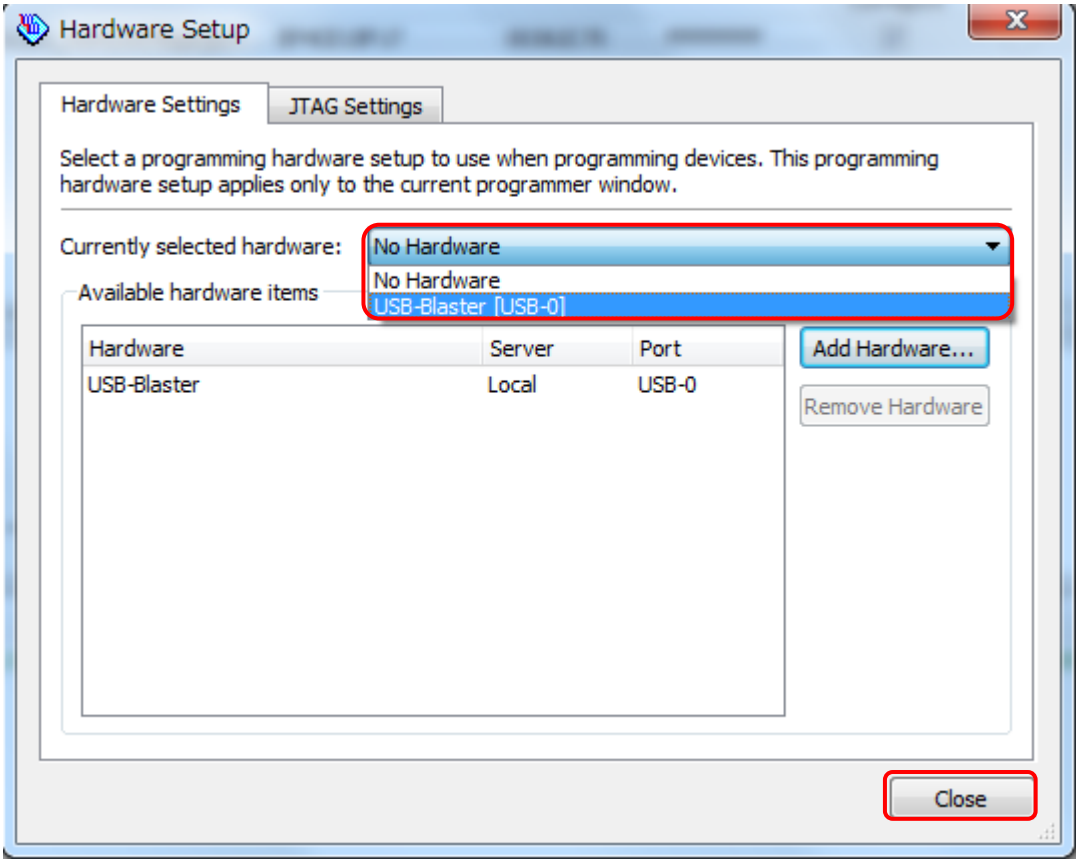
下記のような情報が出てきます。正式製品なら、アルテラ社からライセンスが必要です。評価の場合は、そのまま「OK」ボタンを押します。



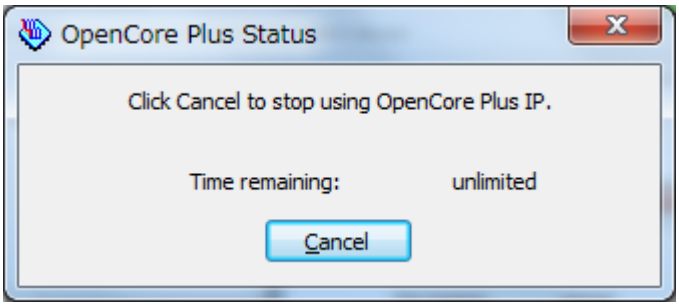
「0k」を押したら、下記画面が表示されます。
赤□の所を確認してから「Start」ボタンを押下



※ハードウエア「USB-Blaster」が表示されない場合、左の「Hardware Setup」をクリック、下記の様に選択してください。

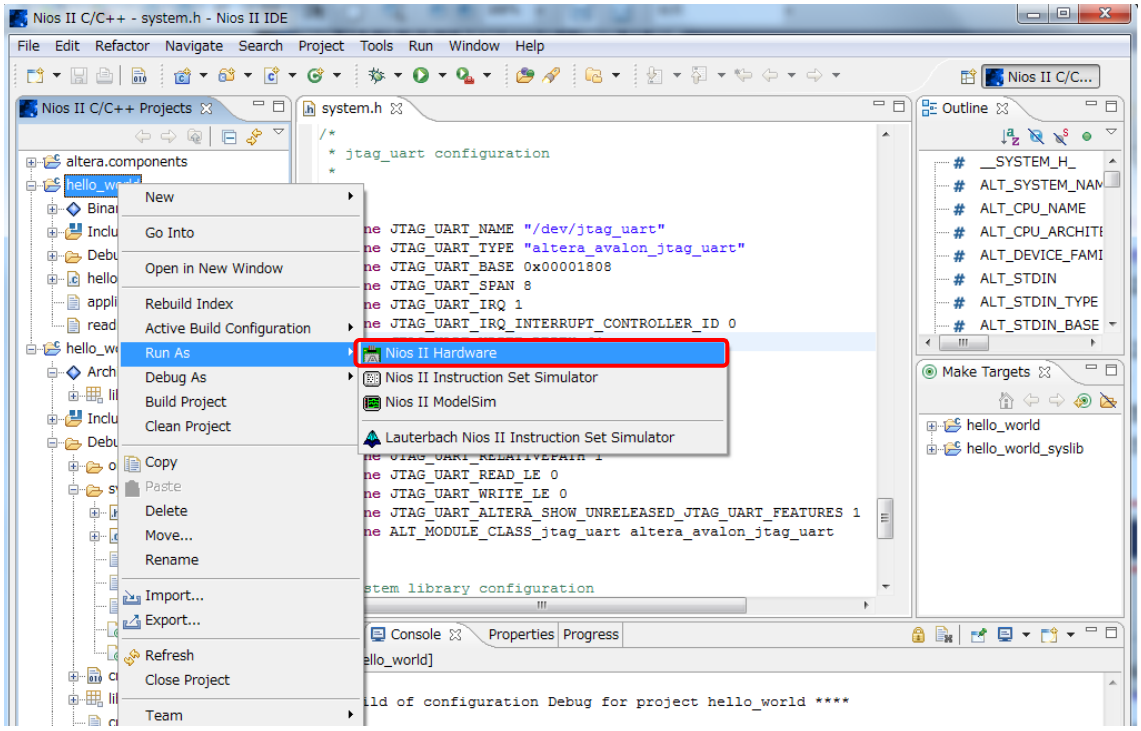


書込みが完了したら、下記画面が表示されます、「Cancel」をクリックしないでください。

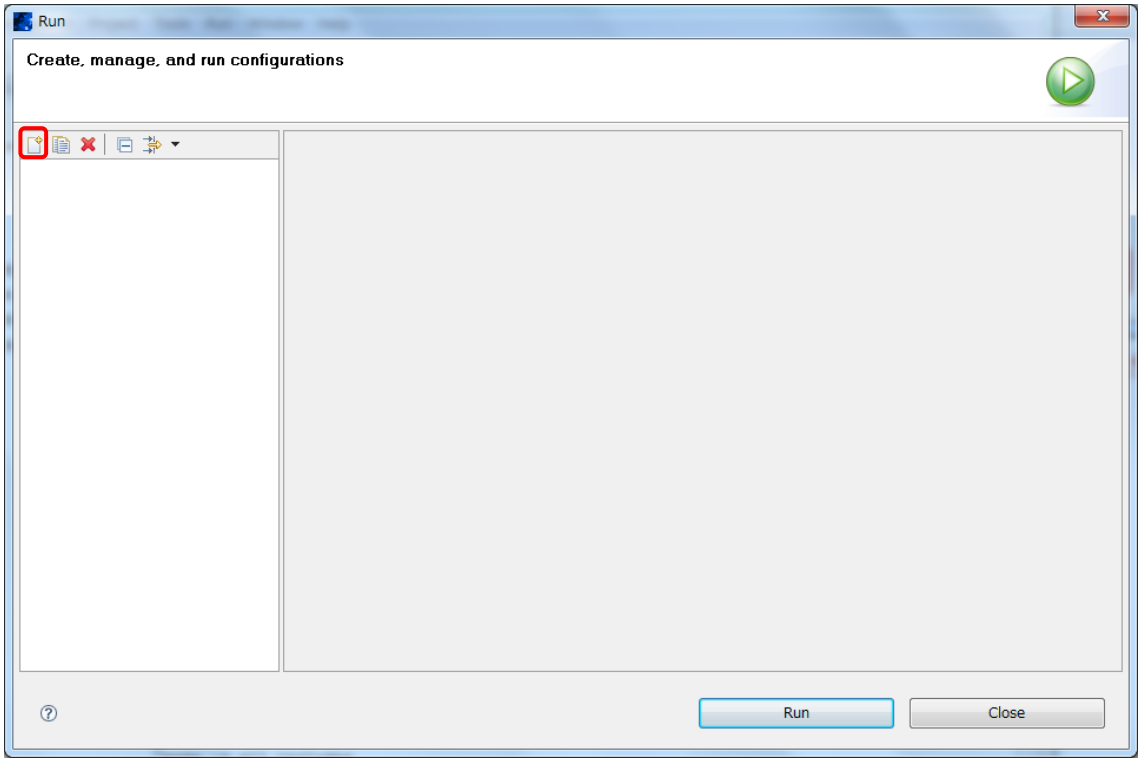


■NIOS II IDE から下記のように操作しましょう。

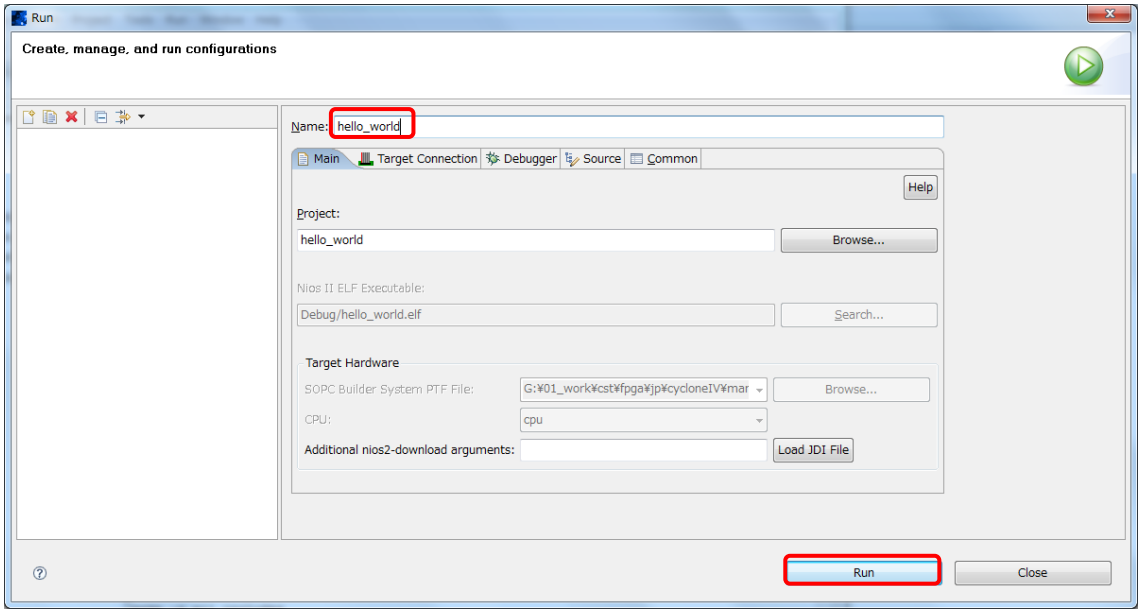
「hello_world」プロジェクト名に右クリック、「Run　As」→「Nios II Hardware」をクリック



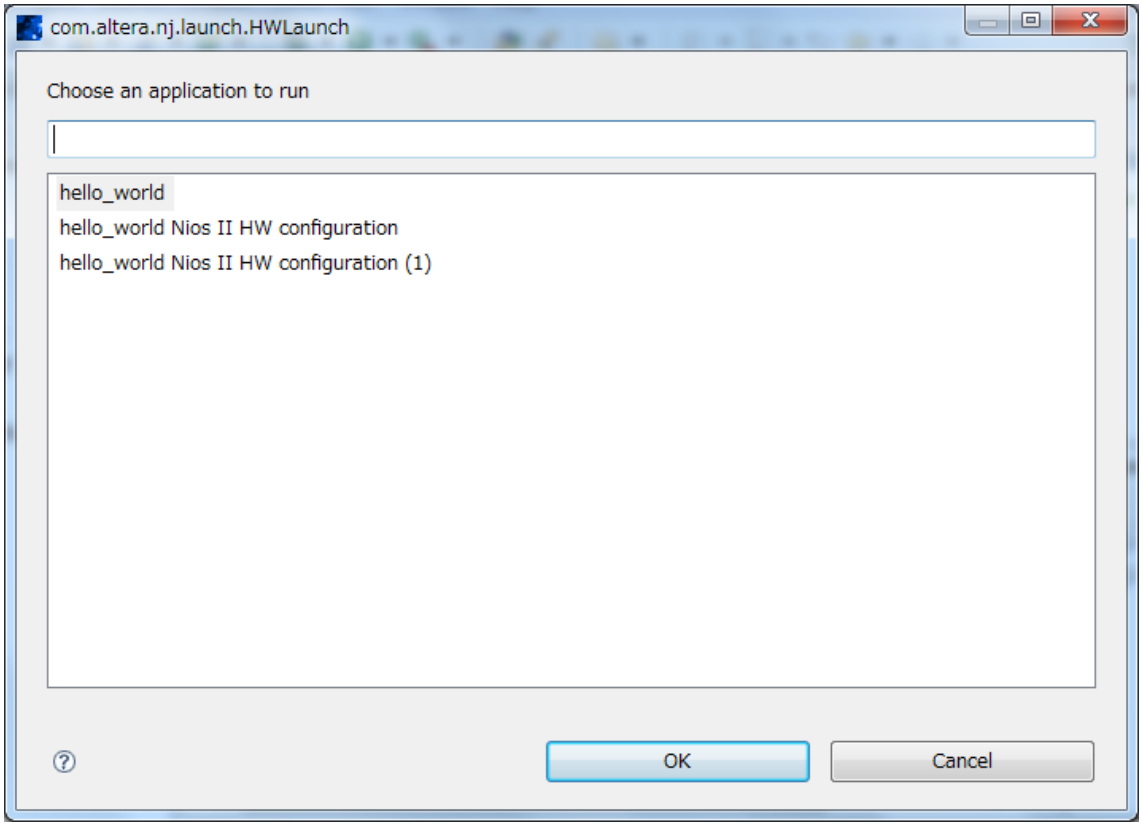
初めて実行の場合、以下の画面が表示されます。赤□の新規画像をクリック



Name：「hello_world」を記入、「Run」ボタンを押す



既存のアプリを実行する場合、下図のように表示されます。（実行したいアプリを選択）



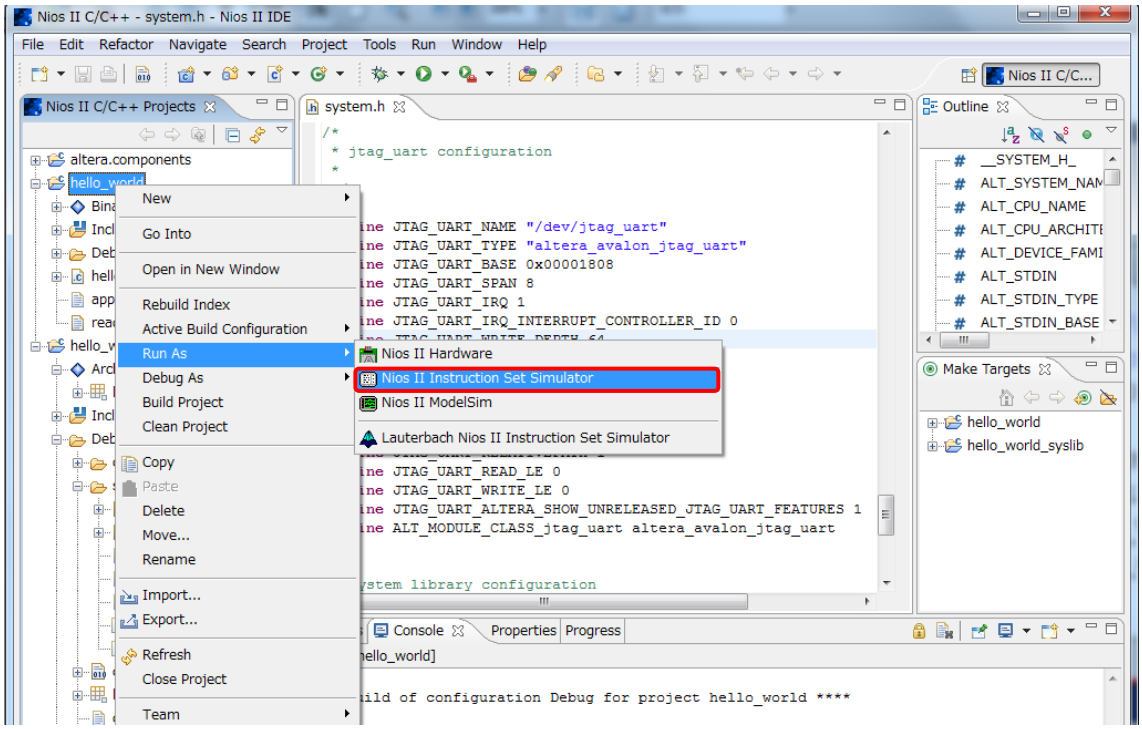
問題がなければ、コンソールから下記のようなメッセージが表示されます。

```
Hello from Nios II!
```

２．ソフトウエアエミュレーション

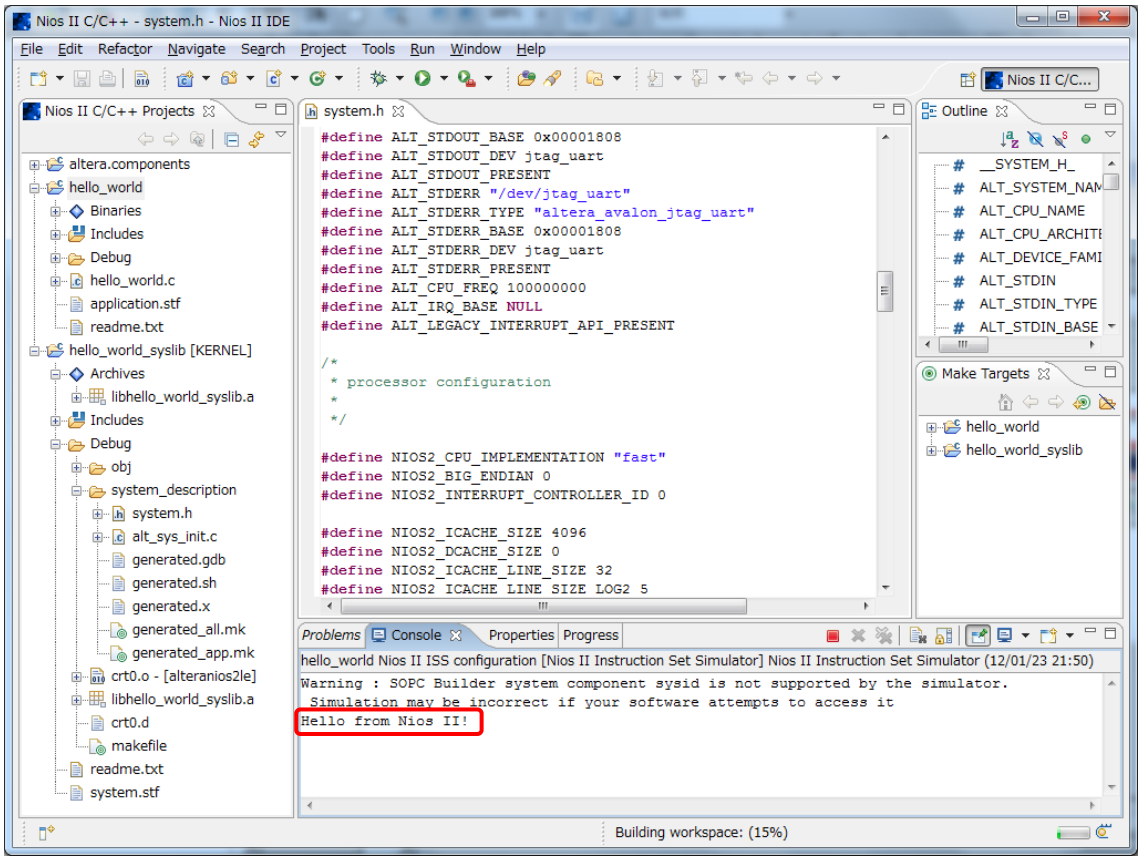
ハードウエアが必要なく、直接 PC で実行できます。

「hello_world」プロジェクト名に右クリック、「Run　As」→「Nios II Instruction Set Simulator」をクリック



ハードウエアオンサインエミュレーションと同じようにコンソールに表示されます。

ハードウエアに関する動作を確認できないため、あまりソフトウェアエミュレーションをお勧めしないです。


Nios II C/C++ - system.h - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++ Projects
altera.components
hello_world
Binaries
Includes
Debug
hello_world.c
application.stf
readme.txt
hello_world_syslib [KERNEL]
Archives
libhello_world_syslib.a
Includes
Debug
obj
system_description
system.h
alt_sys_init.c
generated.gdb
generated.sh
generated.x
generated_all.mk
generated_app.mk
crt0.o - [alteranios2le]
libhello_world_syslib.a
crt0.d
makefile
readme.txt
system.stf
system.h
#define ALT_STDOUT_BASE 0x00001808
#define ALT_STDOUT_DEV jtag_uart
#define ALT_STDOUT_PRESENT
#define ALT_STDERR "/dev/jtag_uart"
#define ALT_STDERR_TYPE "altera_avalon_jtag_uart"
#define ALT_STDERR_BASE 0x00001808
#define ALT_STDERR_DEV jtag_uart
#define ALT_STDERR_PRESENT
#define ALT_CPU_FREQ 100000000
#define ALT_IRQ_BASE NULL
#define ALT_LEGACY_INTERRUPT_API_PRESENT
/*
* processor configuration
*
*/
#define NIOS2_CPU_IMPLEMENTATION "fast"
#define NIOS2_BIG_ENDIAN 0
#define NIOS2_INTERRUPT_CONTROLLER_ID 0
#define NIOS2_ICACHE_SIZE 4096
#define NIOS2_DCACHE_SIZE 0
#define NIOS2_ICACHE_LINE_SIZE 32
#define NIOS2_ICACHE_LINE_SIZE_LOG2 5
Outline
__SYSTEM_H_
ALT_SYSTEM_NAM
ALT_CPU_NAME
ALT_CPU_ARCHITE
ALT_DEVICE_FAMI
ALT_STDIN
ALT_STDIN_TYPE
ALT_STDIN_BASE
Make Targets
hello_world
hello_world_syslib
Problems Console Properties Progress
hello_world Nios II ISS configuration [Nios II Instruction Set Simulator] Nios II Instruction Set Simulator (12/01/23 21:50)
Warning : SOPC Builder system component sysid is not supported by the simulator.
Simulation may be incorrect if your software attempts to access it
Hello from Nios II!
Building workspace: (15%)

第五章 プログラムダウンロード

参考資料：

■アルテラ社の「Nios II フラッシュ・プログラマユーザガイド」

http://www.altera.co.jp/literature/ug/ug_nios2_flash_programmer_j.pdf

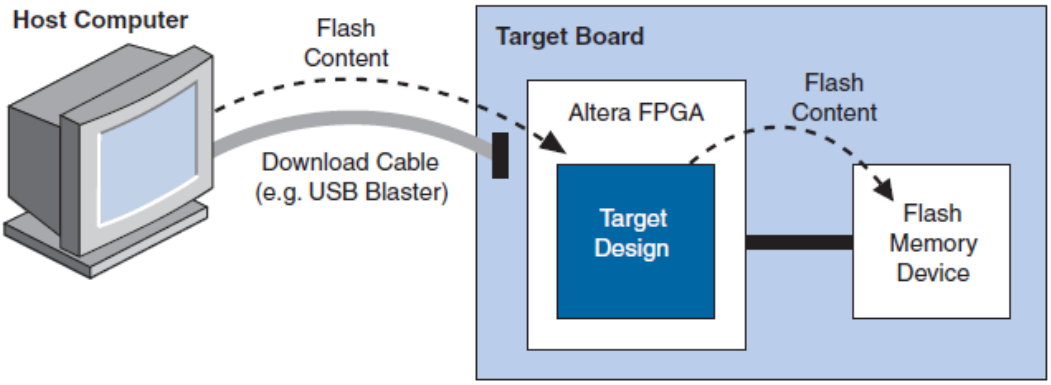
5.1 概要

本章からコンパイルできたプログラムをどのように CycloneIV ボードのコンフィギュレーションチップ（EPCS）にダウンロードするかを説明します。

CycloneIV ボードに 64MBit のコンフィギュレーションチップ（EPCS）があります、このチップは設定情報と NIOS II プログラムを保存するためです。一点を説明しますが、なぜ EPCS にプログラムをダウンロードしパラレル Flash にダウンロードしないですか？理由はパラレル Flash を無くしてパラレル Flash の 32 ピンのスペースを空けます、PCB 実装時にもスペースを節約できます。パラレル Flash のスピードがより早いと思われる人も多いですが、実は、たとえパラレル Flash に保存しても電源を入れた時あるいはリセット時だけ一回を読み込みます、スピード上に影響はあまりないです。(特別のニーズがある場合、或いは頻繁に Flash を操作する場合のみにはパラレル Flash 必要です。)

プログラムダウンロード流れ：

Nios II フラッシュ・プログラマの動作



5.2 コンフィグレーションファイルダウンロード

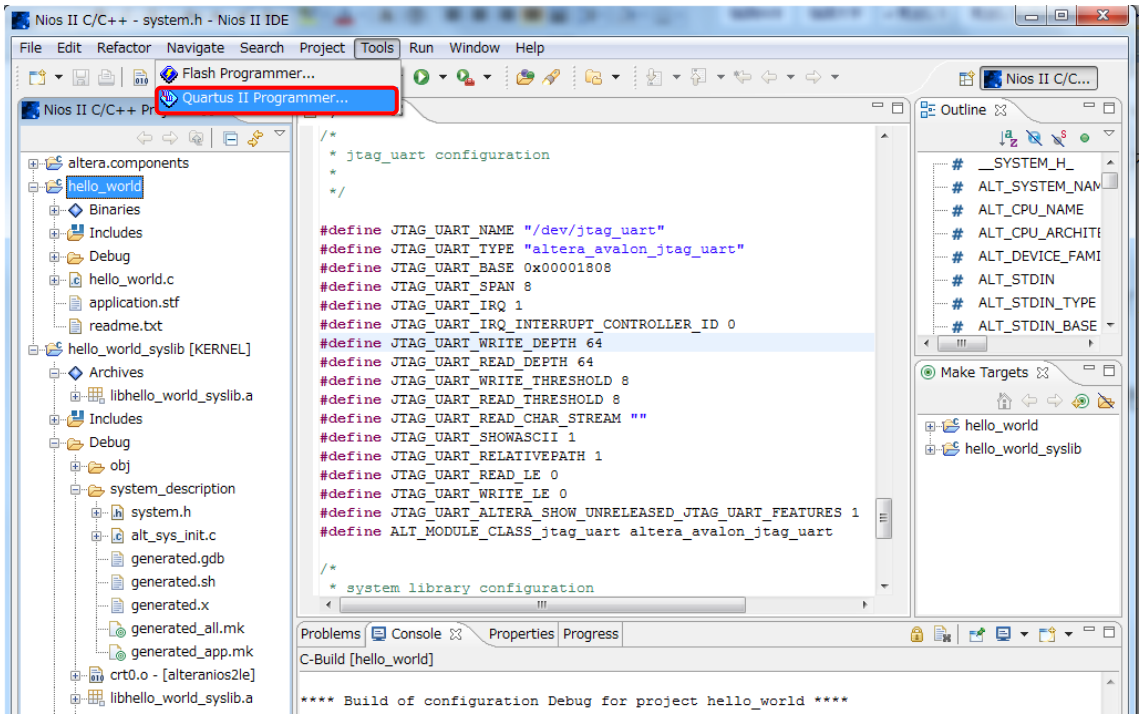
プログラムを EPCS にダウンロードする前、コンフィグレーションファイルをダウンロード必要です。

まず、USB ダウンロードケーブルは開発ボードの JTAG インタフェースと接続にし、次はボードの電源を入れます。最後、NIOS II IDE を立ち上げましょう。

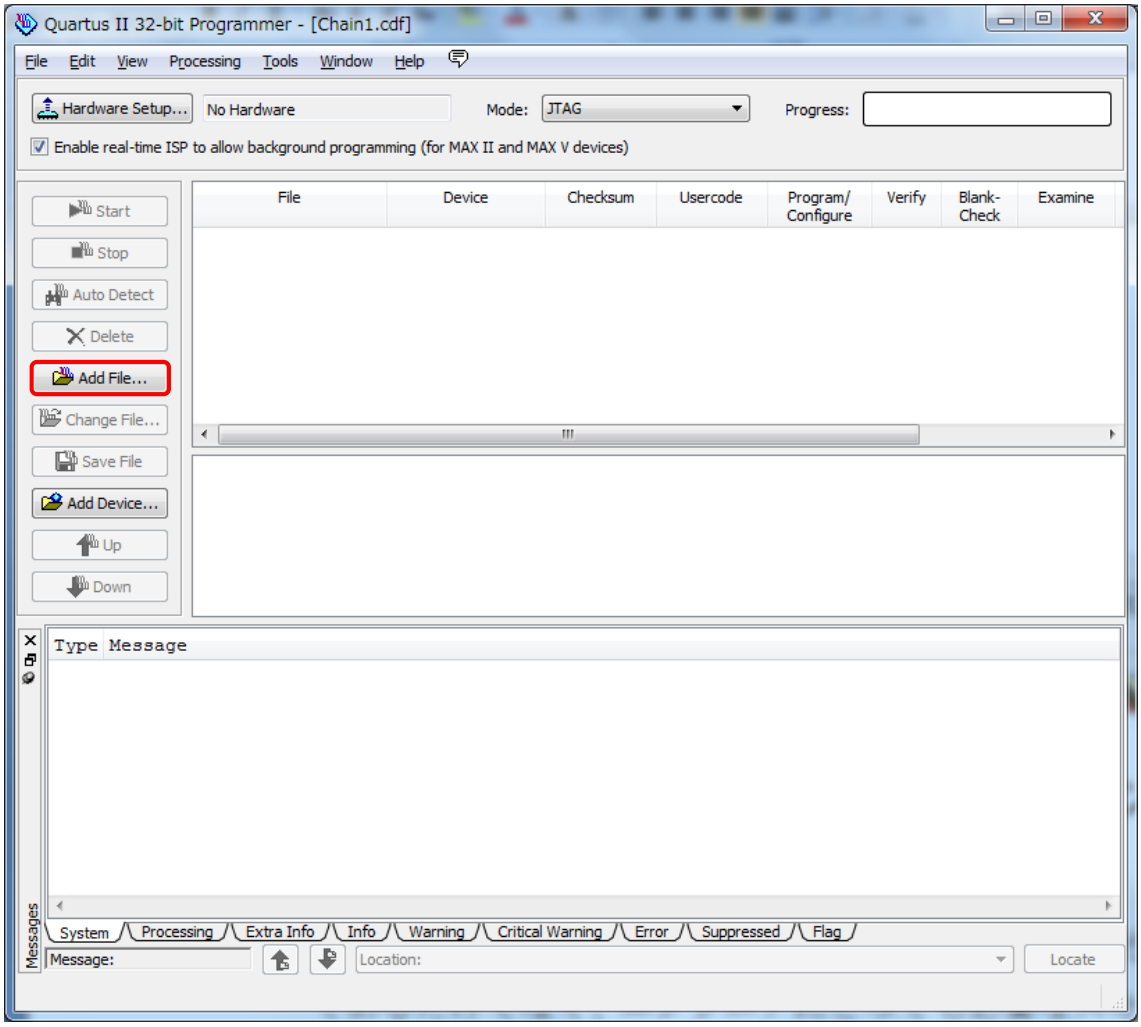
メニュー「Tools」→「Quartus II Programmer...」をクリック

※第四章のように「Quartus II」のメニュー「Tools」→「Programmer」をクリックしても

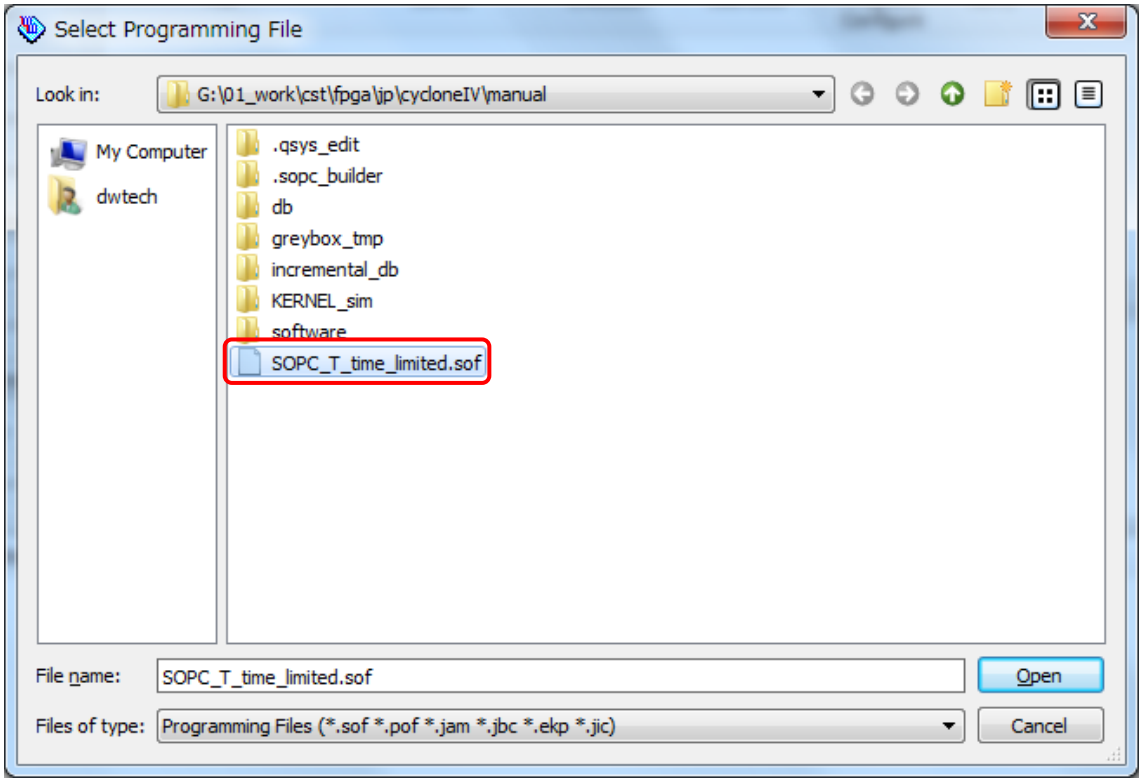
同じです。



ポップアップされたダイアローグに「Add Files…」ボタンを押下

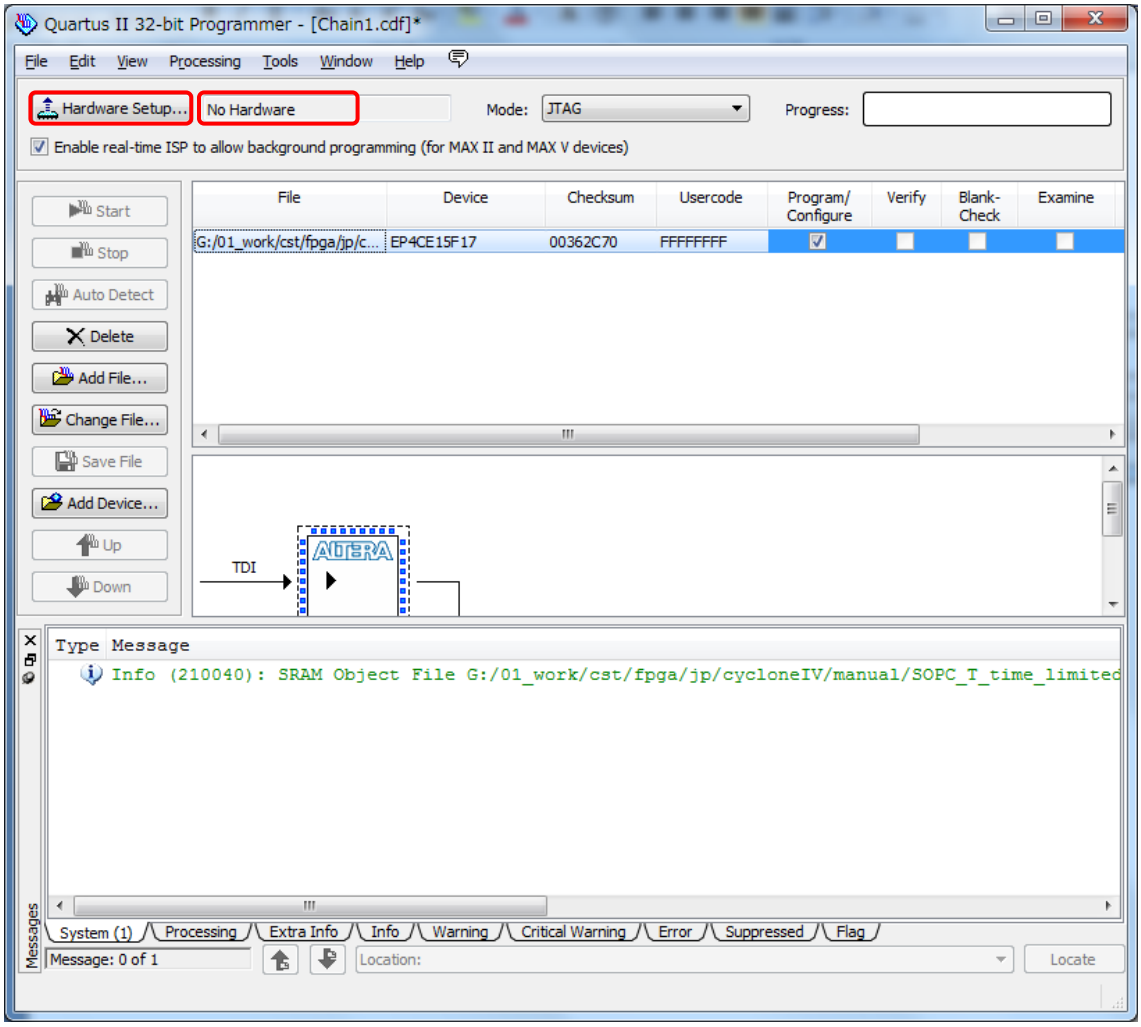

Quartus II 32-bit Programmer - [Chain1.cdf]
File Edit View Processing Tools Window Help
Hardware Setup...
No Hardware
Mode: JTAG
Progress:
Enable real-time ISP to allow background programming (for MAX II and MAX V devices)
Start
Stop
Auto Detect
Delete
Add File...
Change File...
Save File
Add Device...
Up
Down
File
Device
Checksum
Usercode
Program/ Configure
Verify
Blank-Check
Examine
Type Message
System Processing Extra Info Info Warning Critical Warning Error Suppressed Flag
Message:
Location:
Locate
Messages

sof ファイルを選べ

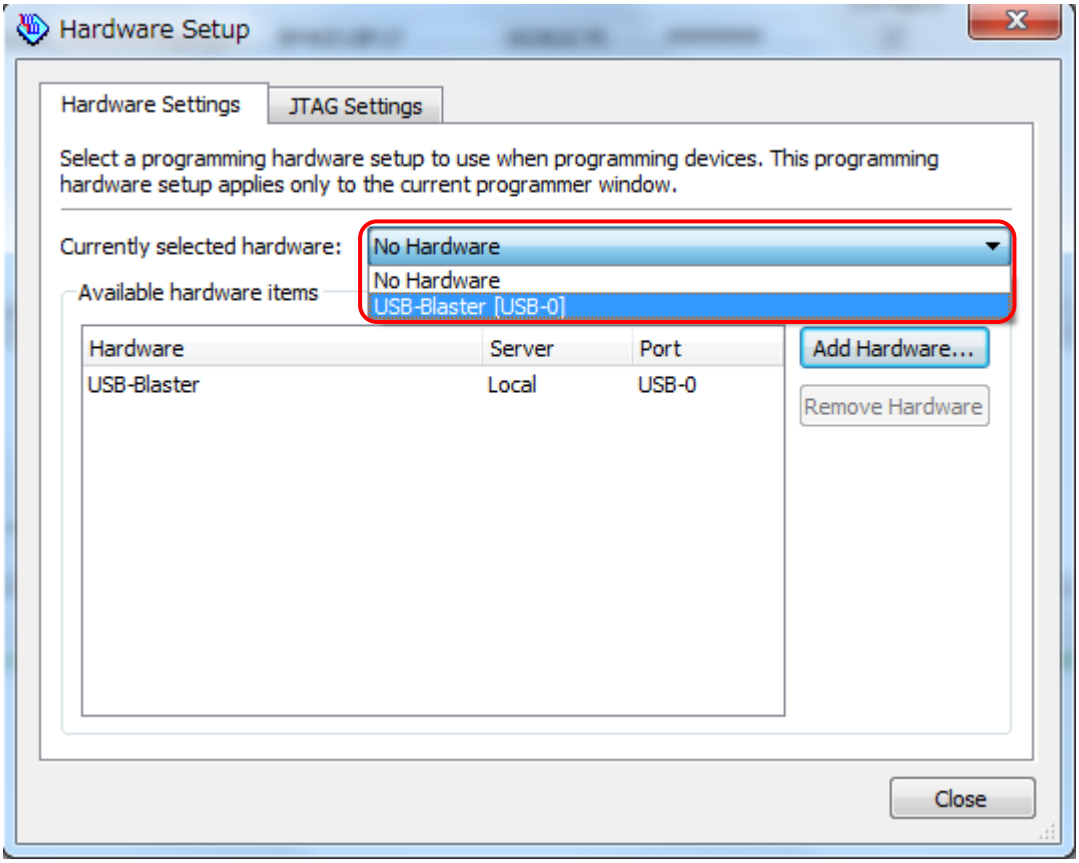


Programmer 画面に戻します、下記画面のように「No Hardware」が表示される場合、まず、ハードウエア（USB ダウンロードケーブル）を設定しましょう。

「Hardware　Setup」をクリック



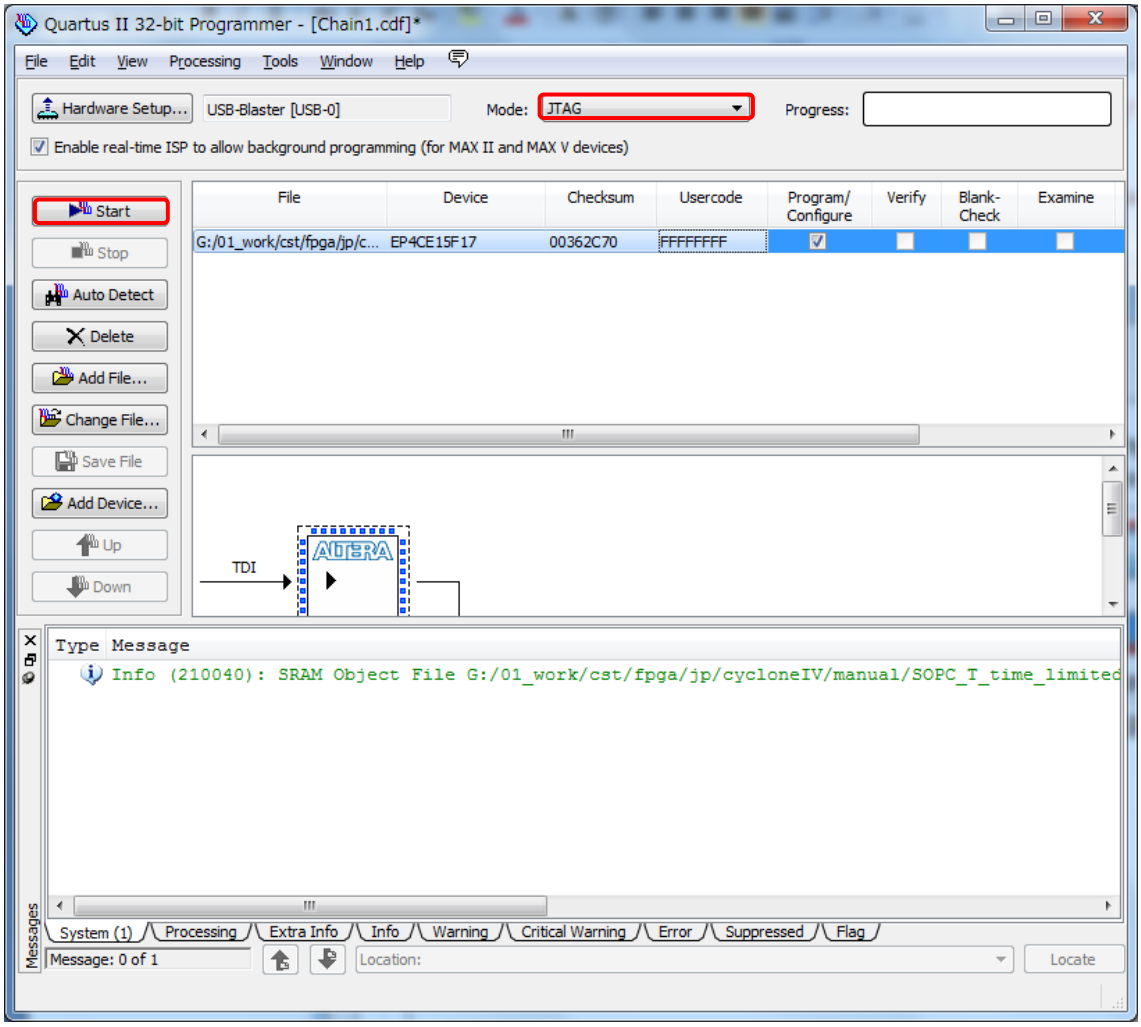
「USB-Blaster」を選択



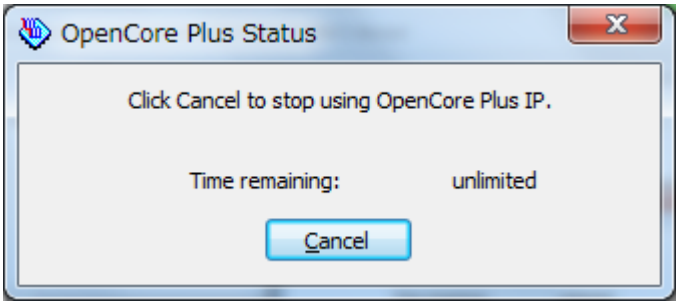
「Start」ボタンを押せる状態になります、「Start」ボタンを押しダウンロードを始めます。

注意：JTAG モードでダウンロード場合、デバッグできます、リセットあるいは電源切断の後、プログラムがなくなった、再度デバッグの時、もう一度ダウンロード必要です。

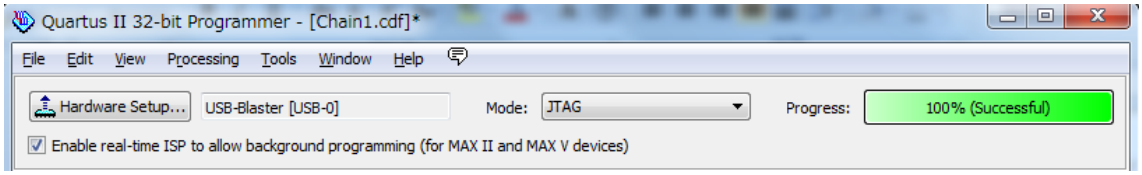
リリースの時、FPGA に ACS モードでダウンロードしてください。JTAG あるいは ACS インタフェースに差し込み、外しの時、必ず電源を切れてから実施してください。電源を入れたまま実施すると、ハードウエアのインタフェースが壊す恐れがあります、ご注意ください。



書込みが完了したら、下記画面が表示されます、「Cancel」をクリックしないでください。
※Quartus II が正式版の場合、下記画面が出ません。



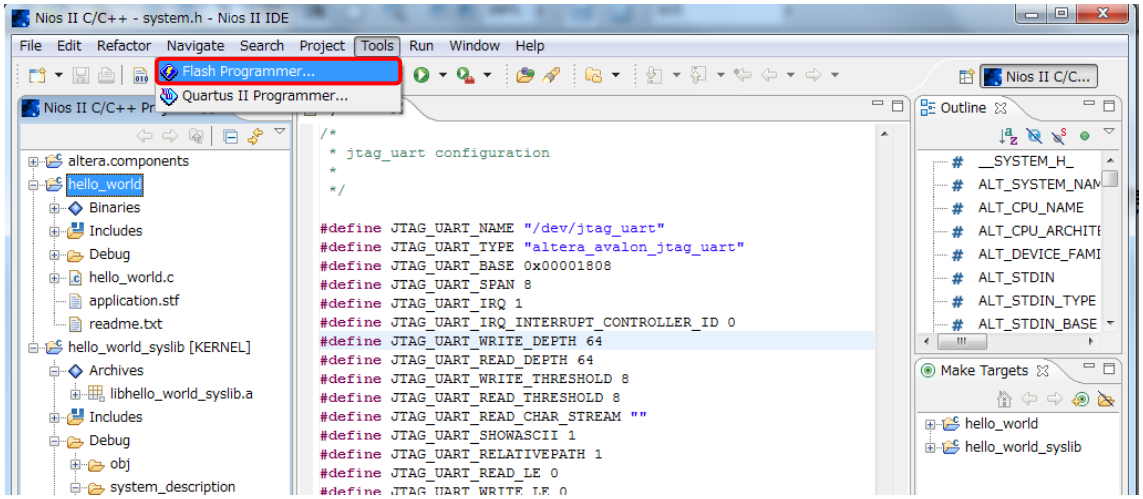
成功すれば、下記画面が出ます。



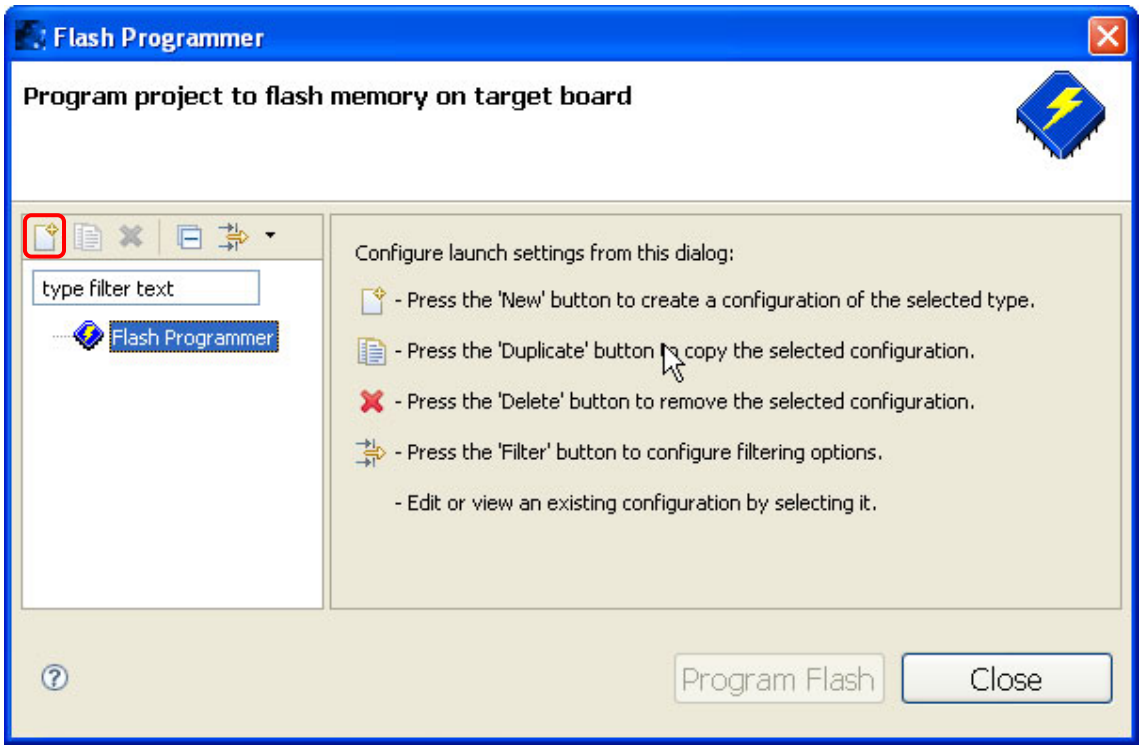
5.3 プログラムダウンロード

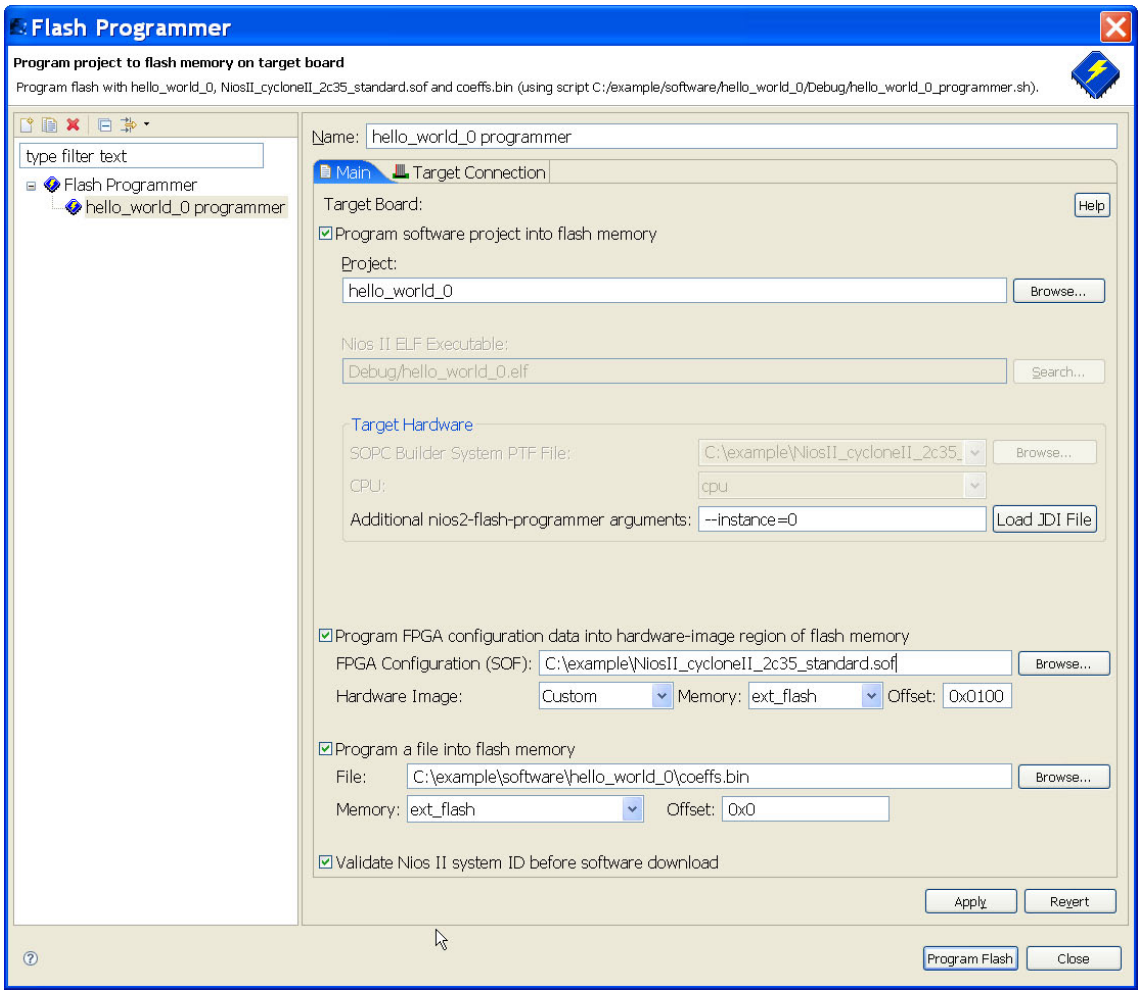
これからプログラムを EPCS にダウンロードしましょう。(デバッグが完了したら、製品に組み込み時にも EPCS をダウンロード必要です。)

メニュー「Tools」→「Flash Programmer…」をクリック



Flash Programmer ダイアログ・ボックスが表示されます。フラッシュを初めてプログラムする場合、ダイアログ・ボックスは書下図のように表示されます。既存のフラッシュ・コンフィギュレーションが存在する場合は、次ページの図のように表示されます。





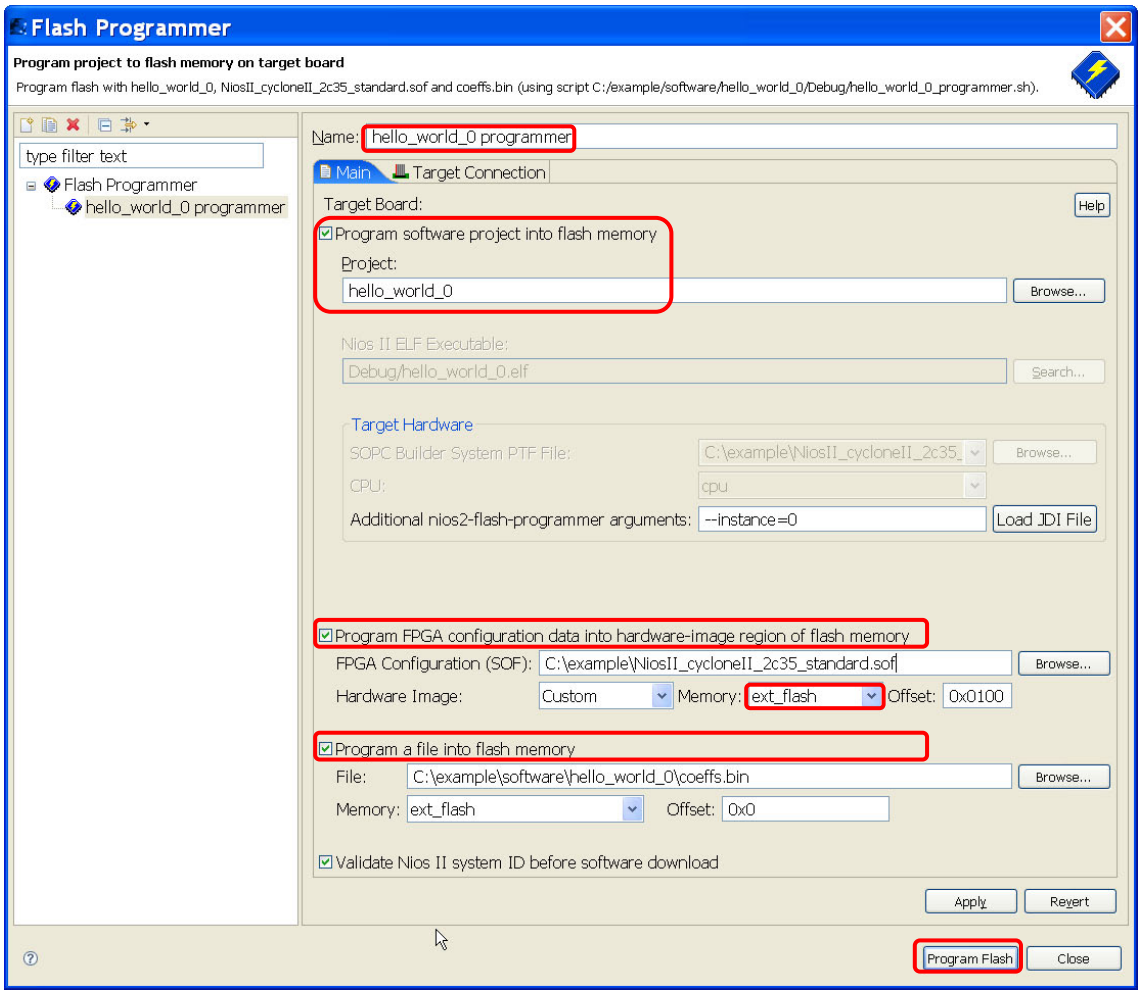
新規の場合、「New」画像ボタンを押して内容は次のように修正

Memory：「EPCS」に変更

Program　FPGA　Configuration data into hardware-image region of flash memory:チェック

Program　a file into Flash memory：チェック

変更完了後、「Programm Flash」をクリックしてからプログラムダウンロードを始めます。


Flash Programmer
Program project to flash memory on target board
Program flash with hello_world_0, NiosII_cycloneII_2c35_standard.sof and coeffs.bin (using script C:/example/software/hello_world_0/Debug/hello_world_0_programmer.sh).
type filter text
Flash Programmer
hello_world_0 programmer
Name: hello_world_0 programmer
Main
Target Connection
Target Board:
Help
Program software project into flash memory
Project:
hello_world_0
Browse...
Nios II ELF Executable:
Debug/hello_world_0.elf
Search...
Target Hardware
SOPC Builder System PTF File:
C:\example\NiosII_cycloneII_2c35_
Browse...
CPU:
cpu
Additional nios2-flash-programmer arguments:
--instance=0
Load JDI File
Program FPGA configuration data into hardware-image region of flash memory
FPGA Configuration (SOF):
C:\example\NiosII_cycloneII_2c35_standard.sof
Browse...
Hardware Image:
Custom
Memory:
ext_flash
Offset:
0x0100
Program a file into flash memory
File:
C:\example\software\hello_world_0\coeffs.bin
Browse...
Memory:
ext_flash
Offset:
0x0
Validate Nios II system ID before software download
Apply
Revert
Program Flash
Close

第六章 プログラミングルール

6.1 プログラミングの参照標準

良いコーディングスタイルとプログラミングの仕様は、良いコードを書くための基礎であるだけでなく、エンジニアにとっても必要なスキルです。本章説明の仕様は、C 言語の創始者：BWKernighan と DMRitchit（K&R とも呼ばれる）書籍"プログラミング言語 C"の例に従い、Linux カーネルのコーディングスタイルへも参照します。この仕様は、C 言語知識がある程度持つユーザに適用されます、入門からのユーザはまず"プログラミング言語 C"から精通するのはお勧めします。それ以外、この仕様はマイクロコントローラ、ARM および他の組み込みプロセッサのファームウェアの開発に適用されます。

6.2 ルールのフォーマット

１．字下げ

函数、if、for、while、switch case、do while 等を書く際、字下げ必要です。普通のエディタあるいは IDE を使用するにもかかわらず、字下げはスペースではなく、“Tab”キーを使われます。一般的に、字下げの長さは 8 文字です。

例：

```
int main(void)
{
        int i;
                for (i = 0; i < 100; i++) {
                        printf(“Hello world!¥r¥n”);
                }
        return 0;
}
```

２．スペース及び空白行

スペース及び空白行はプログラムの可読性を高める事として使われますが、あまり多く（2 個以上）のスペースと空白行を挿入しない方がよいです。函数の最初のローカル変数宣言の後、空白行を一つ追加必要です。函数内部のロジック部分も空白行を追加した方がよいです。ファイルの最後も空白行を追加してください。

コードにスペースを追加した目的はプログラムを分かりやすくするためです、C 言語のキーワードの後もスペースを追加必要です。例：

```
if (a < 100 && a > 50)
        a++;
```

３．ブレース

函数のブレースは函数の第二行目と最後にあります、if、while、switch、do のブレースはキーワードの行とロジック部分の最後にあります、
最後のブレースはキーワードで整列します。

```
while (i++ < 100) {
        ……
        ……
}
```

6.3 要素とネーミングルール

１．ファイル

C 言語のソースファイルは.c ファイルと.h ファイルを含みます、ファイルのネーミングはその意味を反映するための名前にするべきです。例：

FAT32 プロトコルのドライバー：fat32.c を名前に付けられます。a.c、newfiel.c、my.c、wang.c 等の意味不明のファイル名を付けない方がよいです。ファイル名は英数字、アンダーラインを組み合わせてネーミングしましょう、スペース、ascii 以外の文字等を避けた方がよいです。

リソース宣言のため、.c ファイルごとに.h ファイルがあります。.h ファイルの中にマイクロ定義、タイプ定義、グローバル変数、グローバル函数を宣言できます。

.c ファイルはローカル変数、函数プロタイプ、函数本体を含みます、重複使うため、.h ファイルの最初に一般定義コントロールを追加します

例：

```
#ifndef __FAT32_H__
#define __FAT32_H__

……
……

#endif /* __FAT32_H__ */
```

２．マイクロ、列挙体

マイクロ、列挙体は両方も大文字（英数字）及びアンダーラインを組み合わせてネーミングにします、マイクロと定数の間を”Tab”で分離し、同種類のマイクロ定義は一緒に記述されてインクルードファイル(.h)に入れられます。

マイクロは意味を明確するためネーミングにし、専門用語以外、単語でマイクロの定義を名前に付けられます。

異なるマイクロは空白行で分割し、できるだけコメントも追加してください。

例：

```
#define  WR0      0x00
```

```
#define  WR1     0x01
#define  WR2     0x02

#define  SYSTEM_CLK   40000000          //40MHz

#define  MAX_FREQUENCY      20000000 //20MHz
```

３．カスタムタイプ

我々は C 言語のキーワード"typedef"でデータタイプを自分で定義します、データタイプの定義は構造体、共用体及び函数定義を含みます。

ANSI C に既に含まれるデータタイプ（例：unsigned char、unsigned short int、double 等)、の再定義のことが望ましくないです。

構造体、共用体の定義は大文字（英語文字)、後ろに「_T」を付けるのをお勧めします。

例：

```
typedef struct{ char receive_buffer[BUFFER_SIZE];
            unsigned long int baudrate;
            int (* initialize)(int /* baudrate */);
            int (* send_string)(char * /* string */);
            int (* printf)(unsigned char *,...);
}UART_T;
```

４．関数宣言と実体

SVO 構造を使用して関数名がつけられます、アンダーラインで分割され、小文字、数字及びアンダーラインの組み合わせでネーミングにします。

関数のプロトタイプや関数本体にもかかわらず、全ての函数タイプ及びパラメータタイプを含みます。（void 型でも省略できません）

ファイルの内部関数はプロトタイプ前に「static」キーワードを追加必要です、関数のプロトタイプを宣言時、パラメータの説明等のコメントを追加。

例：

```
int write_data(int /* channel */, int /* data */);
```

多くの函数は実行結果を戻す必要です、0:正常、-1：エラーというように定義されます、-1～-999 はカスタマイズで定義のエラーコード、このエラーコードはマイクロあるいは変数で実現できます。

```
int write_data(int ch,int dat)
{
            if (ch > LTC2600_MAX_DATA)
                        return ERROR_LTC2600_DATA_OVERFLOW;
```

```
        if (dat > LTC2600_DACH)
                return ERROR_LTC2600_CHAN_OVERFLOW;

        ……

        return 0;
}
```

ロジック機能から見ると、簡単な、短い関数が一番良いです。函数で実現のロジック内容は函数名前と一致する必要です、関数名前の意味を超えた場合、関数を理解しにくくなります。

一般に、システムライブラリを呼び出して機能を実現します。（全て自分の函数で実現するわけではない）

５．変数と初期化

変数は小文字、数字及びアンダーラインでネーミングにします、グローバル変数はある程度意味を表すべき、フル単語が方がよいです。

一般にローカル変数が視野範囲にありますので、簡単或いはひと文字でも Ok、例えば、i,a 等。

グローバル変数が出来れば少ない方がよいです、変数の属性により分けられ、構造体の形でメインです。

プロジェクトのグローバル変数はインクロードファイル（.h ファイル）にケーワード「extern」で宣言します、

ファイルに所属グローバル変数は変数前にキーワード「static」を付けられます。

変数を使う前初期化必要です、静的な初期化あるいは函数実行時に初期化を行っても良いです。構造体の初期化は c99 仕様の指定初期化をお勧めします。例：

```
UART_T uart0 = {
        .baudrate = 9600,
        .initialize = uart0_init
};
```

配列の次元は、マクロ定義を使用するのが最適です、配列を使う前に必ず初期化必要です。（値付き或いは 0 にセット）

配列の次元を判断する際、sizeof キーワードを使います、直接数字で次元を書いたのはお控えください。

ポインタは変数の一つですので、使う前にも初期化必要です。CPU 周辺デバイスを直接マッピングする時あるいは割り込み内の変数として使う時、最適化を抑止するため、変数前に「volatile」を追加しましょう。

６．コメント

コメントは勿論無くてはいけないですが、多すぎコメントも必要ではありません。プログラムを分かりやすく為適当のコメントを追加しましょう。関数本体内のコメントは”//”をお勧めします。
プログラムファイルの最初、当ファイルの説明用のコメントも追加した方がよいです。

```
/*
 * File：ltc2600.c
 * Description：This file is ltc2600(8-channel 16bit DAC) driver.
 * Author：Csun.
 * Copyright：Csun System Technology Co.,Ltd.
 *
 * History
 * ------------------
 * Rev：0.00
 * Date：01/15/2012
 *
 * create.
 * ------------------
 * Rev：0.01
 * Date：01/15/2012
 *
 * Fix bugs.
 * ------------------
 *
*/
```

関数本体前、説明用のコメントも追加します。例：

```
/*
 * Name：write_data
 * Description：Write data to ltc2600.
 * Author：Csun.
 *
 * History
 * ------------------
 * Rev：0.00
 * Date：01/15/2012
 *
 * create.
```

```
 * ----------------
*/
```

6.4 プロジェクト管理

１．プロジェクトフォルダー

C 言語のプロジェクトごとに、機能によりソースファイルのフォルダーを分けられます。

フォルダーの名前にはスペース、ASCII 以外の文字も使わないでください。

例：

```
doc 説明ドキュメント
config コンフィグフォルダー（パラメータ設定、リンク等の設定ファイル）
driver ハードウエアドライバーフォルダー（チップ、CPU 周辺ドライバー等）
font フォントドライバ
Gui GUI 画面ドライバー
main 主プログラム
include ヘッダファイル
obj コンパイル後のターゲットファイル
```

２．機能分類

機能分類には明確のロジック、階層関係で行われます、分類できたら、所属機能を超え呼び出しはしないでください、システムリソースをお使い切れないためお互いに呼び出す事もしないください。

例えば、driver 内のソースはハードウエアを直接アクセスできますが、それ以外のソースは直接ハードウエアリソースを呼び出す事がないはずです。

３．ファイル管理

ソースファイルを一旦作成したら、ファイルの最初にファイルの役割、バージョン、履歴等を追加しましょう。

毎度修正後、修正履歴を記録します、プロジェクト完了時あるいは段階のリリース時、全てプロジェクトフォルダー、ファイルを読み取り専用に設定、あるいは状況記録（カレント状況、バッグ等)。ソース管理は SVN 等の汎用の無料バージョン管理ツールで実施するのをお勧めします。

※毎日バックアップ実施等対策を取ってください。

6.5 アドバイス

１．コードエディタ

沢山の良いコードエディタがあります、例：Vim、Emacs、Souce-Insight、Edit-Plus、EmEditor、秀丸等

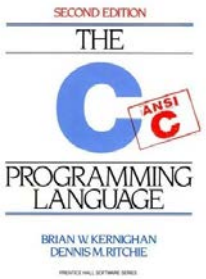
２．PC 側のコンパイラ及び IDE

GCC (GNU Compiler Collection)は非常に良いコンパイラとなります、基本的に全て OS でサポートします、たくさんの CPU でのクロスコンパイラもあります、例えば、SDCC、

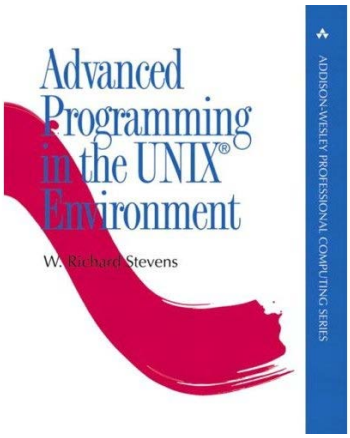
arm-elf-gcc 等。MS-Windows で Mingw32 移植版を、IDE は Dev-cpp、Code::Blocks、Eclipse IDE for C/C++をお勧めします。

３．参考用のリソース及び URL

書籍名：《The C Programming Language》



書籍名：《Advanced Programming in the UNIX Environment》



http://www.gnu.org GNU Operating System

http://www.sf.net Source Forge

http://www.kernel.net The Linux Kernel Archives

第七章 実例開発

ここまで Nios II 開発(ハードウエア、ソフトウエア、ダウンロード・書込み、プログラムルール)を説明しました、これから実例を説明していきましょう。

本マニュアルの実例のソース（ハードウエア、ソフトウエア両方も含まれる）は下記 URL からダウンロードできます。（すべて実例は作成できたら、最後はボード出荷時のプログラムになります。）

http://www.dragonwake.com/download/FPGA/EP4CE15/Example.zip

※ダウンロード後、解凍して Readme.txt ファイルを参照のうえ使ってください。

7.1 LED

この実例でレジスタ方式でハードウエアをどのようにアクセスかを説明します。

7.1.1 概要

本節は一番目のハードウエアに関するサンプルを説明します、簡単ですが、代表的なアプリとも言えます。

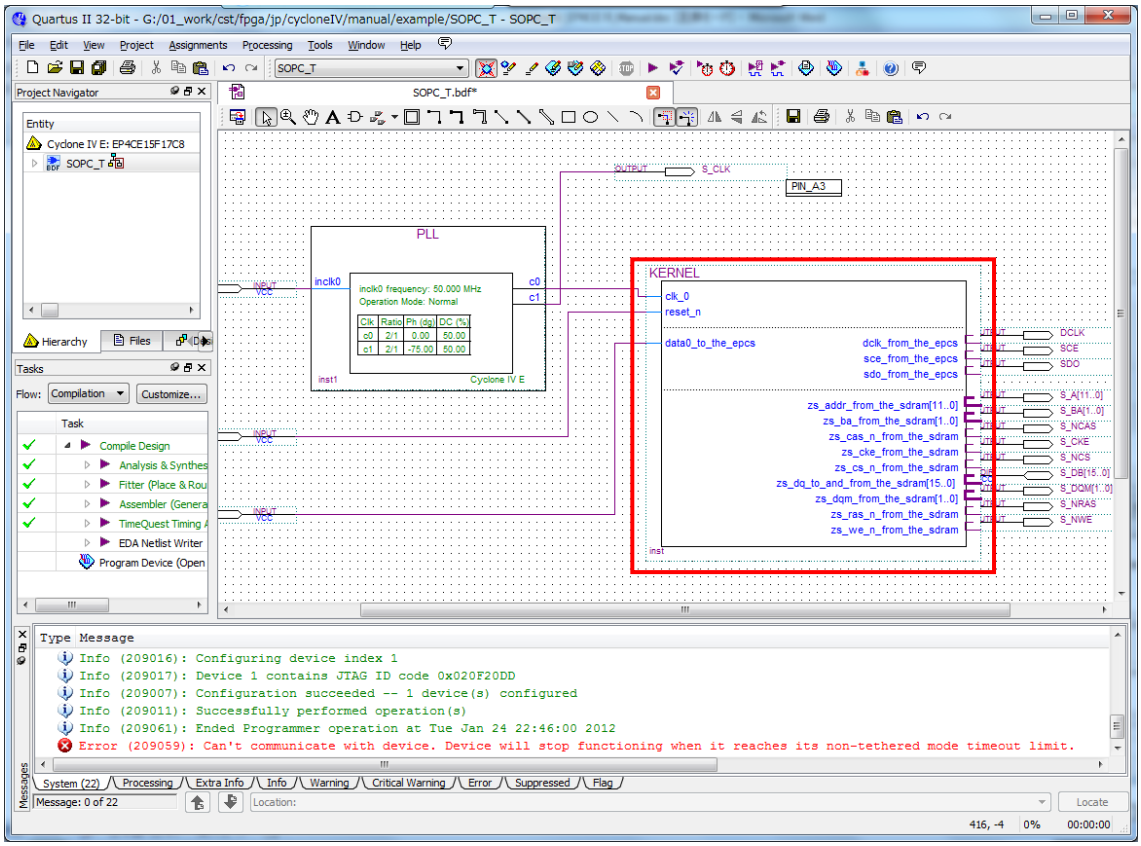
Nios II 開発の本質を見て、レジスタ操作方法で Nios II 開発が SCM と似ています、Nios II IDE が提供する API の使用を控えます。この方法の利点は、まず、SCM 開発経験者がこの操作方法をよく知ります、次、Nios II の API ではなく、自分で開発します、Nios II 開発本質を理解できます。

SCM 開発経験者は LED 試験をよく覚えていると思いますが、これはハードウエアの入門試験となります。この試験により、Nios II 開発世界を見て Nios の魅力を感じましょう。

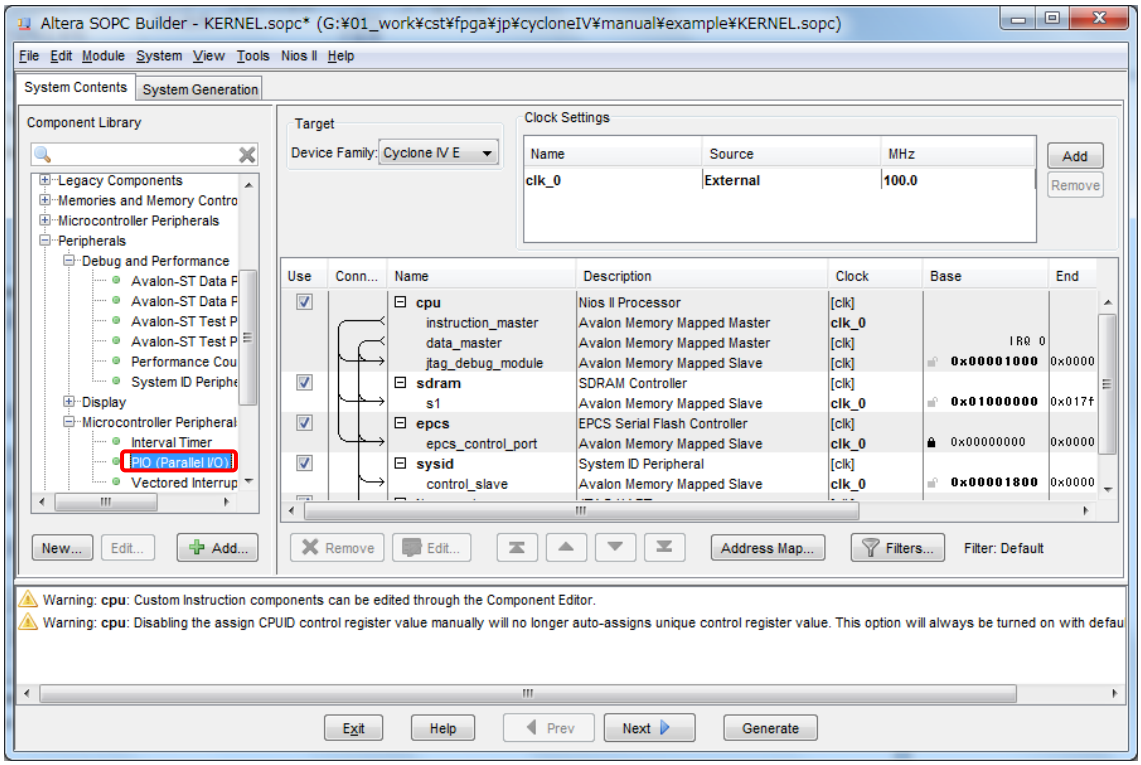
7.1.2 ハードウエア開発

１．PIO モジュールを IP コアに追加

「ハードウエア開発」という章に作成した Quartus プロジェクトを開き、KERNEL をダブルクリックして SOPC Builder を起動させましょう。



SOPC Builder が起動後、左側「Peripherals」→「MicroController Peripherals」→「PIO」をダブルクリック

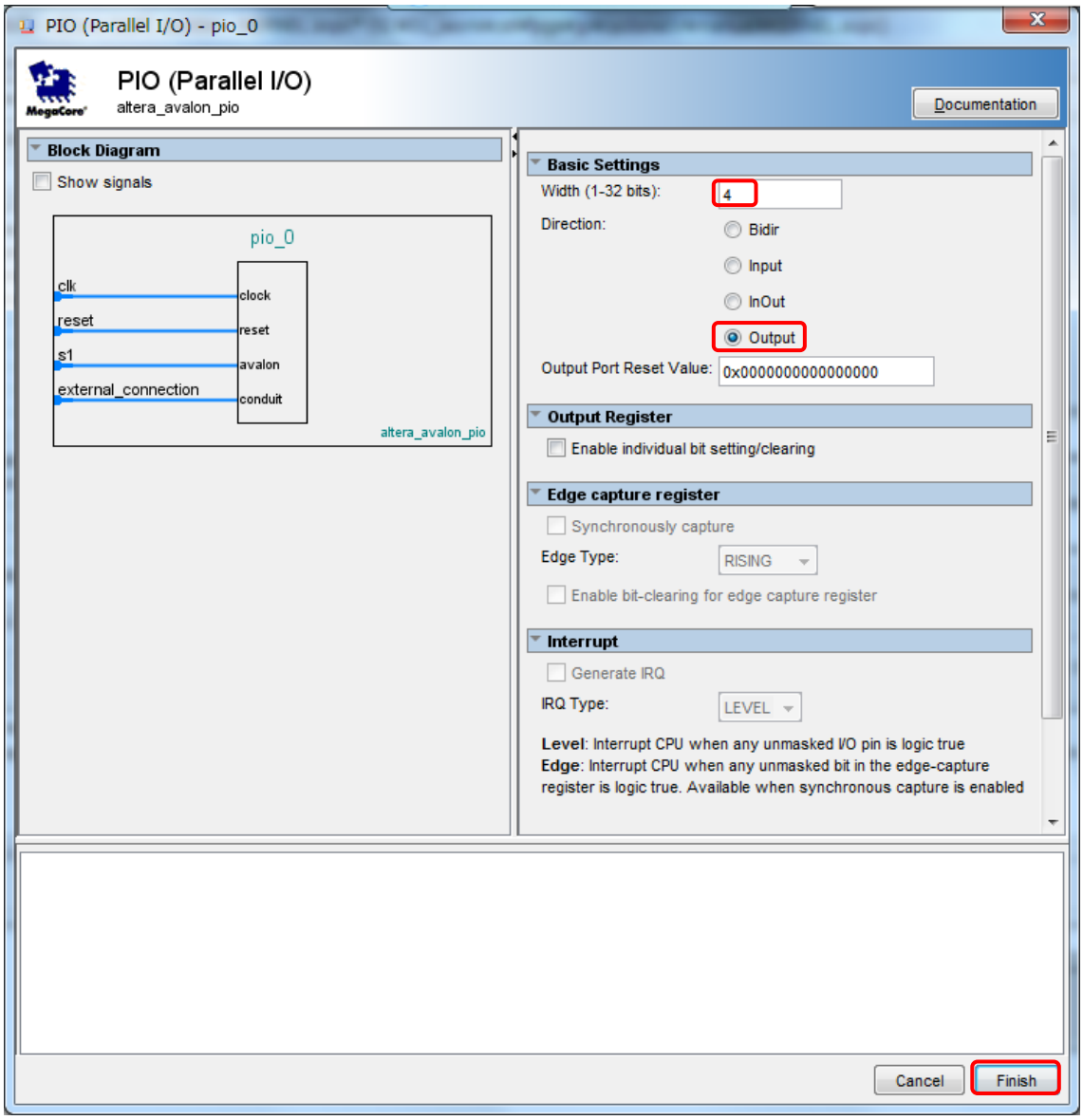


下図のように設定してください。

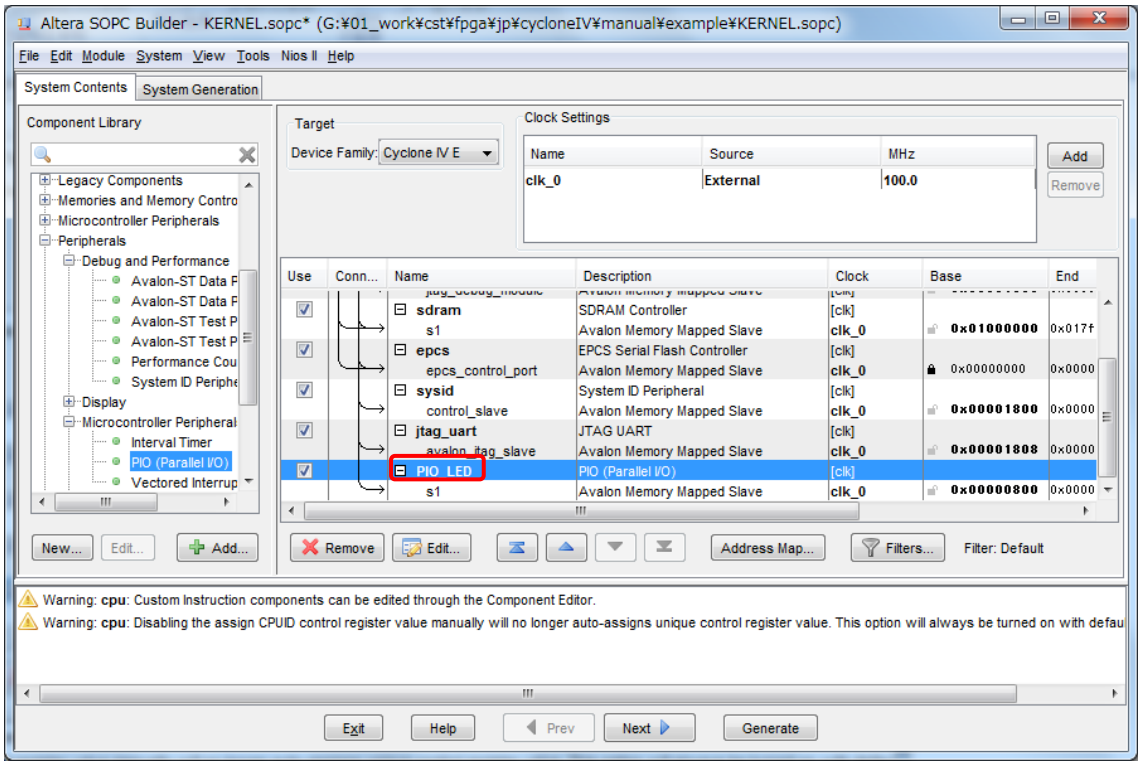
Width：4 を記入（4 個 LED を使う）

Direction：Output をチェック

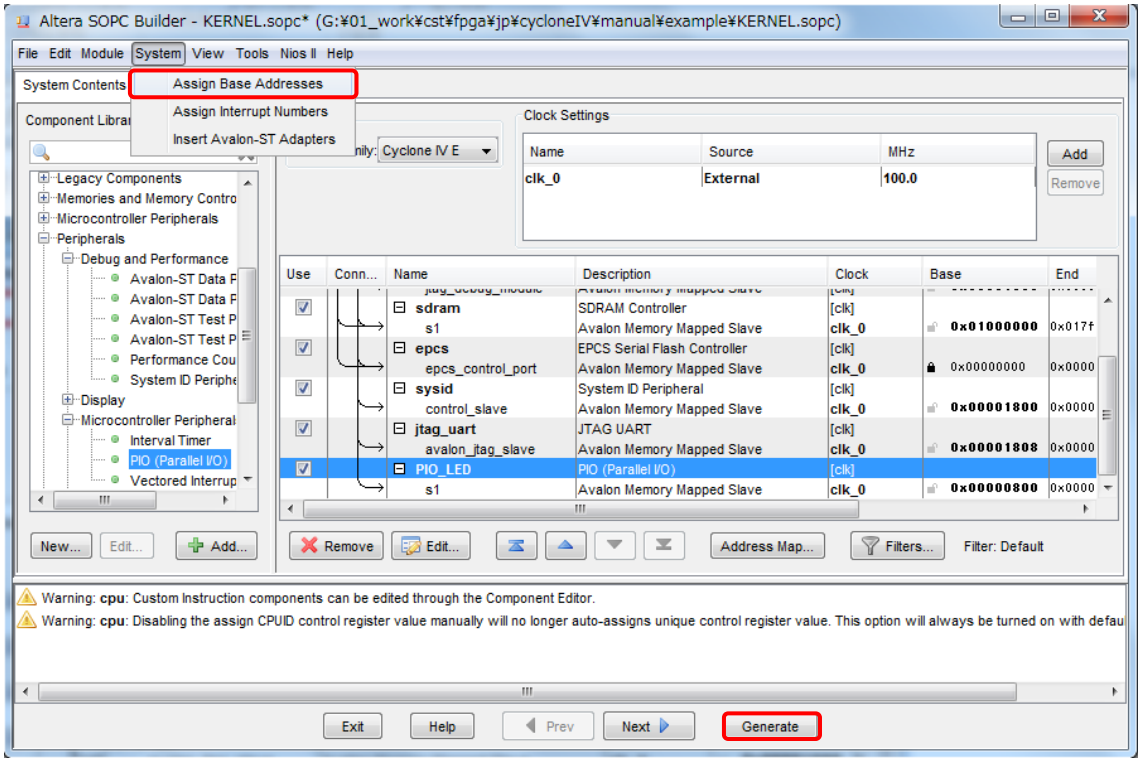
設定が完了後、「Finish」をクリック



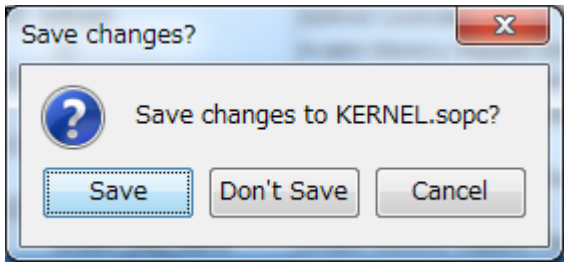
名前を「PIO_LED」に修正



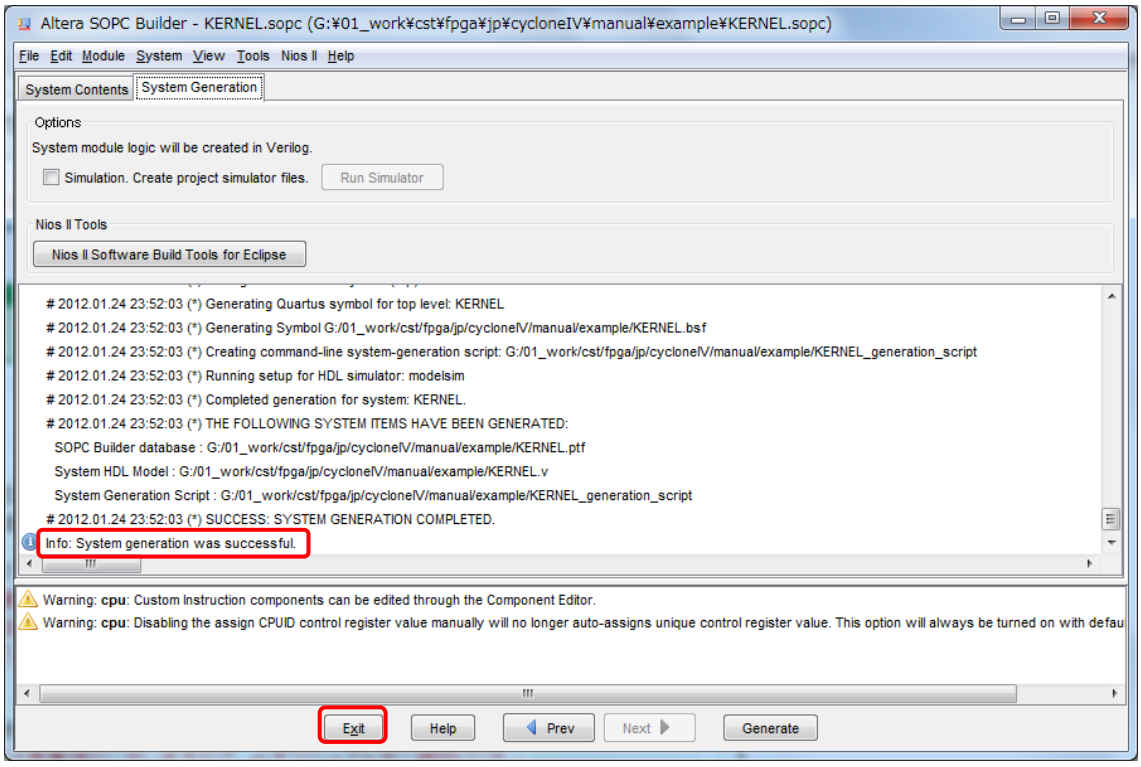
前述のようにアドレスを自動に割り当てにしましょう。



ソフトマクロを生成：上図の「Generate」ボタンをクリックし保存画面が出ます、「Save」ボタンも押し保存してから生成しましょう。

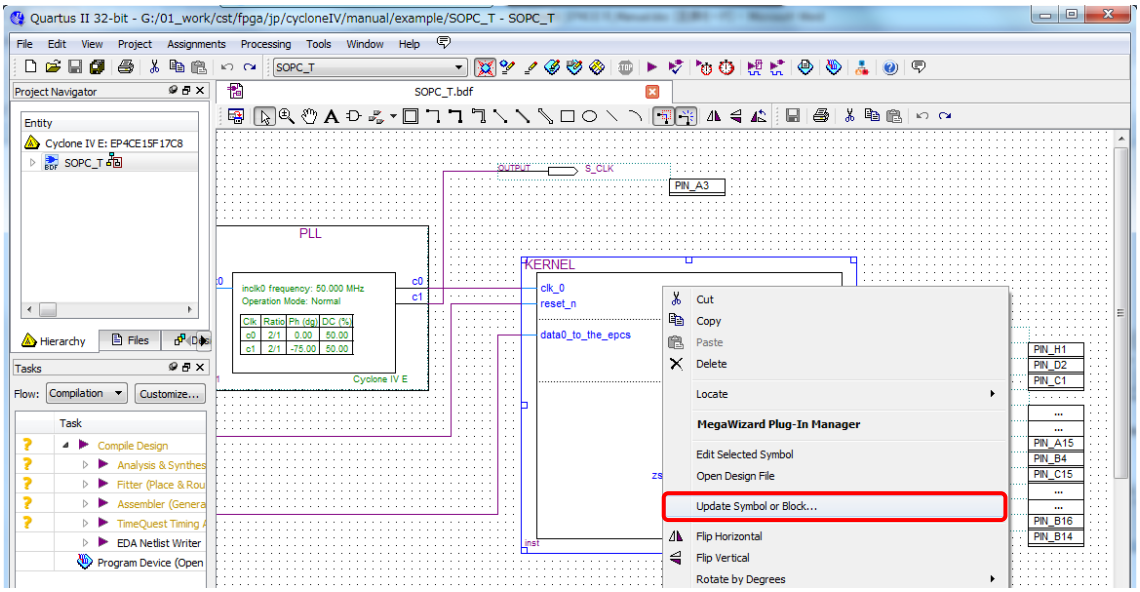


暫く待って、下記の生成完了画面が出来ます。「Exit」ボタンをクリックして「SOPC Builer」を閉じます。

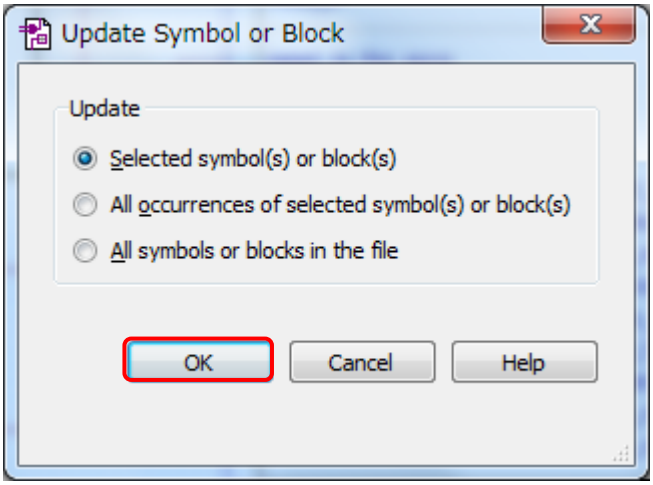


更新後の KERNEL を bdf に反映

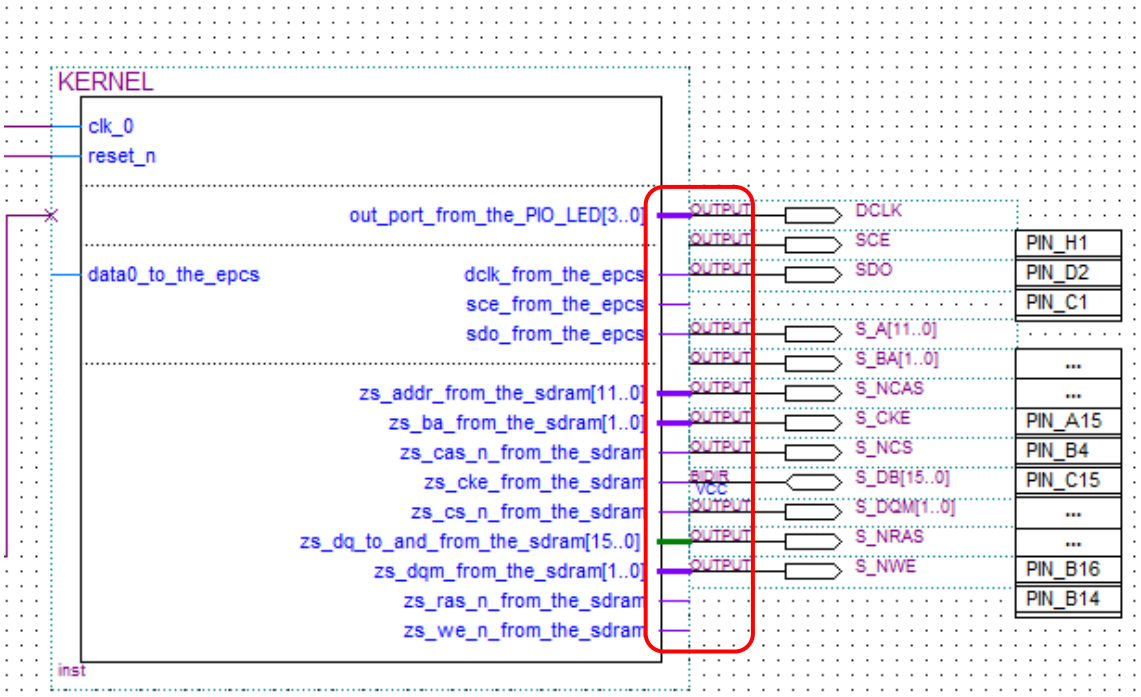
bdf エディタの KERNEL を右クリック、「Update Symbol or Block」を選択



そのまま「OK」ボタンをクリック

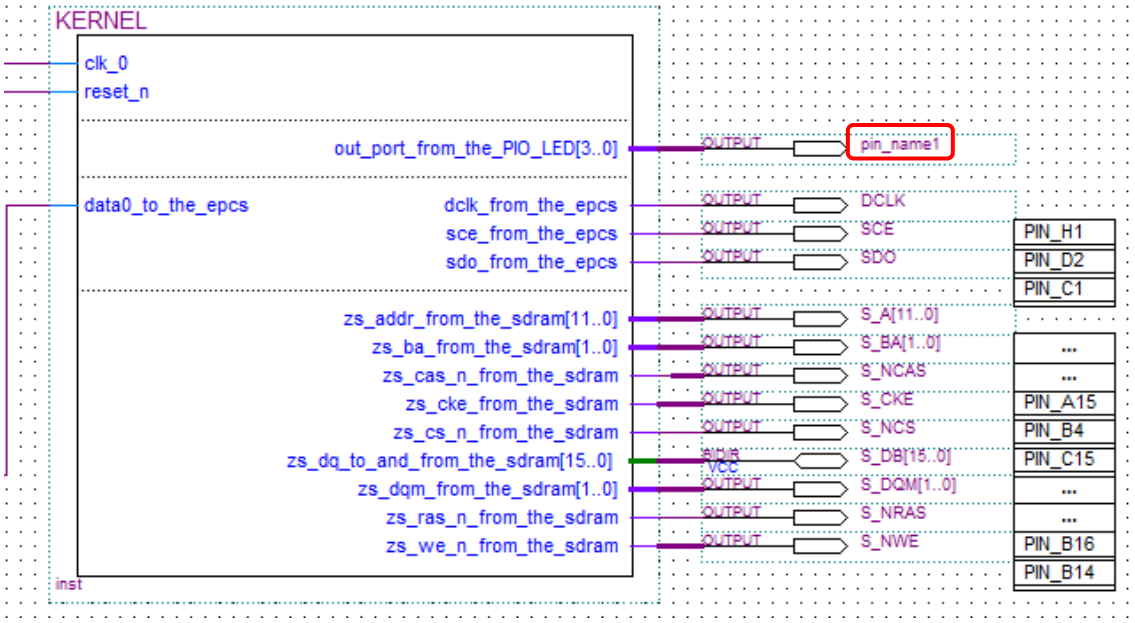


新しいピン「PIO_LED」が反映されますが、ピンのラインがずれになります、PIO_LED ピンの追加と合わせて手動で調整しましょう。

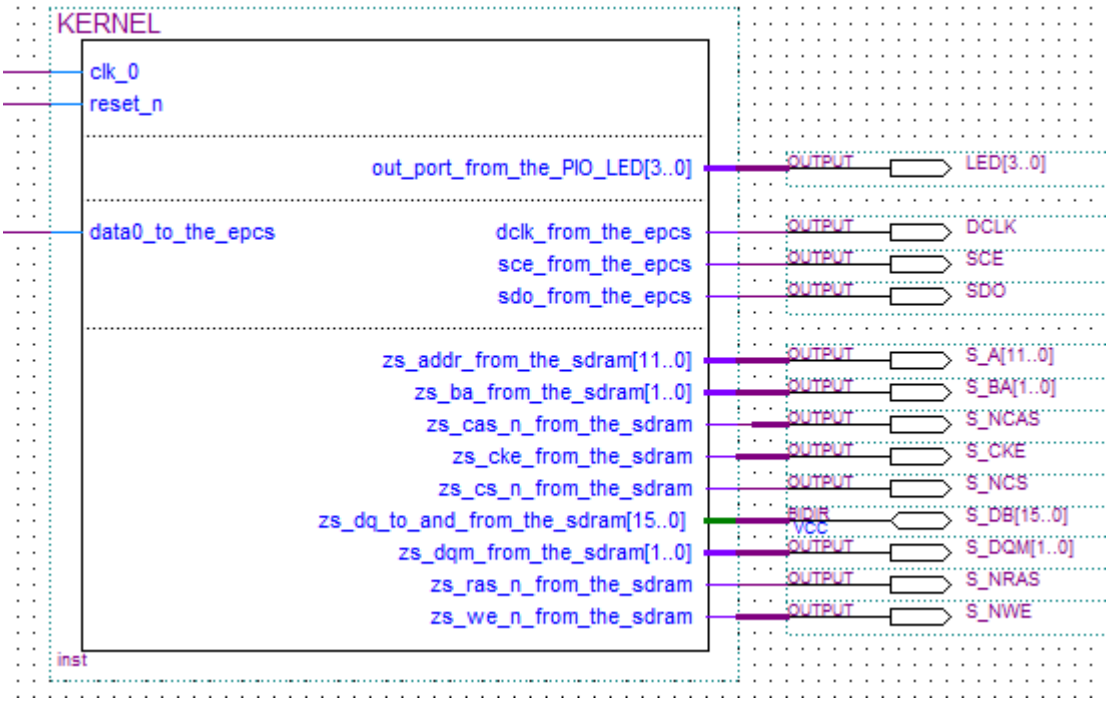


調整後の様子、「PIO_LED」出力ピンの名前を「LED[3..0]」に修正してください。

※名前の所を右クリックしてから「Properties」を選択し名前を修正できます。

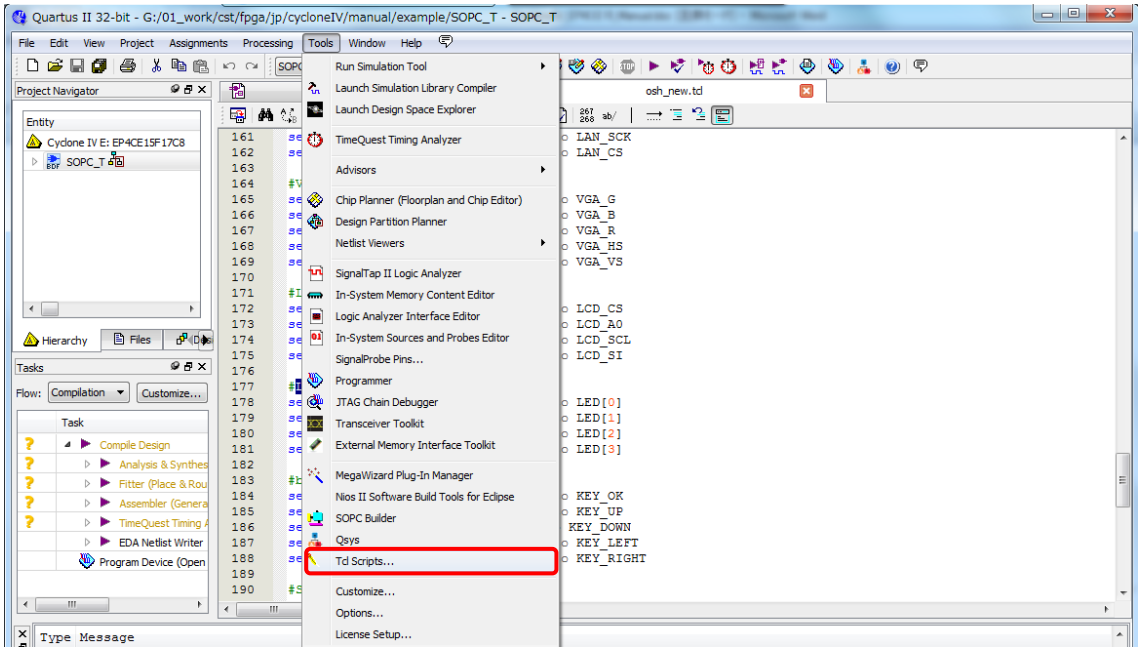


名前を修正後

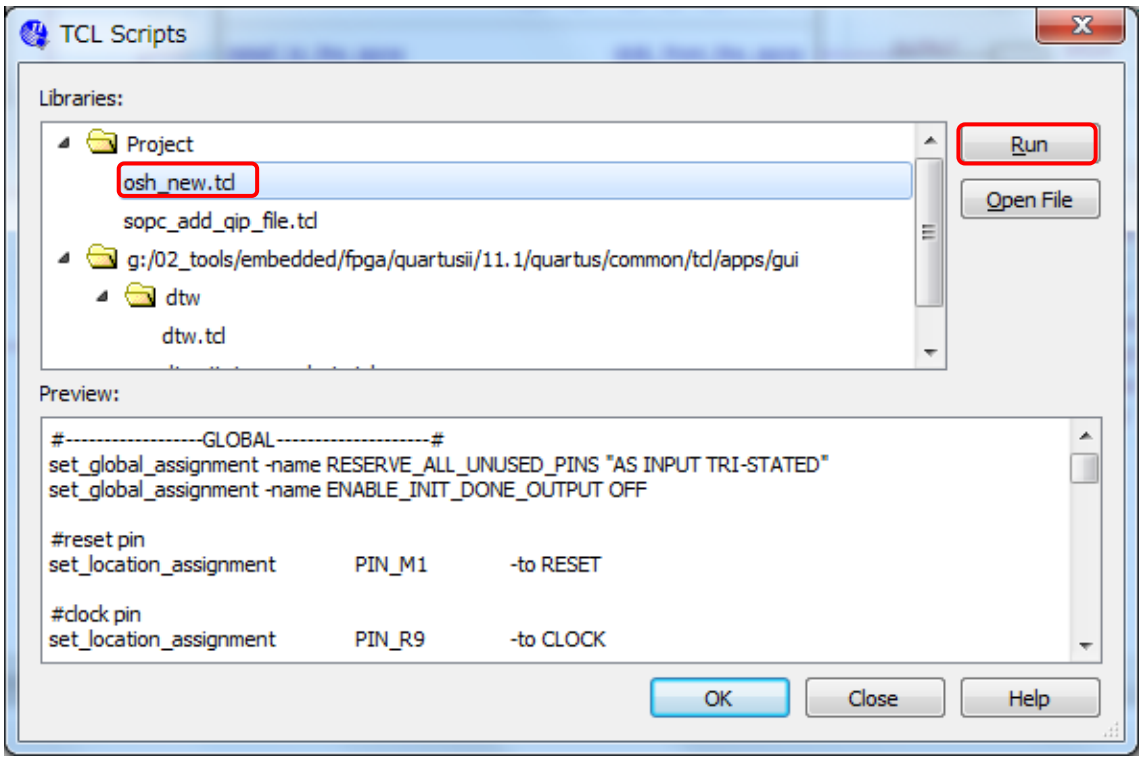


TCL ファイルによりピンを割り当てます。

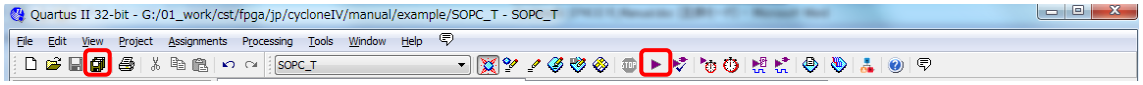
メニュー「Tools」→「Tcl Scripts」をクリック



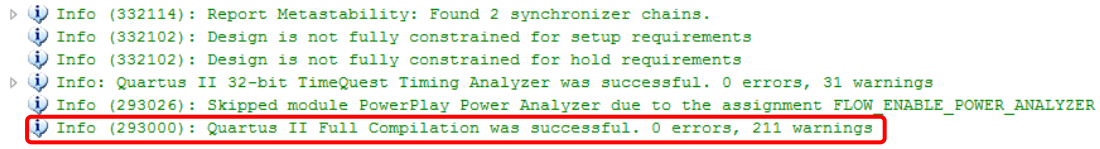
「Project」→「osh_new.td」を選択してから「Run」ボタンを押す



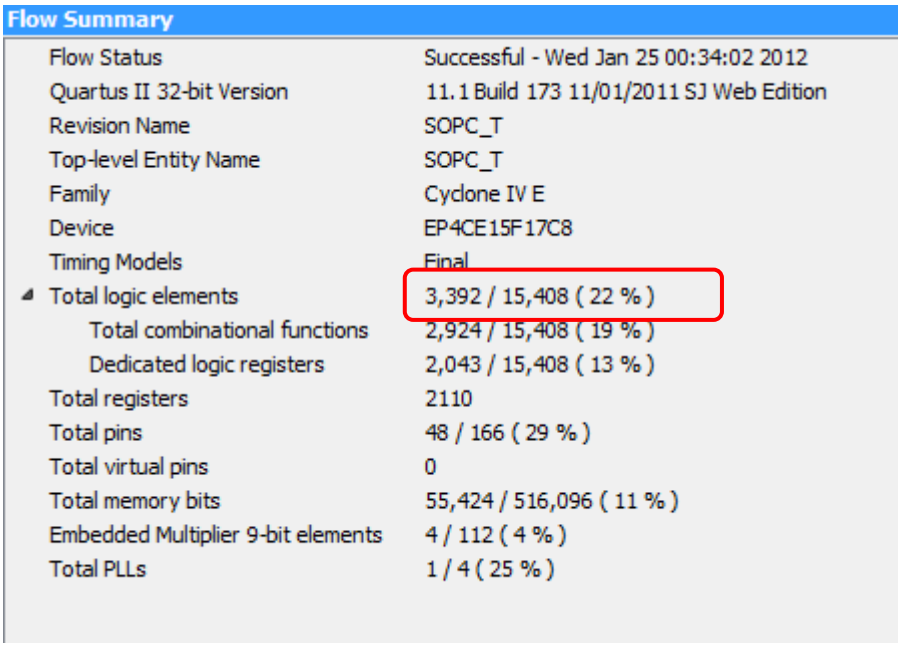
次は保存してからコンパイルしましょう。



下記のコンパイル完了画面が出来ます。



リソースは PIO を追加前と殆ど変りません。（一つ PIO モジュールが使わるリソースが少ない）



Flow Summary	
Flow Status	Successful - Wed Jan 25 00:34:02 2012
Quartus II 32-bit Version	11.1 Build 173 11/01/2011 SJ Web Edition
Revision Name	SOPC_T
Top-level Entity Name	SOPC_T
Family	Cyclone IV E
Device	EP4CE15F17C8
Timing Models	Final
Total logic elements	3,392 / 15,408 (22 %)
Total combinational functions	2,924 / 15,408 (19 %)
Dedicated logic registers	2,043 / 15,408 (13 %)
Total registers	2110
Total pins	48 / 166 (29 %)
Total virtual pins	0
Total memory bits	55,424 / 516,096 (11 %)
Embedded Multiplier 9-bit elements	4 / 112 (4 %)
Total PLLs	1 / 4 (25 %)

7.1.3 ソフトウエア開発

NIOS II IDE11.1 を起動しましょう。起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、何にか変わったか、system.h ファイルの内容を確認しましょう。

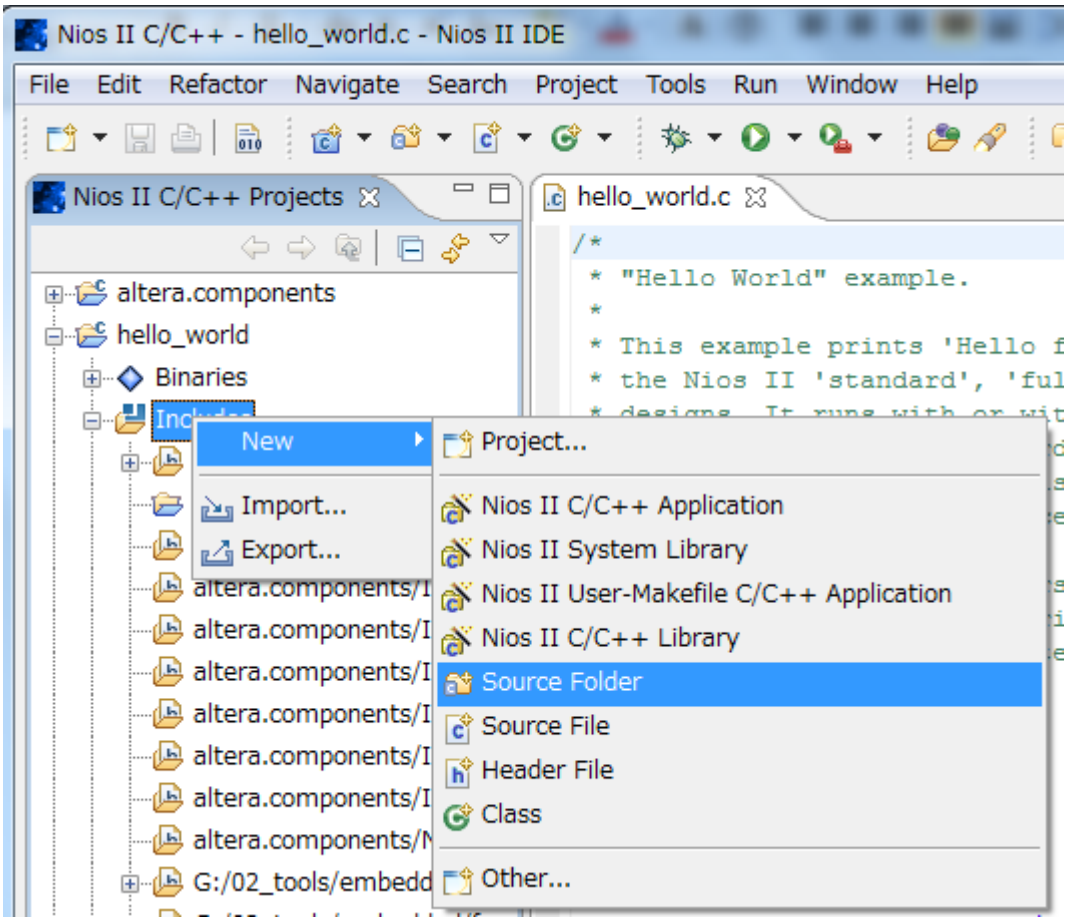
この前のファイルと比べて、下記内容が追加されました。これは PIO_LED モジュールを追加したことにより新しい内容です。これから PIO_LED_BASE を使う必要です、これは PIO_LED の全てレジスタのベースアドレスです。では、このアドレスを使って四つ LED をコントロールにしましょう。

```
/*
 * PIO_LED configuration
 *
 */

#define PIO_LED_NAME "/dev/PIO_LED"
#define PIO_LED_TYPE "altera_avalon_pio"
#define PIO_LED_BASE 0x00000800
#define PIO_LED_SPAN 16
#define PIO_LED_DO_TEST_BENCH_WIRING 0
#define PIO_LED_DRIVEN_SIM_VALUE 0
#define PIO_LED_HAS_TRI 0
#define PIO_LED_HAS_OUT 1
#define PIO_LED_HAS_IN 0
#define PIO_LED_CAPTURE 0
#define PIO_LED_DATA_WIDTH 4
#define PIO_LED_RESET_VALUE 0
#define PIO_LED_EDGE_TYPE "NONE"
#define PIO_LED_IRQ_TYPE "NONE"
#define PIO_LED_BIT_CLEARING_EDGE_REGISTER 0
#define PIO_LED_BIT_MODIFYING_OUTPUT_REGISTER 0
#define PIO_LED_FREQ 100000000
#define ALT_MODULE_CLASS_PIO_LED altera_avalon_pio
```

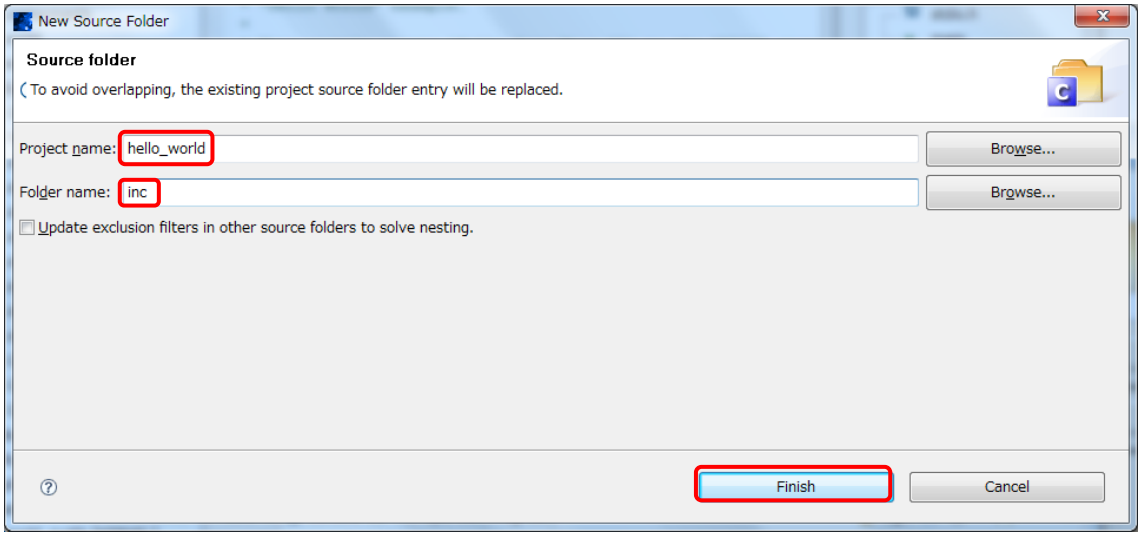
次はプログラムを作成して行きます。

１．ヘッダファイル格納フォルダー作成

左側の「hello_world」→「Includes」を右クリック―>「New」→「Source　Folder」をクリック

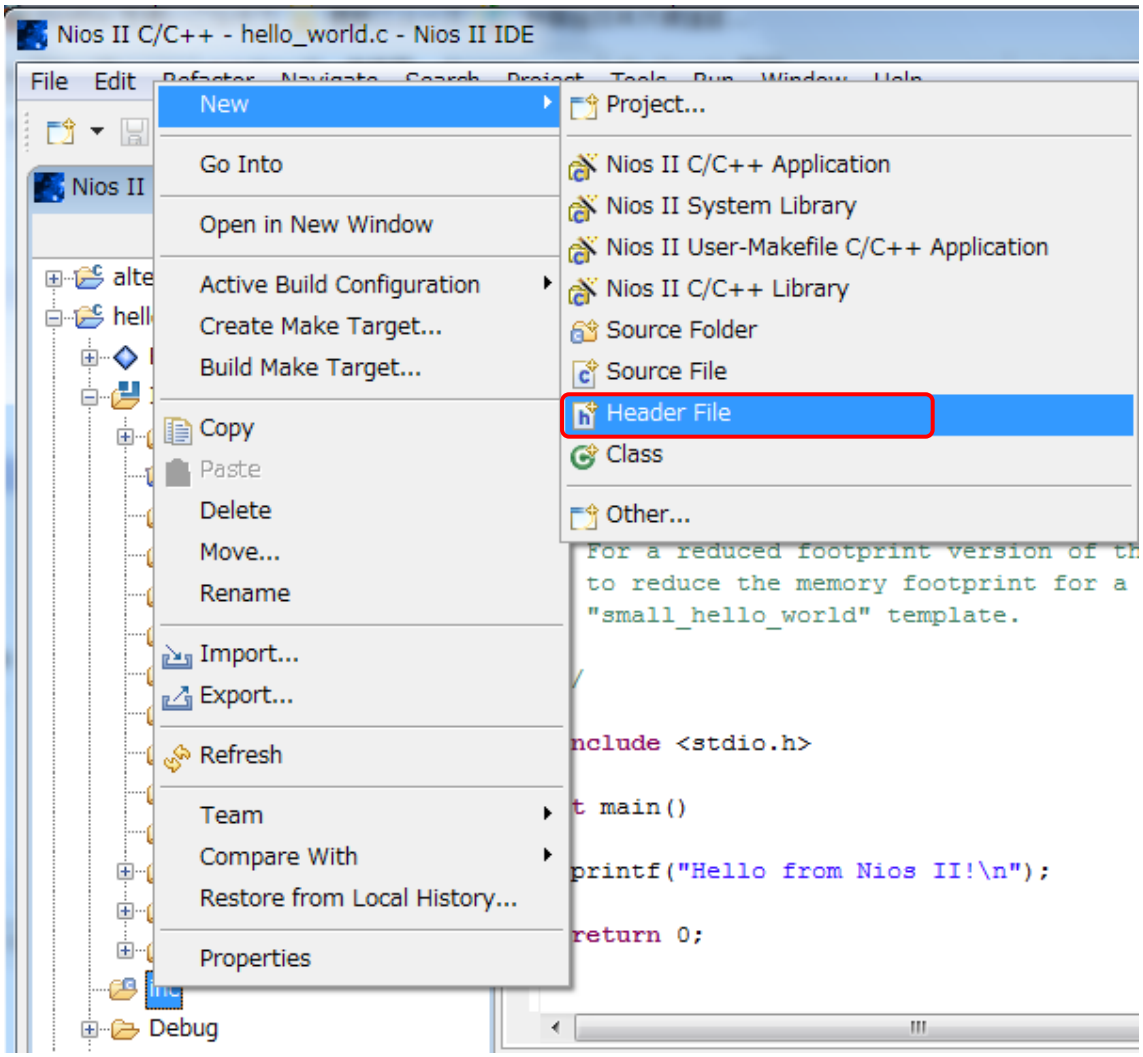


Project name:hello_world、Folder name：inc を入れ、「Finish」ボタンをクリック

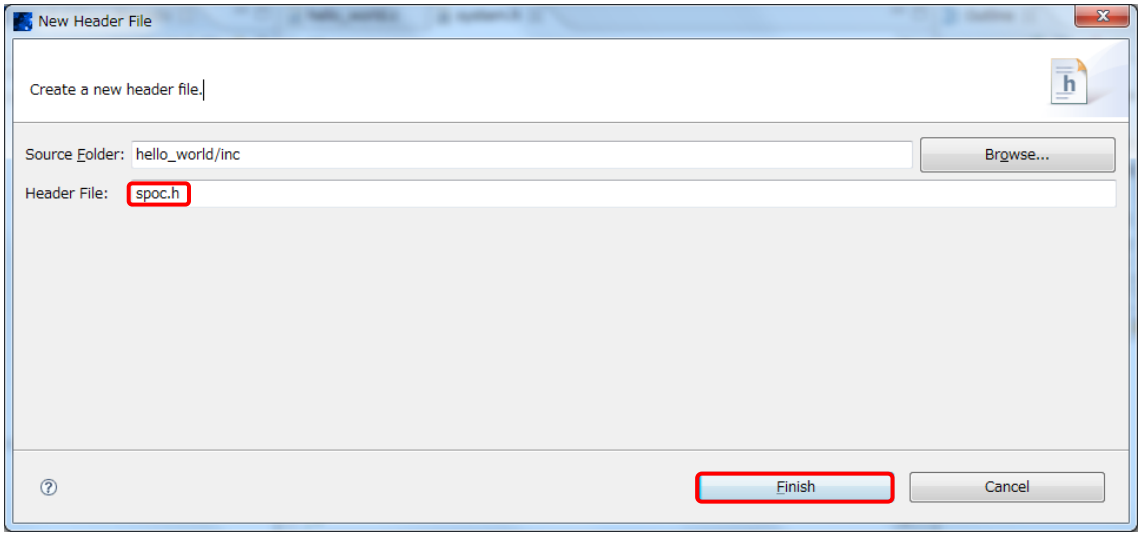


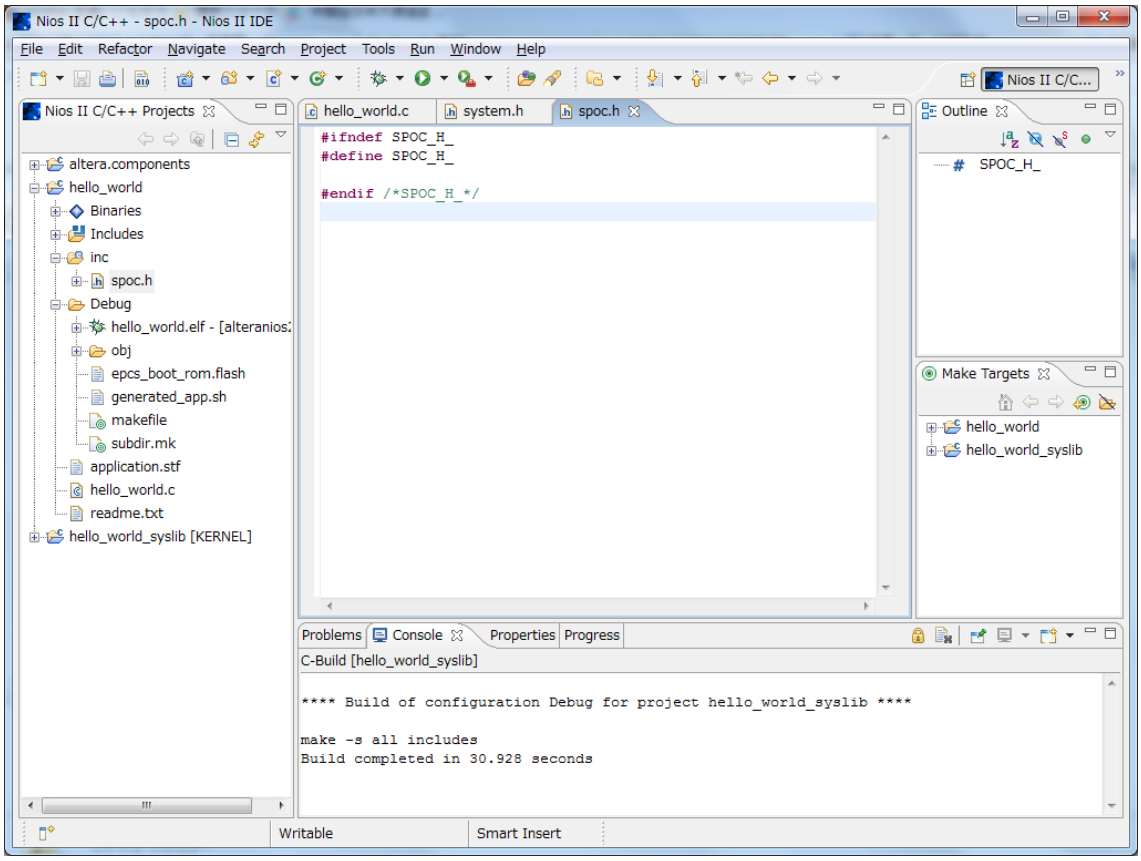
２．ヘッドファイル作成

左側の「hello_world」→「Includes」→「inc」を右クリック—>「New」→「Header File」をクリック



Header File に「spoc.h」を記入、「Finish」ボタンをクリック




Nios II C/C++ - spoc.h - Nios II IDE
File Edit Refactor Navigate Search Project Tools Run Window Help
Nios II C/C++ Projects
altera.components
hello_world
Binaries
Includes
inc
spoc.h
Debug
hello_world.elf - [alteranios2
obj
epcs_boot_rom.flash
generated_app.sh
makefile
subdir.mk
application.stf
hello_world.c
readme.txt
hello_world_syslib [KERNEL]
hello_world.c
system.h
spoc.h
#ifndef SPOC_H_
#define SPOC_H_
#endif /*SPOC_H_*/
Outline
SPOC_H_
Make Targets
hello_world
hello_world_syslib
Problems
Console
Properties
Progress
C-Build [hello_world_syslib]
**** Build of configuration Debug for project hello_world_syslib ****
make -s all includes
Build completed in 30.928 seconds
Writable
Smart Insert

ソースを修正

hello_world.c | main.c | sopc.h | system.h

```
#ifndef SOPC_H_
#define SOPC_H_

/*-------------------------------------------------------------------------
 *  Include
 *-------------------------------------------------------------------------*/
#include "system.h"

/*-------------------------------------------------------------------------
 *  Define
 *-------------------------------------------------------------------------*/
#define _LED

/*-------------------------------------------------------------------------
 *  Peripheral registers structures
 *-------------------------------------------------------------------------*/
typedef struct
{
    unsigned long int DATA;
    unsigned long int DIRECTION;
    unsigned long int INTERRUPT_MASK;
    unsigned long int EDGE_CAPTURE;

}PIO_STR;

/*-------------------------------------------------------------------------
 *  Peripheral declaration
 *-------------------------------------------------------------------------*/
#ifdef _LED
#define LED          ((PIO_STR *) PIO_LED_BASE)
#endif /*_LED*/

#endif /*SOPC_H_*/
```

①

②

③

ソースコードの説明：

①：LED 用のマイクロを定義（③の制御するため）

②：この構造体は NIOS II チップのデータシート《Embedded Peripherals IP User Guide》により宣言されます。（下図はデータシートから抜粋もの）

《Embedded Peripherals IP User Guide》：

http://www.altera.com/literature/ug/ug_embedded_ip.pdf

下の内容は PIO Core レジスタのマッピングです、構造体はこの内容により作成されたものです、重要なものは順番、即ち offset です。

1 番：data、2 番：IO 方向、3 番：割り込み制御、4 番：エッジ制御、この 4 つ項目は全て n 桁があります、ここの n がソフトマクロ作成時に設定された Width です、ハードウエア作成時に 4 が設定されますので、n が 4 に設定にします。

③：PIO_LED のベースアドレスをマイクロで定義

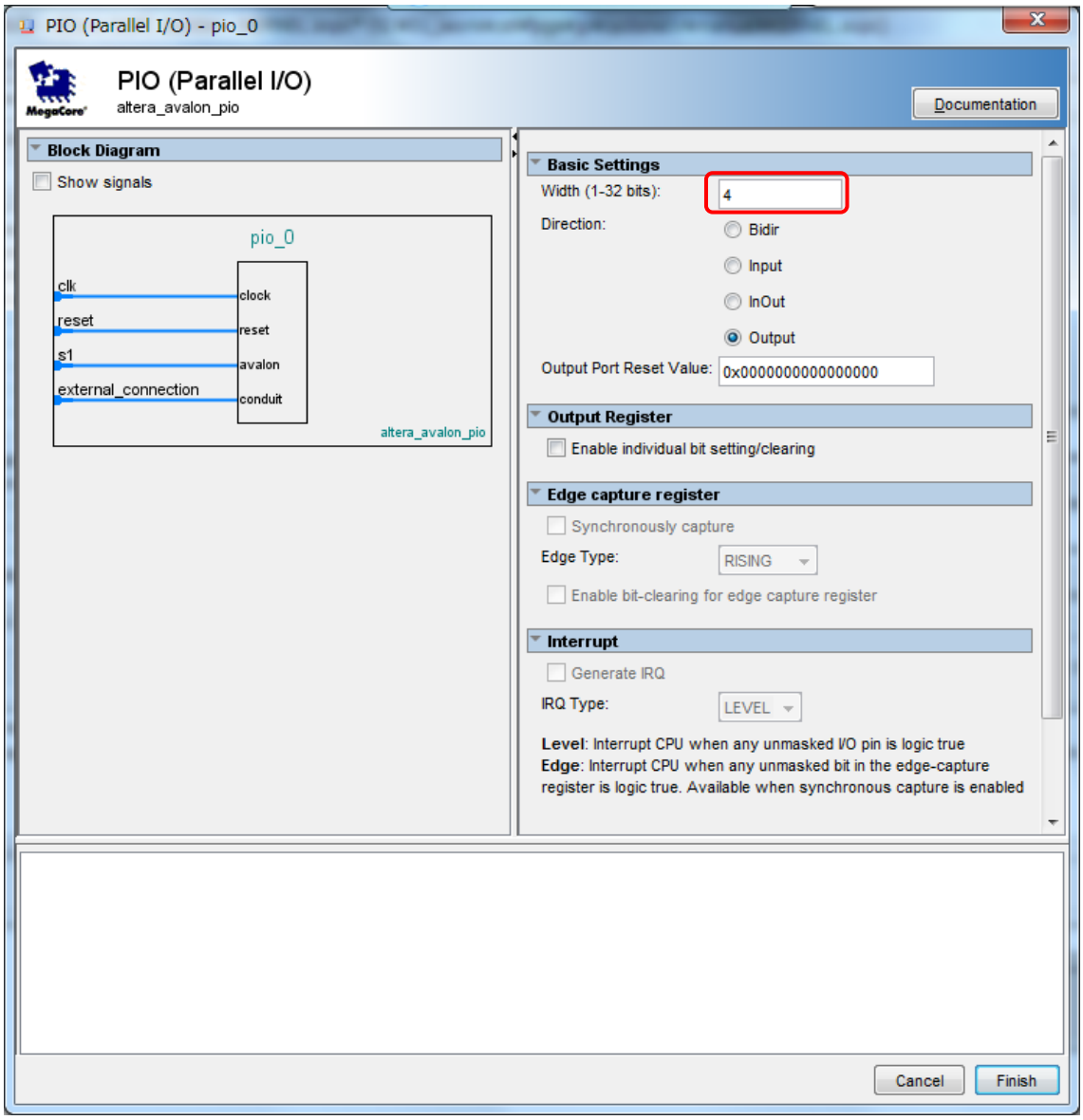
Table 10–2. Register Map for the PIO Core

Offset	Register Name		R/W	Fields				
				(n-1)	...	2	1	0
0	data	read access	R	Data value currently on PIO inputs.				
		write access	W	New value to drive on PIO outputs.				
1	direction *(1)*		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				
2	interruptmask *(1)*		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	edgecapture *(1)*, *(2)*		R/W	Edge detection for each input port.				
4	outset		W	Specifies which bit of the output port to set.				
5	outclear		W	Specifies which output bit to clear.				

Notes to Table 10–2:

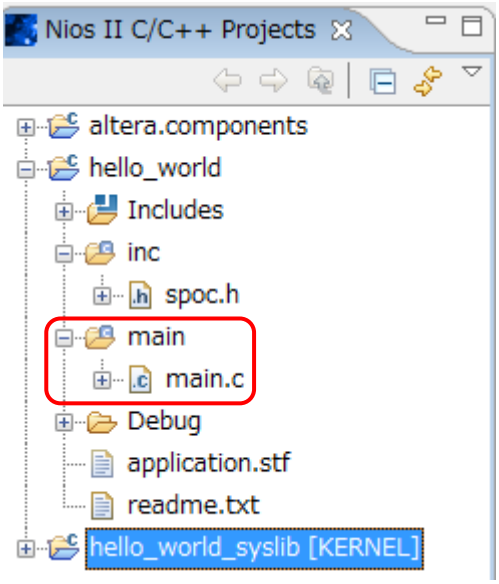
(1) This register may not exist, depending on the hardware configuration. If a register is not present, reading the register returns an undefined value, and writing the register has no effect.

(2) Writing any value to edgecapture clears all bits to 0.



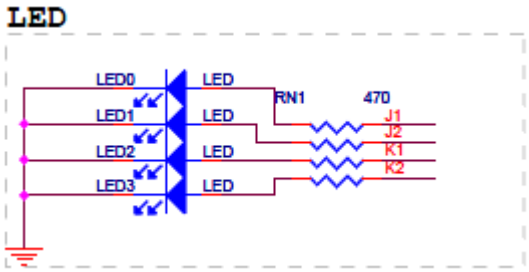
３．C ソース修正

hello_world というプロジェクトに基づき修正しますが、可読性を高めるため、hello_world.c という名前を「main.c」に変更。そして、hello_world の直下「main」、「driver」フォルダを作成してから名前を変更後の「main.c」を「main」フォルダーに移動します。



このプログラムは 4 つ LED を順次に点灯するように修正しましょう。

コアボード回路図により、FPGA の該当のピンがレベル 1 に設定される場合、LED が点灯、レベル 0 に設定すれば、LED が消灯。



修正後のソースコード：

説明：

①ヘッダファイルをインクルード（相対パスで移植性が高い）

②LED->DATA = 1 << i;

LED はヘッダファイルに定義されたマイクロでしょう、構造体のポインタですよね。

4 つ LED が順次に繰り返して 1 を設定にします。

usleep は ns 単位で延びます、ここに 5ms を設定されます。

hello_world.c | hello_world.c.bak | *main.c

```
/*---------------------------------------------------------------------
 * Include
 *--------------------------------------------------------------------*/
#include "../inc/sopc.h"
#include <unistd.h>

/*
 * ===  FUNCTION  ======================================================
 *         Name:  main
 *  Description:
 * =====================================================================
 */
int main(void)
{
    int i;

    while(1){
        for(i=0;i<4;i++){
            LED->DATA = 1 << i;
            usleep(100000);
        }
    }

    return 0;
}
```

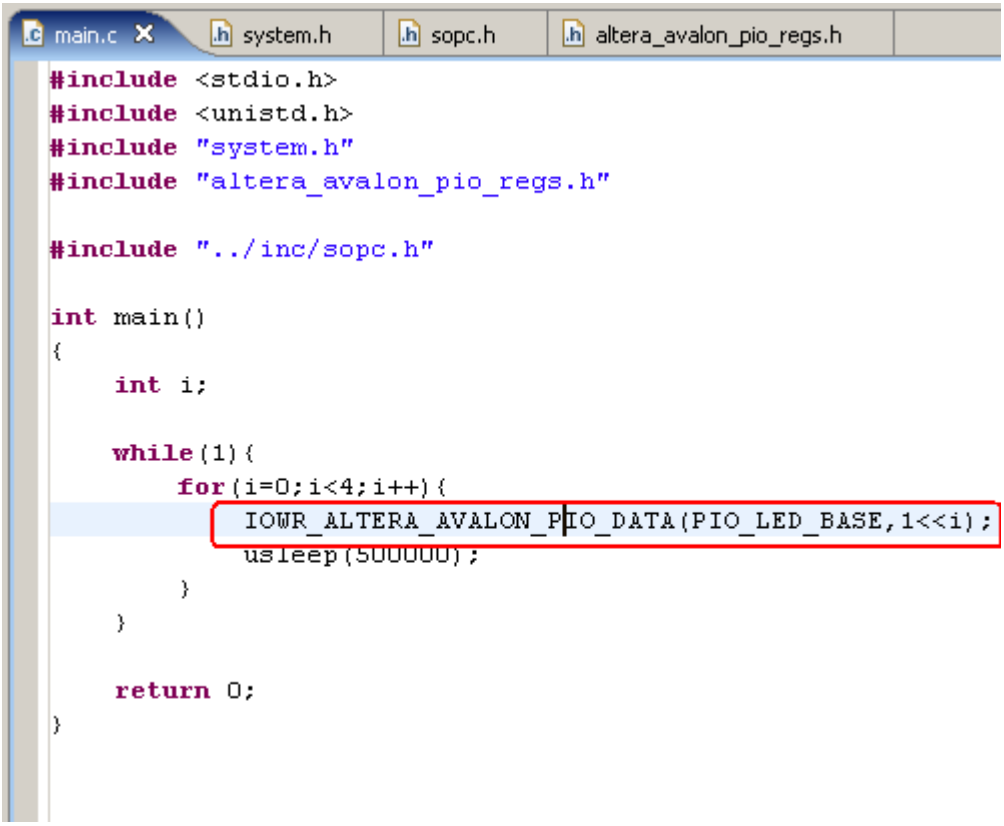
①

②

最後、コンパイルしましょう、コンパイル出来たら、第五章のダウンロード方法を参照してボードにダウンロードして実行してみましょう。

４．アルテラ社の API と比較

アルテラ社の API でも実現できます、ソースは下記通りです。



IOWR_ALTERA_AVALON_PIO はマイクロです、これは「altera_avalon_pio_regs.h」に定義されます。皆様が興味があれば、NIOS ソースコードを見てください。このマイクロはハードウエアの操作を実現します、我々はレジスタを直接制御します。勿論、全て操作はメーカの API を利用しないわけではない、例えば、Flash の操作であれば、複雑しますので、逆に自分で実現すれば、大変でしょう。

```
#ifndef __ALTERA_AVALON_PIO_REGS_H__
#define __ALTERA_AVALON_PIO_REGS_H__

#include <io.h>

#define IOADDR_ALTERA_AVALON_PIO_DATA(base)          __IO_CALC_ADDRESS_NATIVE(base, 0)
#define IORD_ALTERA_AVALON_PIO_DATA(base)            IORD(base, 0)
#define IOWR_ALTERA_AVALON_PIO_DATA(base, data)      IOWR(base, 0, data)
```

7.2 割り込み

これから NIOS II のハードウエア割り込み内容を説明します、この節を通じて、NIOS II IDE のオンラインデバッグ方法とノウハウも紹介します。

7.2.1 概要

割り込み基本知識：

ISR(Interrupt Service Routine)割り込みサービス関数はハードウエア割り込みためのサブプログラムです。NIOS II　プロセッサは 32 個ハードウエア割り込みをサポートします、

有効にする各割り込みは該当の ISR があります。割り込みが発生する時、ハードウェア割り込みハンドラは検知した有効の割り込みレベルにより該当の ISR を呼び出し割り込みサービスを実施します。

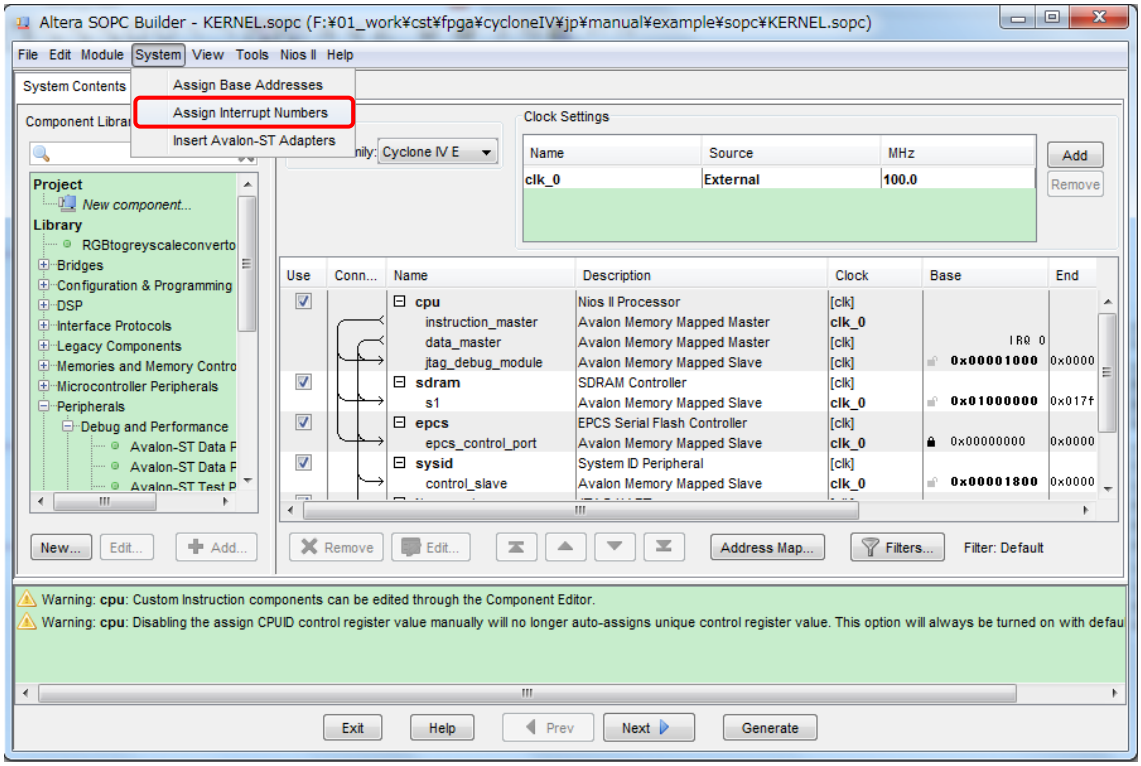
ハードウエア割り込みサービスを行えるため、2 点を実施する必要です。

①割り込み関数 ISR をレジスタ

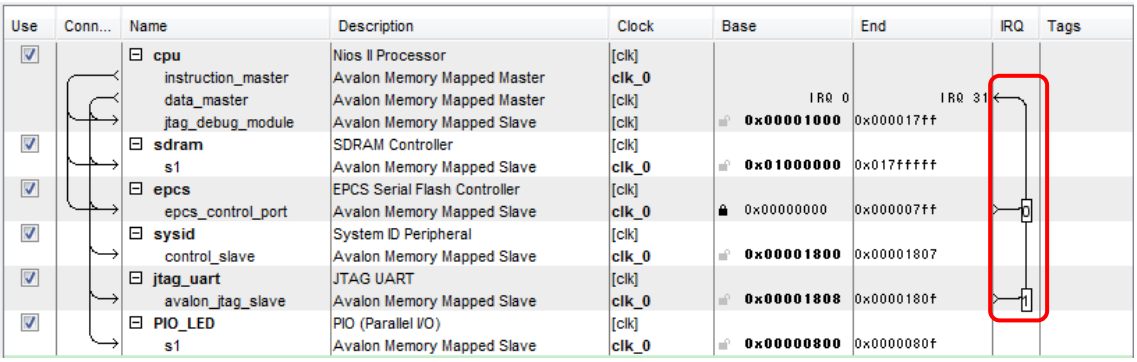
関数のプロタイプ：Int alt_irq_register(alt_u32 id, void* context, void(*handler)(void*,alt_u32));

id:割り込み優先度、即ちレジスタの ISR はどんな優先度の割り込みを実施します。

割り込み優先度は SOPC Builder の中割り当てられます。前章に下図のように割り込み優先度の割り当てることを覚えるでしょう。



割り当てられた後の状態：(IRQ が重複しなければ、皆様のニーズにより手動で割り当てても OK です)



NIOS II IDE に「system.h」というファイルから見えます。IRQ は両方も 1 になります。理由は NIOS II IDE の内容は SOPC Builder で作成したソフトマクロにより自動生成されます。

```
/*
 * jtag_uart configuration
 *
 */

#define JTAG_UART_NAME "/dev/jtag_uart"
#define JTAG_UART_TYPE "altera_avalon_jtag_uart"
#define JTAG_UART_BASE 0x00001808
#define JTAG_UART_SPAN 8
#define JTAG_UART_IRQ 1
#define JTAG_UART_IRQ_INTERRUPT_CONTROLLER_ID 0
#define JTAG_UART_WRITE_DEPTH 64
#define JTAG_UART_READ_DEPTH 64
#define JTAG_UART_WRITE_THRESHOLD 8
#define JTAG_UART_READ_THRESHOLD 8
#define JTAG_UART_READ_CHAR_STREAM ""
#define JTAG_UART_SHOWASCII 1
#define JTAG_UART_RELATIVEPATH 1
#define JTAG_UART_READ_LE 0
#define JTAG_UART_WRITE_LE 0
#define JTAG_UART_ALTERA_SHOW_UNRELEASED_JTAG_UART_FEATURES 1
#define ALT_MODULE_CLASS_jtag_uart altera_avalon_jtag_uart
```

他の二つパラメータ：

context：レジスタの ISR に渡すパラメータ、NULL でも OK。

handler：割り込みサービス関数のポインタ

戻し値：レジスタが成功の場合、0 に戻す、失敗の場合、マイナスの値に戻す。

注意点：

handler が NULL ではない場合、レジスタ成功後優先度割り込みを自動に有効になります、即ち、handler のポインタに該当の ISR があれば、有効にする処理が不要です。

②ISR 関数は皆様が自分で作成

HAL システムから提供するわけではありません。

関数プロタイプ：void ISR_handler(void* context, alt_u32 id);

context：ISR に渡すパラメータ、NULL でも OK

id:割り込み優先度

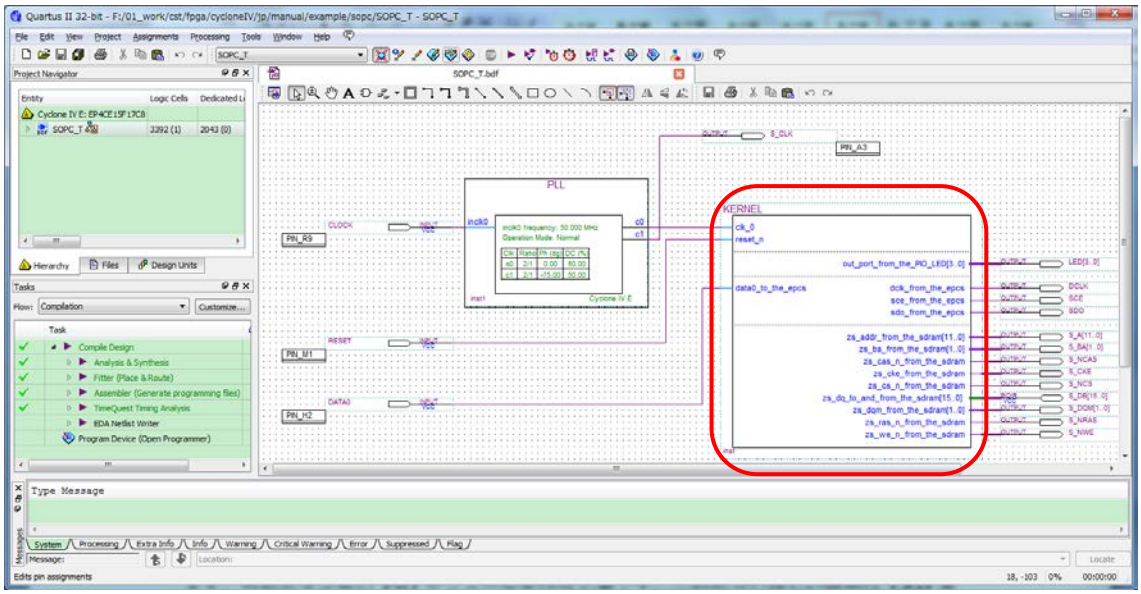
7.2.2 ハードウエア開発

1．PIO モジュールを IP コアに追加

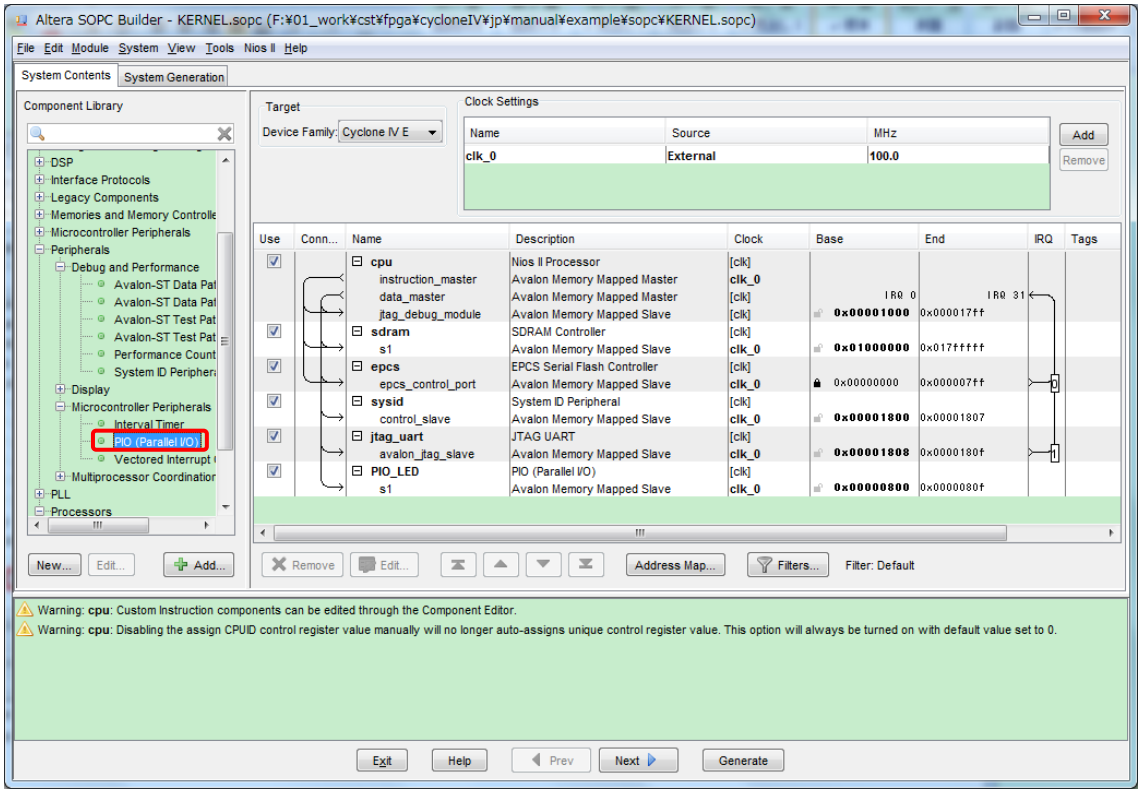
※作成方法は LED 実例の「1．PIO モジュールを IP コアに追加」と同じです。

「ハードウエア開発」という章に作成した Quartus プロジェクトを開き、KERNEL をダ

ブルクリックして SOPC Builder を起動させましょう。



SOPC Builder が起動後、左側「Peripherals」→「MicroController Peripherals」→「PIO」をダブルクリック



下図のように設定してください。

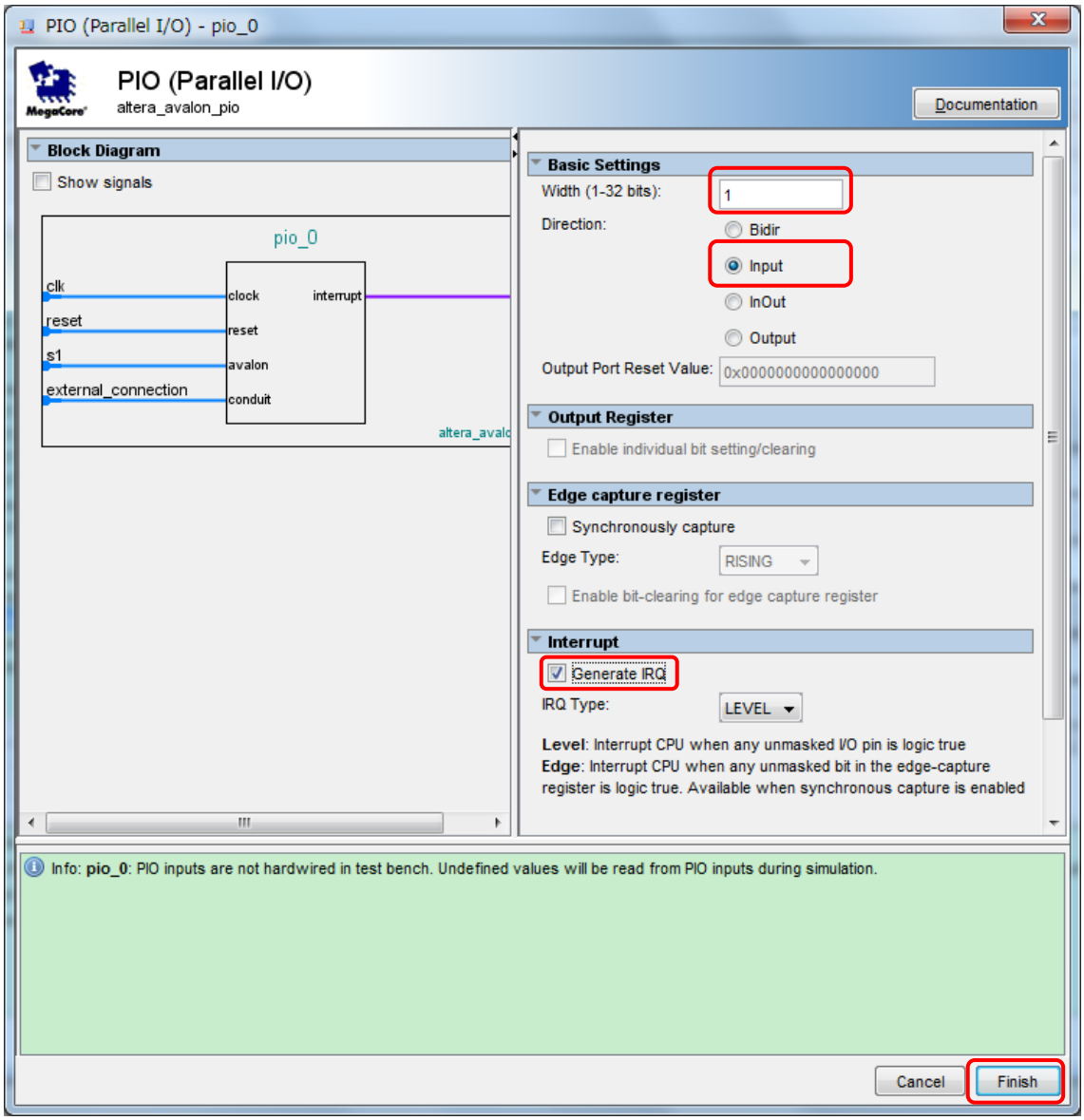
Width：1 を記入（1 個ボタンを使う）

Direction：Input をチェック

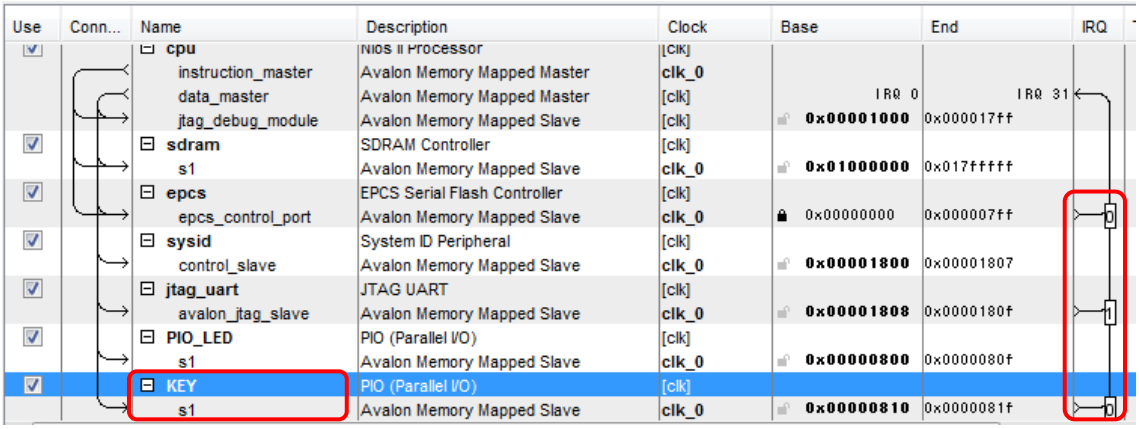
Interrupt：Generate IRQ をチェック

※割り込みは二つ方式を分けあれます。一つはレベル割り込み、即ち高レベル/低レベル割り込み、もう一つはエッジ割り込み、プラスエッジとマイナスエッジを含みます。エッジ割り込みで実現する場合、Edge capture register の「Synchronously capture」をチェックにし、Edge Type で 3 方式があります、お好きな方式を選択できます。

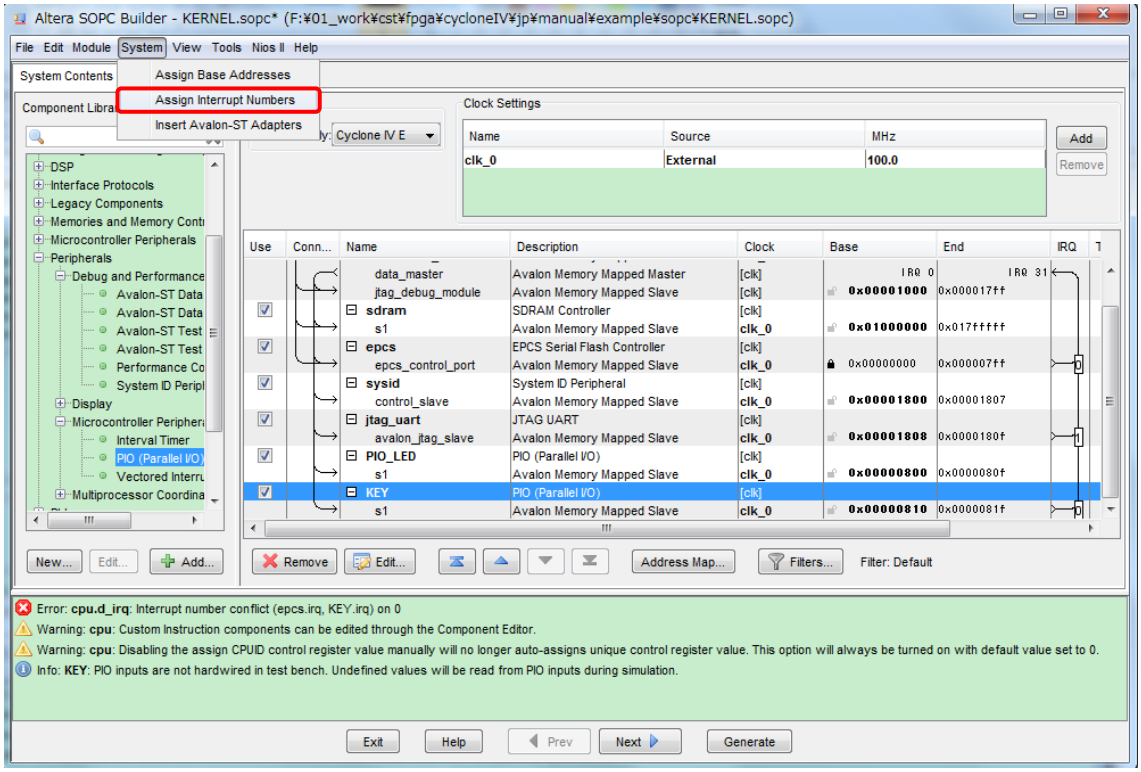
設定が完了後、「Finish」をクリック



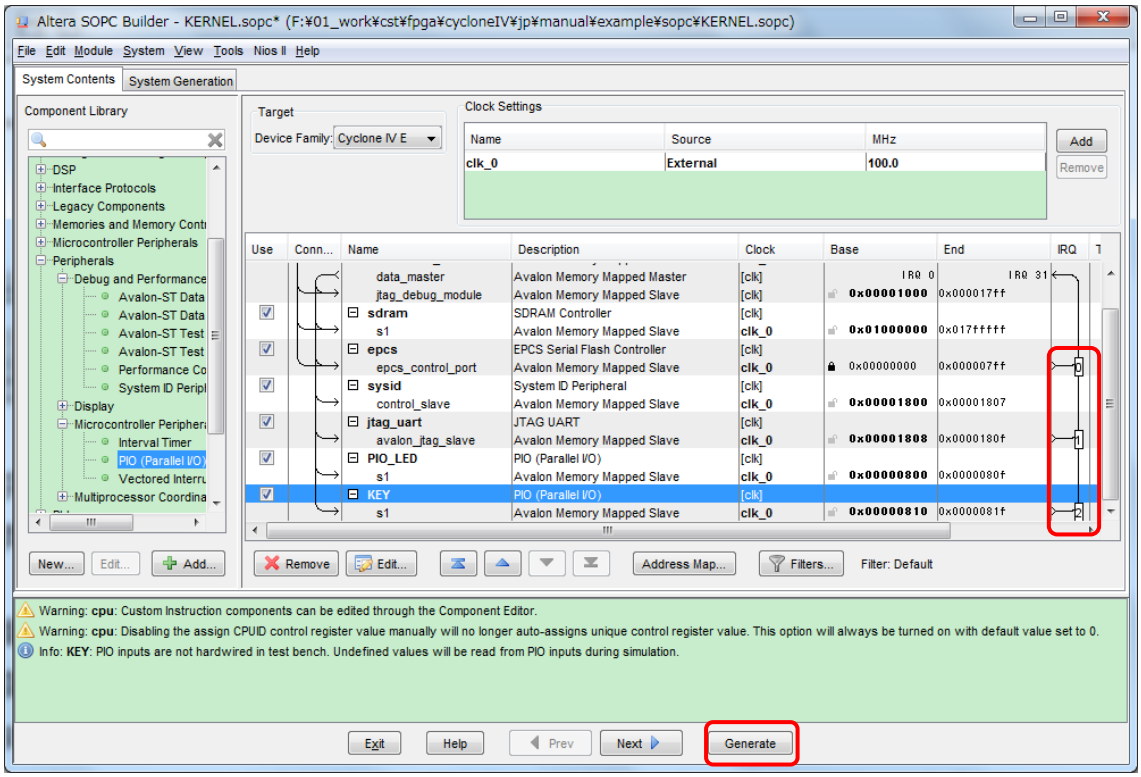
次は分かりやすくするためモジュールの名前を「KEY」に修正、あと、IRQ は同じナンバー「0」が二箇所あります、これは重複となりますので、修正必要です。（だから毎回作成完了後、自動割り当てもう一度実行必要ですようね。）



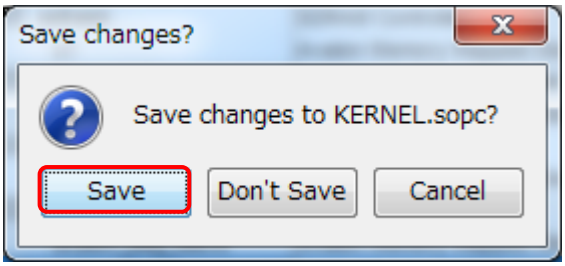
IRQ 自動割り当てにしましょう。メニュー「System」→「Assign Interrupt Numbers」をクリック



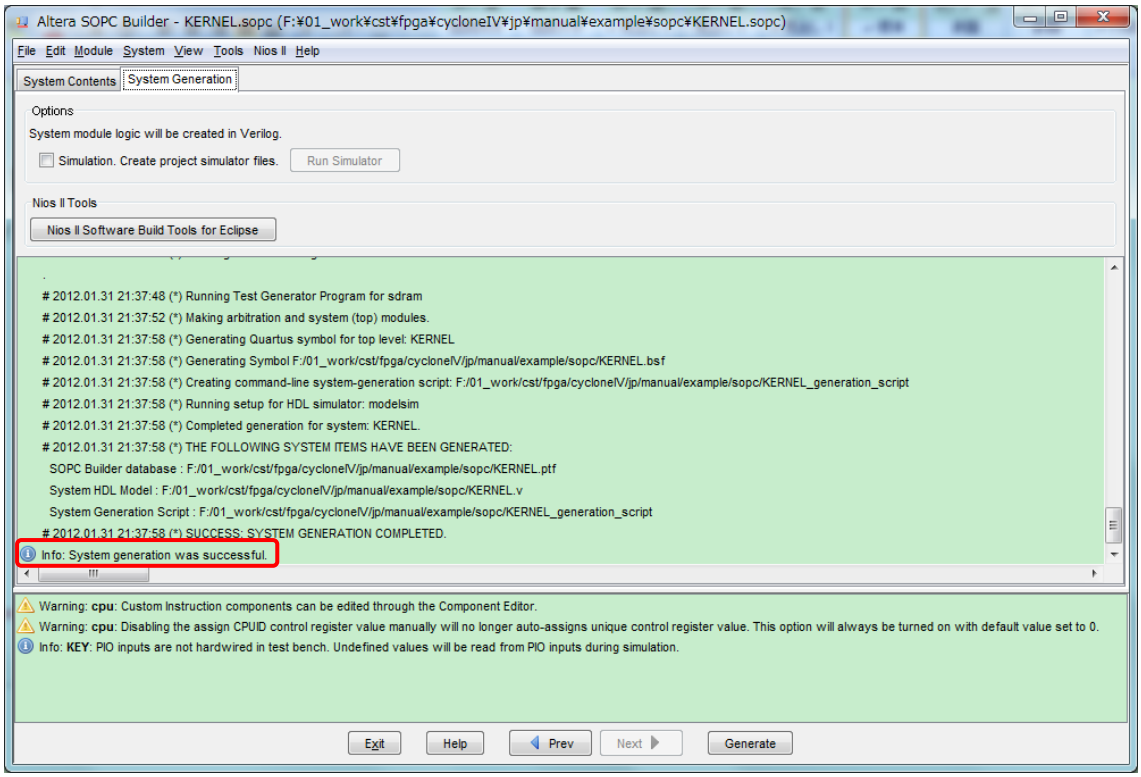
自動割り当て完了後の様子：



ソフトマクロを生成：上図の「Generate」ボタンをクリックし保存画面が出ます、「Save」ボタンも押し保存してから生成しましょう。

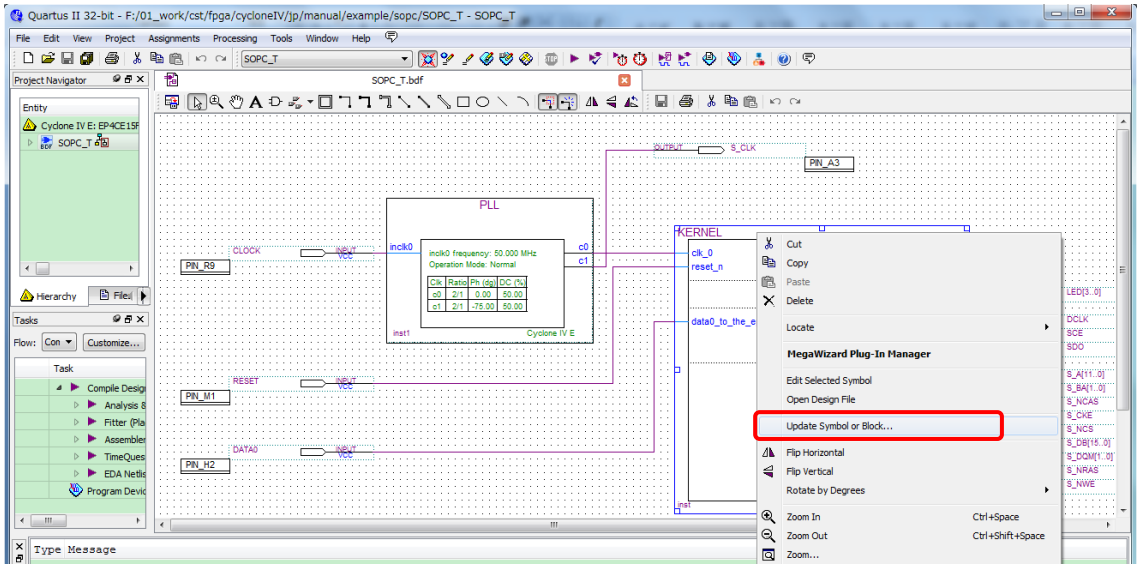


暫く待って、下記の生成完了画面が出来ます。「Exit」ボタンをクリックして「SOPC Builer」を閉じます。

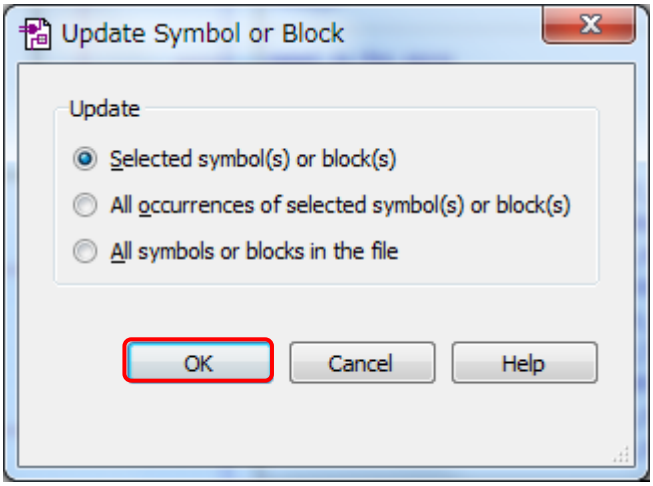


更新後の KERNEL を bdf に反映

bdf エディタの KERNEL を右クリック、「Update Symbol or Block」を選択

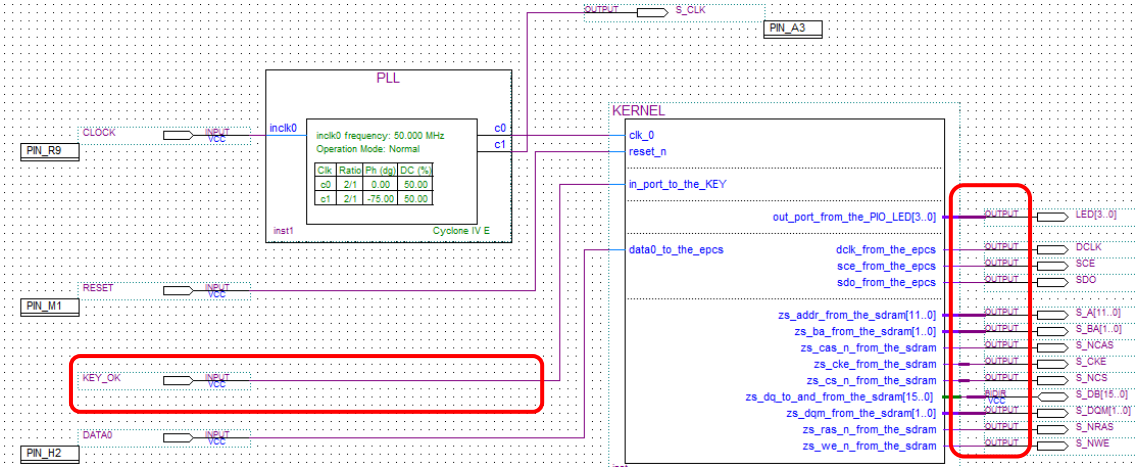


そのまま「OK」ボタンをクリック



新しいピン「KEY」が反映されますが、ピンのラインがずれになります、KEY ピンの追加と合わせて手動で調整しましょう。調整後の様子、「KEY」入力ピンの名前を「KEY_OK」に修正してください。（ここボードの OK ボタンを使います。）

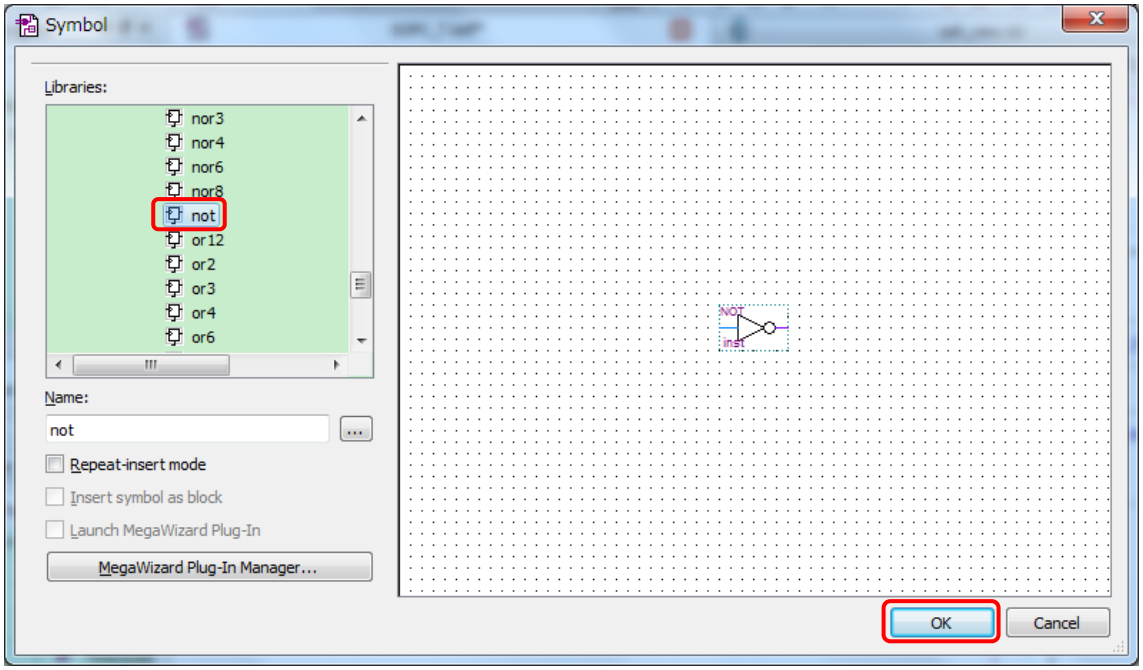
※名前の所を右クリックしてから「Properties」を選択し名前を修正できます。



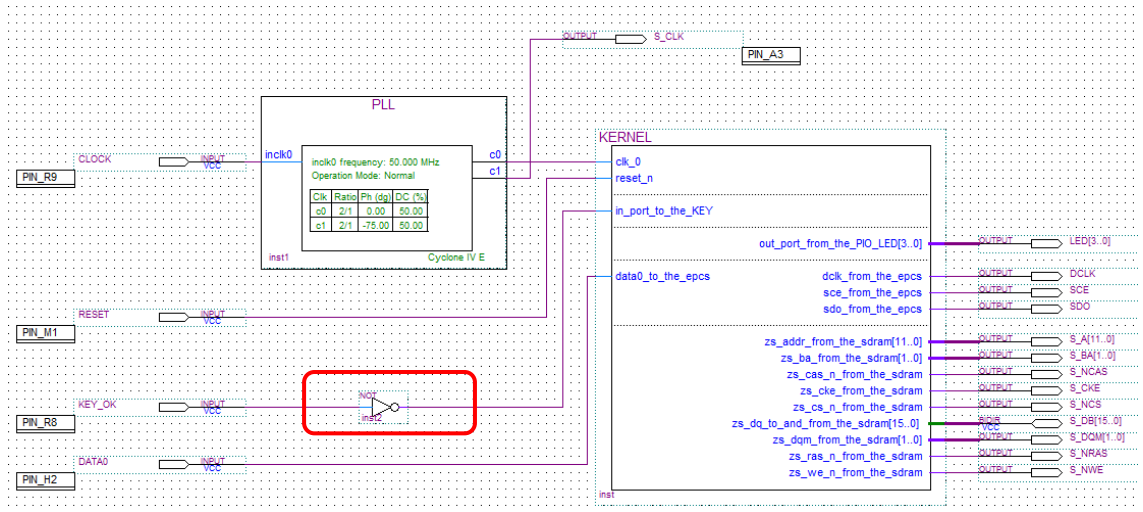
レベル割り込みの時、NIOS は高レベルのみで割り込み有効になります、低レベルに有効にするため、NOT ゲートを追加必要です。

NOT ゲートの追加方法は input ピンの追加と同じです。

空白の所にダブルクリック、「primitives」→「logic」→「not」を選択

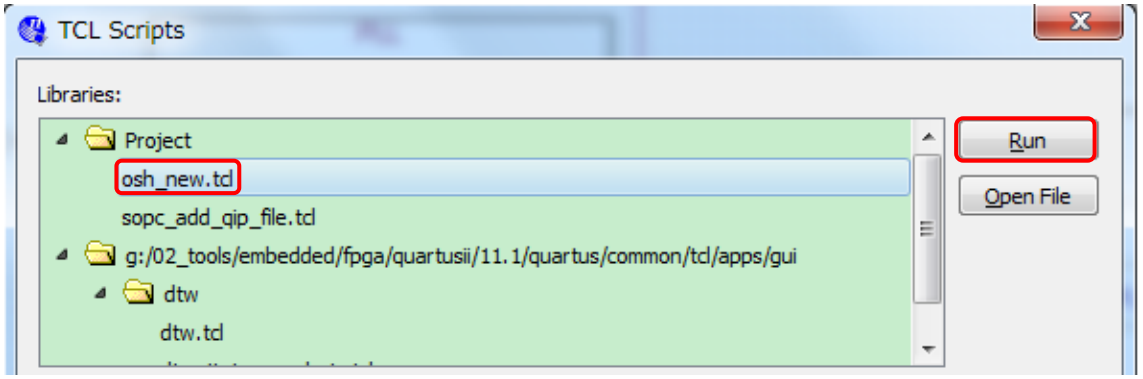


追加後の様子：

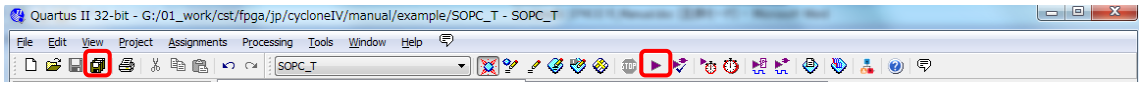


TCL ファイルによりピンを割り当てます。

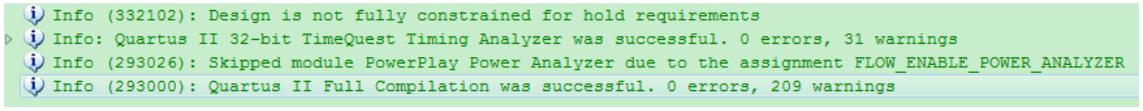
前章の同じように、メニュー「Tools」→「Tcl Scripts」をクリック、下記画面が出ます。

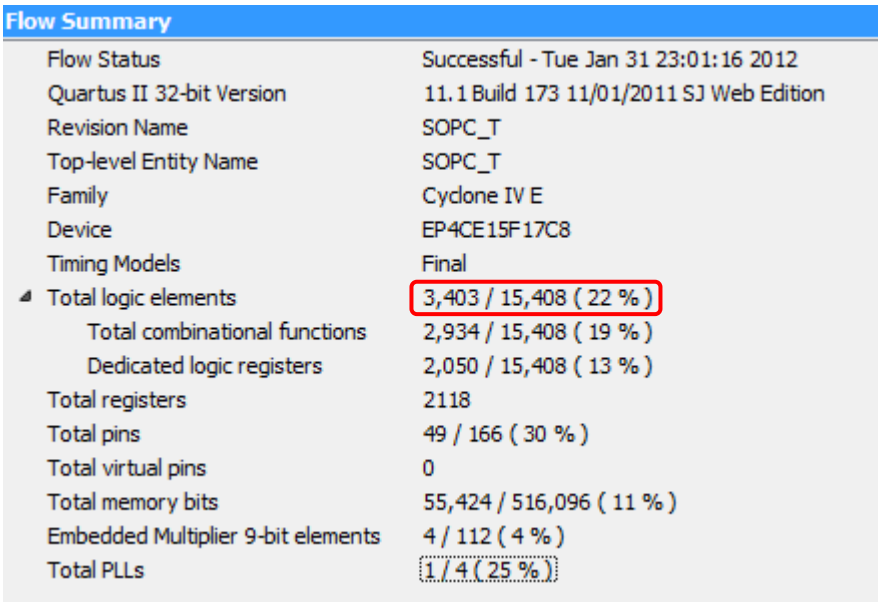


次は保存してからコンパイルしましょう。



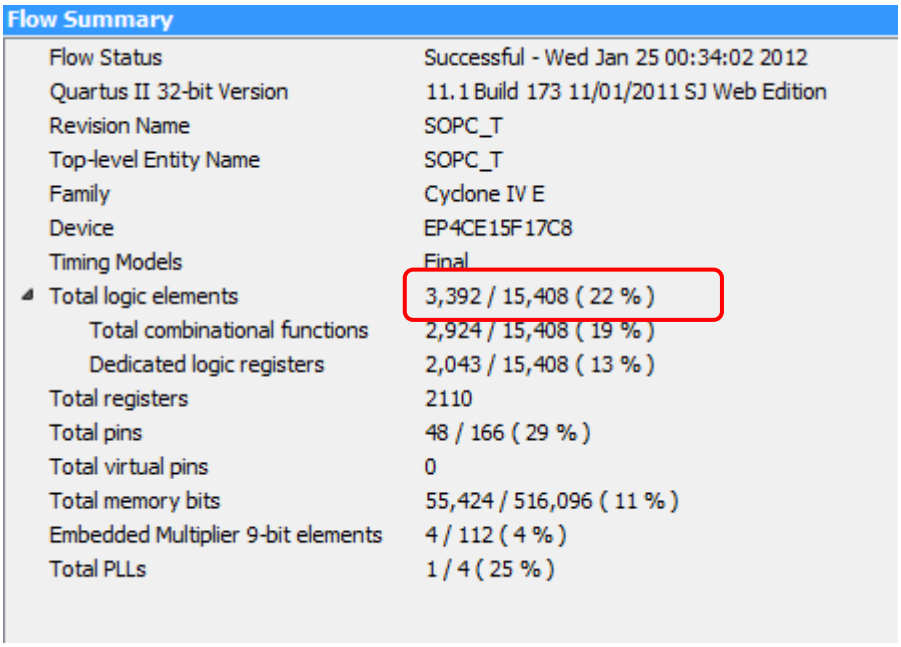
下記のコンパイル完了画面が出来ます。





前章のリソース：(下図)

比較すれば、リソースは KEY PIO を追加前と殆ど変りません。(一つ PIO モジュールが使わるリソースが少ない)



Flow Summary	
Flow Status	Successful - Wed Jan 25 00:34:02 2012
Quartus II 32-bit Version	11.1 Build 173 11/01/2011 SJ Web Edition
Revision Name	SOPC_T
Top-level Entity Name	SOPC_T
Family	Cyclone IV E
Device	EP4CE15F17C8
Timing Models	Final
Total logic elements	3,392 / 15,408 (22 %)
Total combinational functions	2,924 / 15,408 (19 %)
Dedicated logic registers	2,043 / 15,408 (13 %)
Total registers	2110
Total pins	48 / 166 (29 %)
Total virtual pins	0
Total memory bits	55,424 / 516,096 (11 %)
Embedded Multiplier 9-bit elements	4 / 112 (4 %)
Total PLLs	1 / 4 (25 %)

最後、AS モードあるいは JTAG モードで書き込んでください。

7.2.3 ソフトウエア開発

前章の続き、LED 用のプロジェクトを NIOS II IDE で開いてコンパイルしましょう。（ショットキー：Ctrl+b）

コンパイル後の system.h ファイル内容を見てみましょう。

KEY に関する内容が前章により多くなります。下の赤□の内容は KEY のベースアドレスと割り込み番号です。

```
/*
 * KEY configuration
 *
 */

#define KEY_NAME "/dev/KEY"
#define KEY_TYPE "altera_avalon_pio"
#define KEY_BASE 0x00000810
#define KEY_SPAN 16
#define KEY_IRQ 2
#define KEY_IRQ_INTERRUPT_CONTROLLER_ID 0
#define KEY_DO_TEST_BENCH_WIRING 0
#define KEY_DRIVEN_SIM_VALUE 0
#define KEY_HAS_TRI 0
#define KEY_HAS_OUT 0
#define KEY_HAS_IN 1
#define KEY_CAPTURE 0
#define KEY_DATA_WIDTH 1
#define KEY_RESET_VALUE 0
#define KEY_EDGE_TYPE "NONE"
#define KEY_IRQ_TYPE "LEVEL"
#define KEY_BIT_CLEARING_EDGE_REGISTER 0
#define KEY_BIT_MODIFYING_OUTPUT_REGISTER 0
#define KEY_FREQ 100000000
#define ALT_MODULE_CLASS_KEY altera_avalon_pio
```

１．ヘッダファイル編集

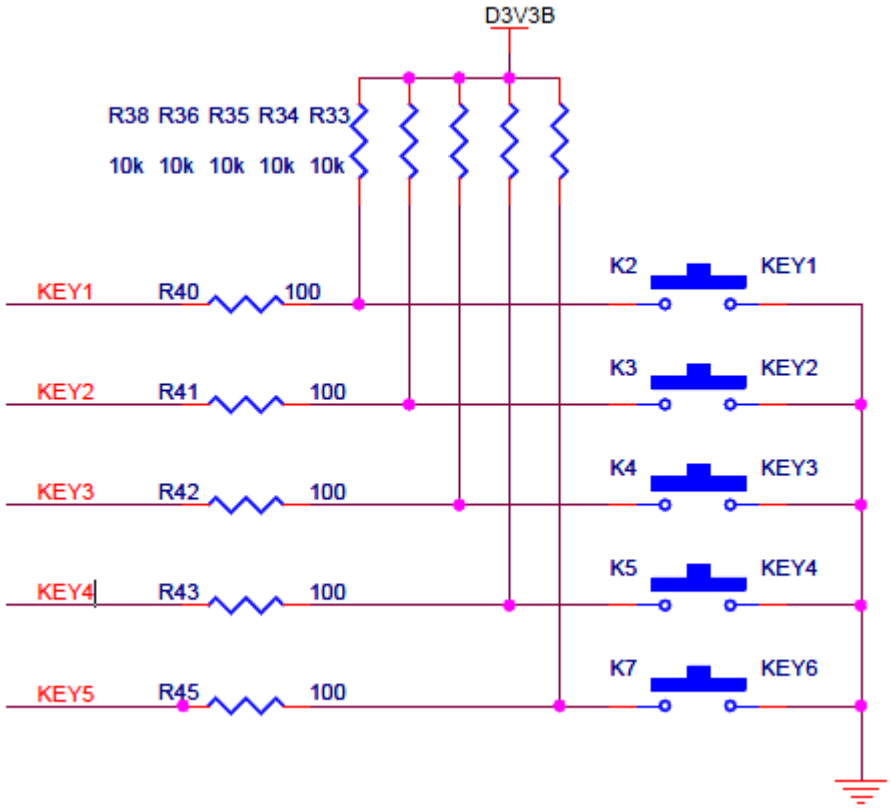
sopc.h というヘッダファイルに下記内容を追加してください。

内容は LED と似ています。

```
#define _LED
#define _KEY

/*------------------------------------------------------------------------------
 *  Peripheral registers structures
 *----------------------------------------------------------------------------*/
typedef struct
{
    unsigned long int DATA;
    unsigned long int DIRECTION;
    unsigned long int INTERRUPT_MASK;
    unsigned long int EDGE_CAPTURE;

}PIO_STR;

/*------------------------------------------------------------------------------
 *  Peripheral declaration
 *----------------------------------------------------------------------------*/
#ifdef _LED
#define LED          ((PIO_STR *) PIO_LED_BASE)
#endif /*_LED*/

#ifdef _KEY
#define KEY          ((PIO_STR *) KEY_BASE)
#endif /*_KEY*/

#endif /*SOPC_H_*/
```

２．C ファイル修正

ボタンに関連の回路図：



main.c プログラムは下記通りです。（弊社 NIOS II サンプルからコピーしても OK）
このプログラムではボタンの安定処理がありあせん、割り込み処理のサンプルだけです。実際の運用でボタンの安定処理を追加必要です、ここ説明しません。main.c は外部ボタンにより割り込み処理を行います、上記回路図によりボタンを押すと、低レベルになります、そして、当初 SOPC Builder の設定により低レベルで割り込みを起こすため、割り込み関数処理に入ります。割り込み関数の中、key_flag を論理演算で否定にします。main 関数の中、続けて検知して key_flag が 1 になると、LED->DATA を 1 に設定し LED を点灯します。逆に、key_flag が 0 になると、LED->DATA を 0 に設定し LED を消灯します。

```
#include "../inc/sopc.h"
#include "system.h"
#include "sys/alt_irq.h"
#include <unistd.h>
#include <stdio.h>
int key_flag = 0;
void ISR_key(void * context,unsigned long id)
{
    key_flag = ~key_flag;
}
int init_key(void)
{
    KEY->INTERRUPT_MASK = 1;
    KEY->EDGE_CAPTURE = 0;

    return  alt_irq_register(KEY_IRQ,NULL,ISR_key);
}

int main()
{
    if(!init_key()){
        printf("register successfully!\n");
    }
    else{
        printf("Error: register failure!\n");
    }

    while(1){
        if(key_flag){
            LED->DATA = 1;
        }
        else{
            LED->DATA =0;
        }
    }

    return 0;
}
```

プログラム説明：

■初期化関数：

①：割り込みを有効にする

KEY->INTERRUPT_MASK は割り込みコントローラレジスタのメモリマッピングです、1 を設定される場合、割り込みを有効になります、0 を設定される場合、割り込みを無効になります。

②：この前、見た事があるでしょう。この関数は割り込みのレジスタを実現します。KEY_IRQ は system.h で定義されます、ISR_key は ISR 関数です、戻し値により成功にレジスタかどうかを判断します。（0：成功；0 以外：失敗）

```
int init_key(void)
{
    KEY->INTERRUPT_MASK = 1;    ①
    KEY->EDGE_CAPTURE = 0;

    return  alt_irq_register(KEY_IRQ,NULL,ISR_key);    ②
}
```

■ISR_key 関数：

割り込み度に key_flag を論理演算で否定します。

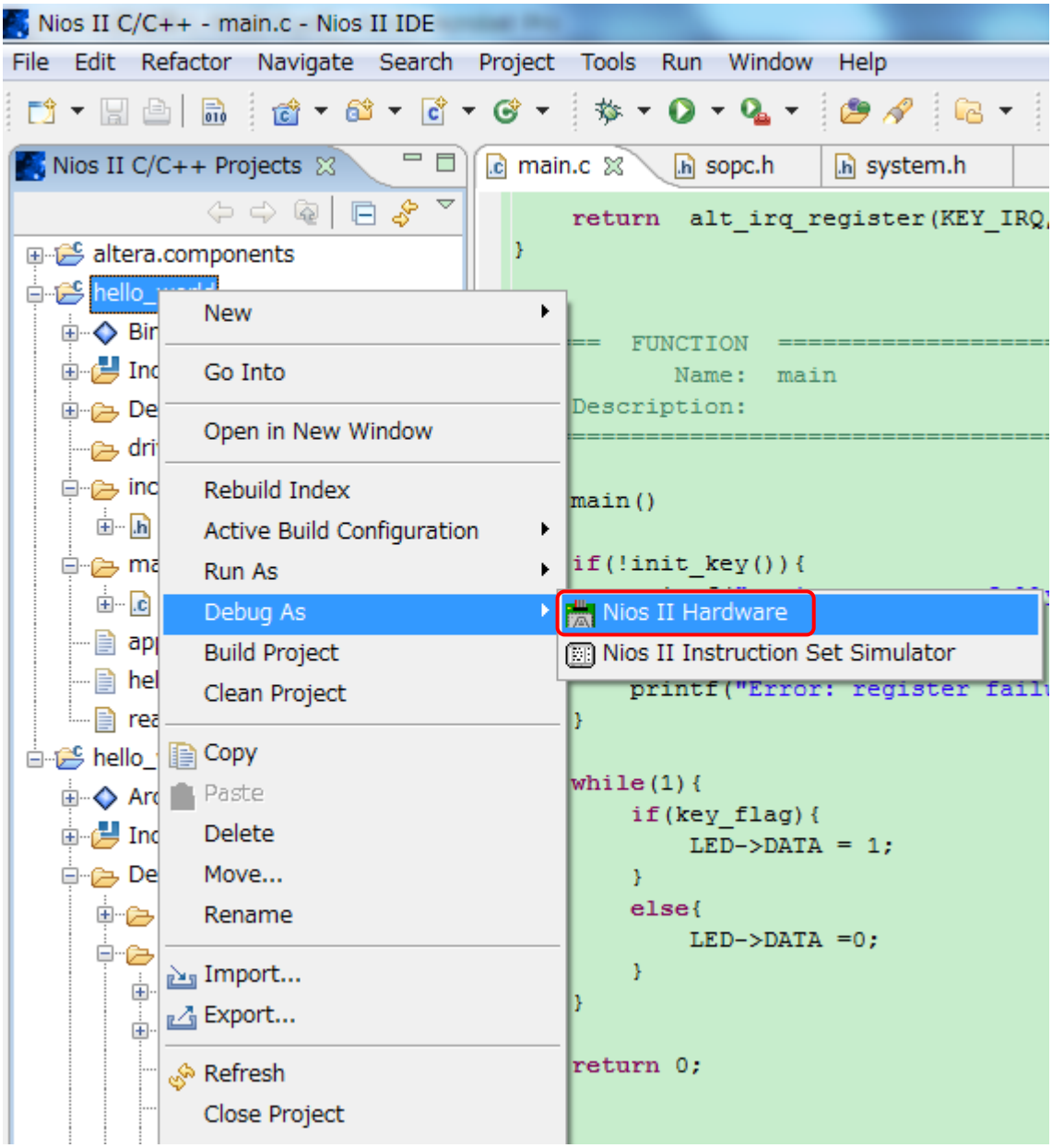
```
void ISR_key(void * context,unsigned long id)
{
    key_flag = ~key_flag;
}
```

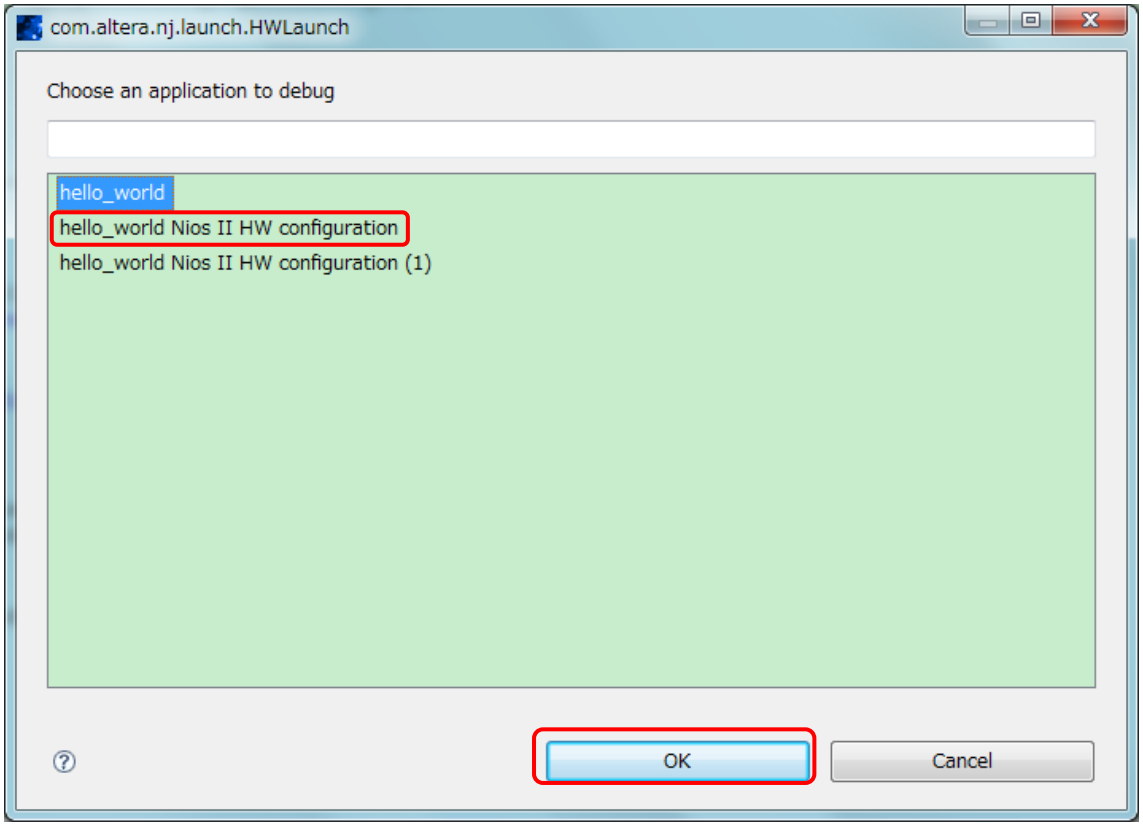
■main 関数：赤 1 は割り込みレジスタが成功かどうかを判断します、そして結果メッセージも表示します。赤 2 は割り込みにより点灯、消灯処理を行います。

```
int main()
{
    if(!init_key()){                    1
        printf("register successfully!\n");
    }
    else{
        printf("Error: register failure!\n");
    }

    while(1){
        if(key_flag){
            LED->DATA = 1;
        }                               2
        else{
            LED->DATA =0;
        }
    }

    return 0;
}
```

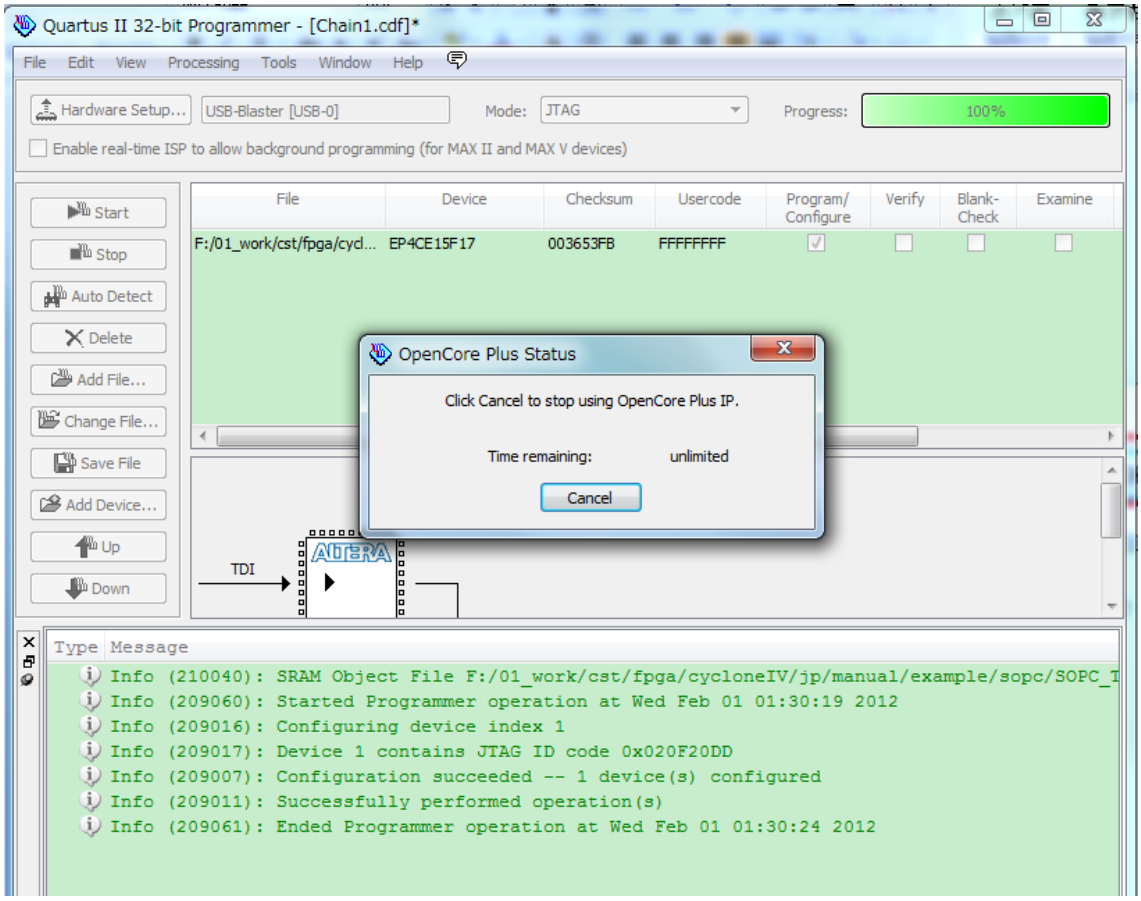
３．NIOS II IDE のオンラインデバッグ

USB ダウンロードケーブルを開発ボードの JTAG インタフェースと接続し、電源を入れてから、まず、IP マクロをダウンロードしてください。(ダウンロード方法はダウンロード節を参照ください。)プロジェクト「hello_world」を右クリック、「Debug As」→「NIOS II Hardware」を選択

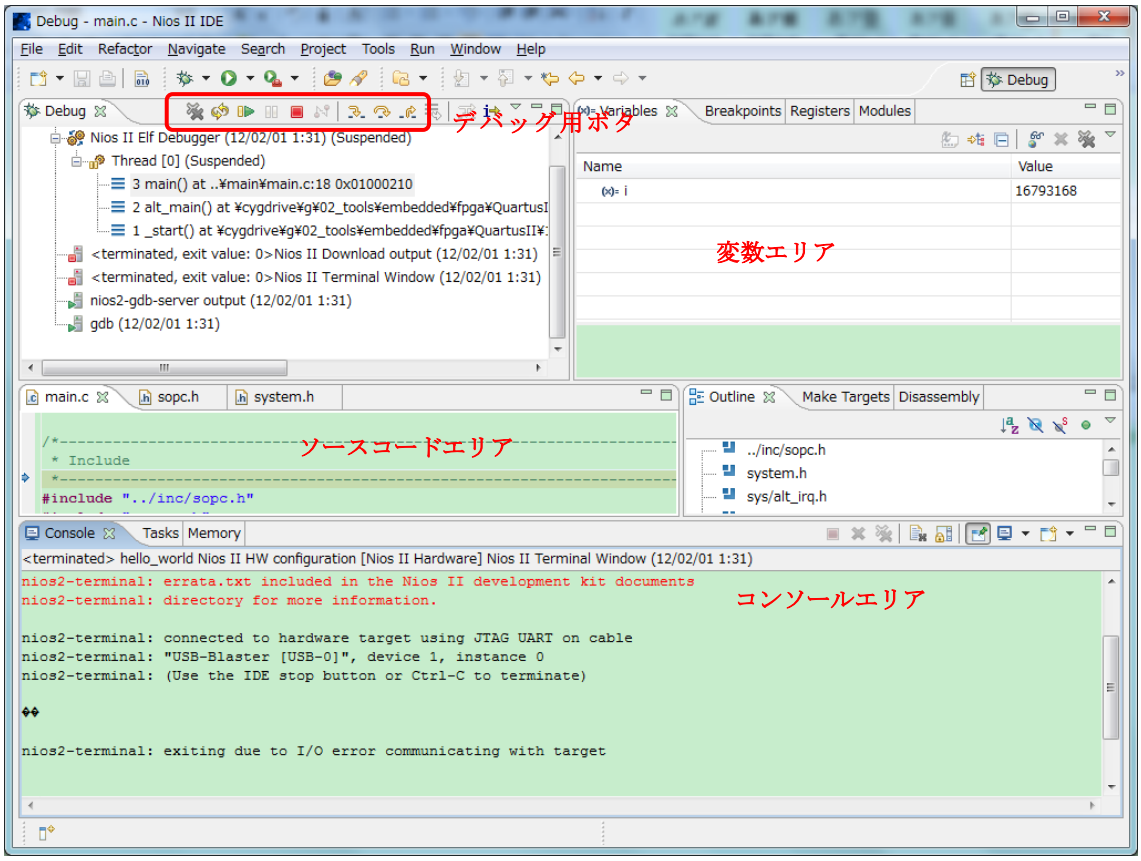




ダウンロードしていない場合、自動にダウンロード画面が出ます、詳しくはダウンロード節を参照ください。



ダウンロード済みの場合、デバッグ画面が出来ます。



デバッグのノウハウ：

■ブレークポイント設定

設定したいところにダブルクリックすれば、設定できます。

```
void ISR_key(void * context,unsigned long id)
{
    key_flag = ~key_flag;
}
```

デバッグ用ボタンの「Resume」を押してプログラムを実行

開発ボードのキーを押す（OK ボタン：真ん中のボタン）

上記割り込み関数の所に止まったら、割り込み処理に入って問題がないと言う事です。

この方法は割り込みのデバッグでよく使われる手法です。

7.3 シリアルポート

本章は NIOS II でシリアルポートをどのように操作するか、シリアルポート割り込みで受信等の内容を説明します。

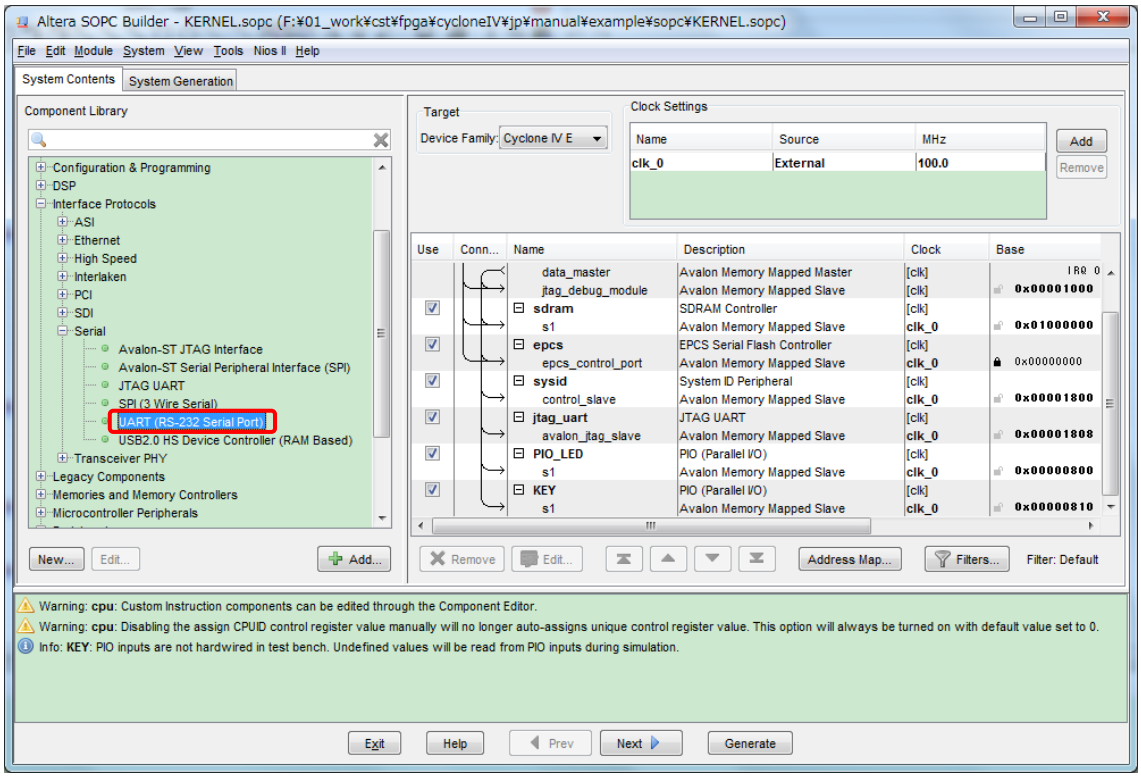
7.3.1 概要

本節はこの前の実例より複雑ですが、詳しく説明します。RS232 機能を実現する説明だけではなく、作成後のプログラムをメンテナンスしやすく、移植しやすくにするため、厳密な、専門的なプログラムをどのように作るかというプログラミングのアイディアも説明します。

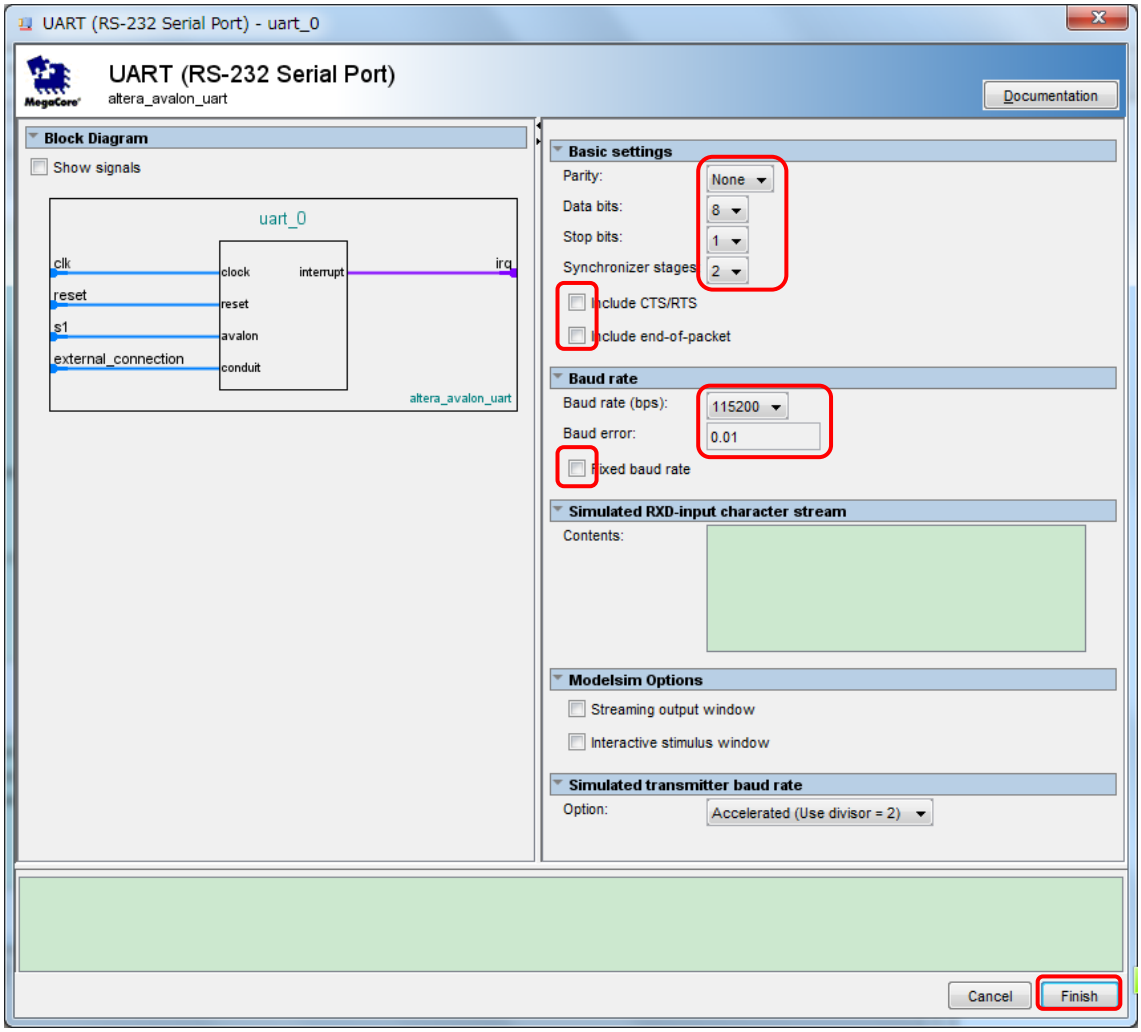
7.3.2 ハードウエア開発

前の実例と同じで RS232 モジュールを追加しましょう。Quartus II を立ち上げ、SOPC Builder を起動させ（メニュー「Tools」→「SOPC Builder」）

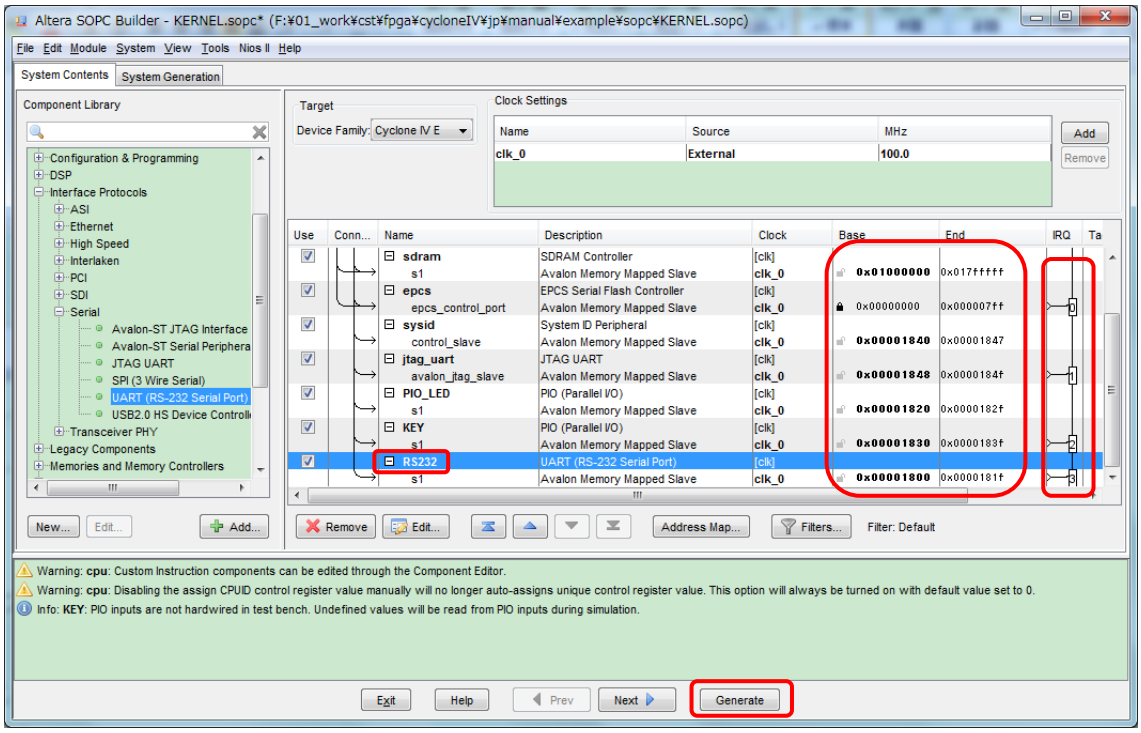
SOPC Builder 画面で「Interface Protocols」→「Serial」→「UART(RS-232 Serial Port)」をダブルクリック



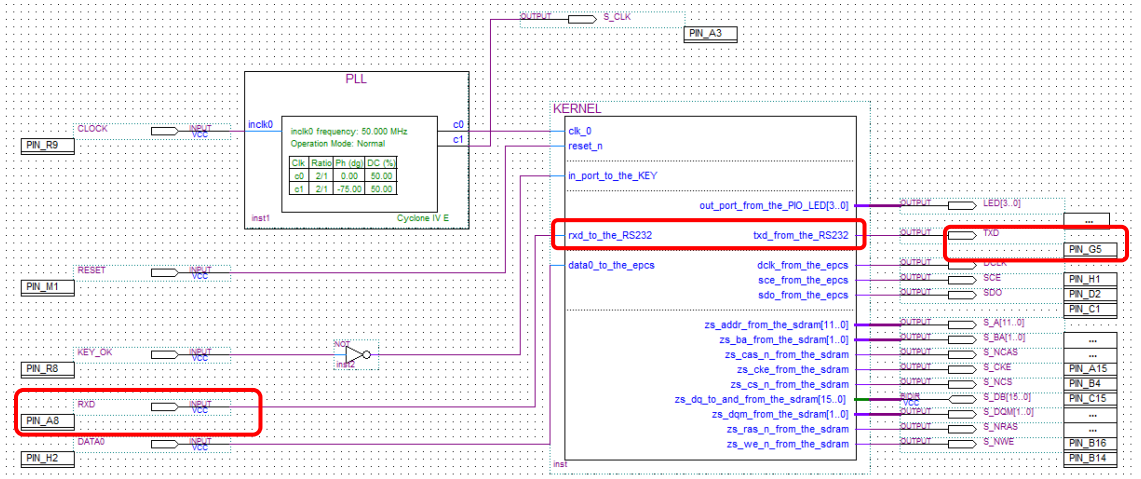
下図のように設定してから「Finish」ボタンを押す



RS232 モジュールを構築後、名前を「RS232」に変更、アドレス、IRQ を自動割り当てにします。全て変更が終わってから、「Generate」ボタンを押しましょう。（保存かを聞かれるダイアログが出る時、「Save」を押す）



暫く待って正常にコンパイル完了後、「Exit」をクリック、Quartus II 画面に戻します。それから前章の実例と同じように KERNEL を更新し、ピンの名前を修正し、修正後、TCL ファイルを実行してからピンが割り当てられます。（詳しい操作方法は前章を参照）



ピンを割り当てにしたら、保存・コンパイルしてから FPGA に AS か JTAG かでダウンロードしてください。

7.3.3 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、RS232 に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
/*
 * RS232 configuration
 *
 */

#define RS232_NAME "/dev/RS232"
#define RS232_TYPE "altera_avalon_uart"
#define RS232_BASE 0x00001800
#define RS232_SPAN 32
#define RS232_IRQ 3
#define RS232_IRQ_INTERRUPT_CONTROLLER_ID 0
#define RS232_BAUD 115200
#define RS232_DATA_BITS 8
#define RS232_FIXED_BAUD 0
#define RS232_PARITY 'N'
#define RS232_STOP_BITS 1
#define RS232_SYNC_REG_DEPTH 2
#define RS232_USE_CTS_RTS 0
#define RS232_USE_EOP_REGISTER 0
#define RS232_SIM_TRUE_BAUD 0
#define RS232_SIM_CHAR_STREAM ""
#define RS232_RELATIVEPATH 1
#define RS232_FREQ 100000000
#define ALT_MODULE_CLASS_RS232 altera_avalon_uart
```

次はプログラムを修正していきましょう。

１．ヘッダファイル修正

①sopc.h ファイルは下記のように修正します。

```
/*-------------------------------------------------------------------------↓
 *  Define↓
 *-------------------------------------------------------------------------*/↓
#define _LED↓
#define _KEY↓
#define _UART↓
```

```
/*--------------------------UART-----------------------------------------*/
typedef struct
{
    union{
        struct{
            volatile unsigned long int RECEIVE_DATA         :8;
            volatile unsigned long int NC                   :24;
        }BITS;
        volatile unsigned long int WORD;
    }RXDATA;

    union{
        struct{
            volatile unsigned long int TRANSMIT_DATA        :8;
            volatile unsigned long int NC                   :24;
        }BITS;
        volatile unsigned long int WORD;
    }TXDATA;

    union{
        struct{
            volatile unsigned long int PE           :1;
            volatile unsigned long int FE           :1;
            volatile unsigned long int BRK          :1;
            volatile unsigned long int ROE          :1;
            volatile unsigned long int TOE          :1;
            volatile unsigned long int TMT          :1;
            volatile unsigned long int TRDY         :1;
            volatile unsigned long int RRDY         :1;
            volatile unsigned long int E            :1;
            volatile unsigned long int NC           :1;
            volatile unsigned long int DCTS         :1;
            volatile unsigned long int CTS          :1;
```

受信レジスタ

送信レジスタ

ステータス
レジスタ

ステータス
レジスタ

```
            volatile unsigned long int EOP          :1;
            volatile unsigned long int NC1          :19;
        } BITS;
        volatile unsigned long int WORD;
    }STATUS;

    union{
        struct{
            volatile unsigned long int IPE          :1;
            volatile unsigned long int IFE          :1;
            volatile unsigned long int IBRK         :1;
            volatile unsigned long int IROE         :1;
            volatile unsigned long int ITOE         :1;
            volatile unsigned long int ITMT         :1;
            volatile unsigned long int ITRDY        :1;
            volatile unsigned long int IRRDY        :1;
            volatile unsigned long int IE           :1;
            volatile unsigned long int TRBK         :1;
            volatile unsigned long int IDCTS        :1;
            volatile unsigned long int RTS          :1;
            volatile unsigned long int IEOP         :1;
            volatile unsigned long int NC           :19;
        }BITS;
        volatile unsigned long int WORD;
    }CONTROL;
```

コントローラ
レジスタ

```
    union{
        struct{
            volatile unsigned long int BAUD_RATE_DIVISOR    :16;
            volatile unsigned long int NC                   :16;
        }BITS;
        volatile unsigned  int WORD;
    }DIVISOR;

}UART_ST;
```

ボーレート分周器

この構造体は５つ共用体を含みます、この５つ共用体は RS232 の５つレジスタに対応します。レジスタには下記資料（P78）を見てましょう。

《Embedded Peripherals IP User Guide》:

http://www.altera.com/literature/ug/ug_embedded_ip.pdf

Table 7–4. UART Core Register Map

Offset	Register Name	R/W	Description/Register Bits													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	Reserved					*(1)*	*(1)*	Receive Data						
1	txdata	WO	Reserved					*(1)*	*(1)*	Transmit Data						
2	status *(2)*	RW	Reserved	eop	cts	dcts	(1)	e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	Reserved	ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor *(3)*	RW	Baud Rate Divisor													
5	endof-packet *(3)*	RW	Reserved					*(1)*	*(1)*	End-of-Packet Value						

Notes to Table 7–4:

(1) These bits may or may not exist, depending on the **Data Width** hardware option. If they do not exist, they read zero, and writing has no effect.
(2) Writing zero to the `status` register clears the `dcts`, `e`, `toe`, `roe`, `brk`, `fe`, and `pe` bits.
(3) This register may or may not exist, depending on hardware configuration options. If it does not exist, reading returns an undefined value and writing has no effect.

上図の(1)の述べたように、第 7,8bit は設定されるデータにより変わります、ここ 8bit を設定していますので、7,8bit は 1-6bit と同じです。

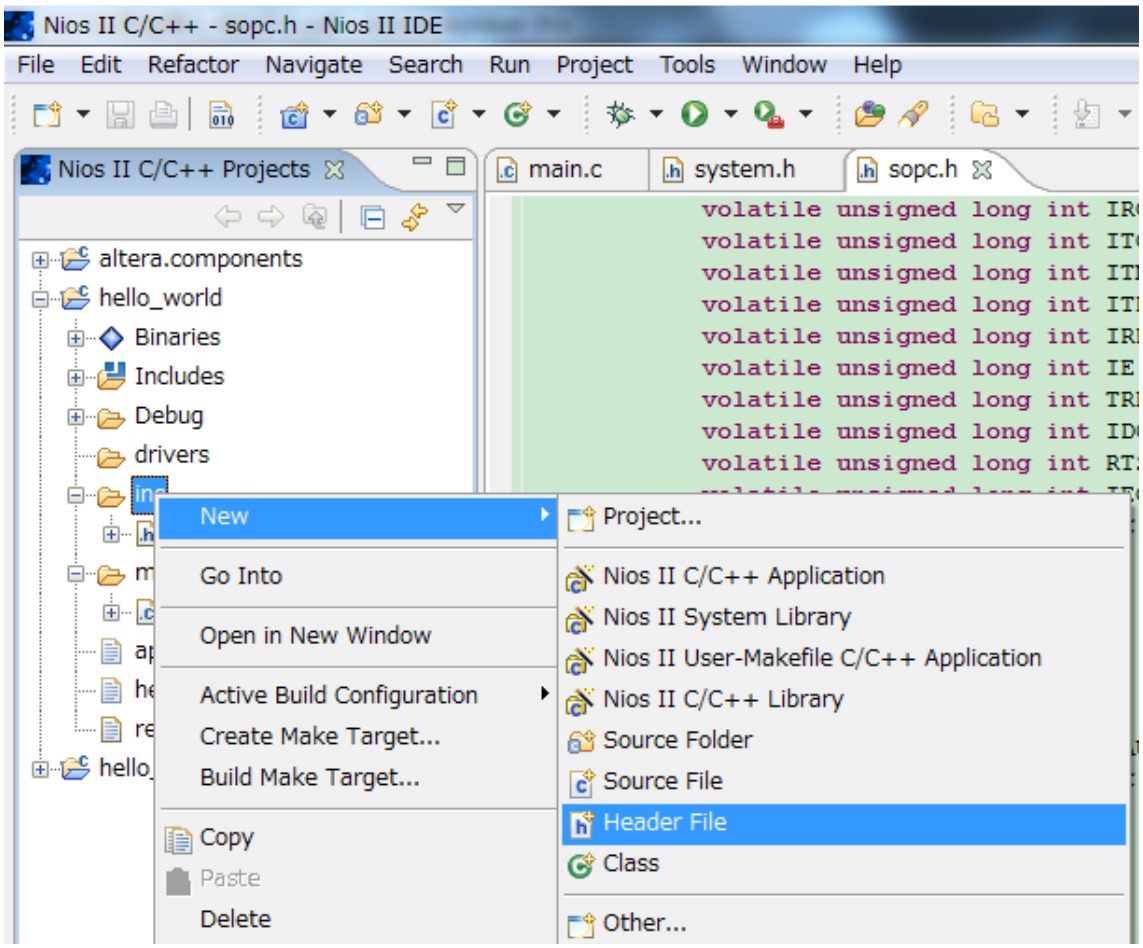
前章に説明した PIO と似ています、本章で宣言した構造体もレジスタ順番により定義されます（endofpacket を使われないので、定義しません）、このやり方の理由はオフセットで操作できることです。レジスタを bit ごとにコントロールできるため、構造体に共用体を組み込まれます、勿論、全体からレジスタもコントロールできます。

最後、RS232 用のマイクロを定義します。

```
#ifdef _UART
#define UART              ((UART_ST *) RS232_BASE)
#endif /*_UART */
```

②uart.h 新規作成

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック



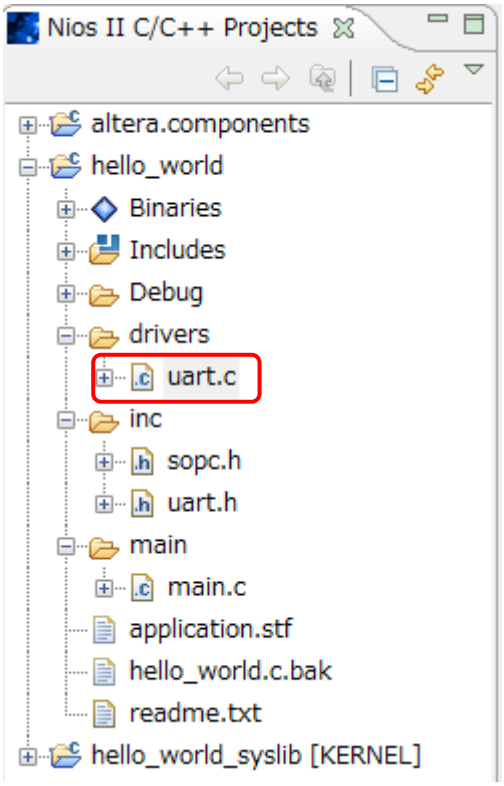
uart.h 内容：

ソースの構造体「UART_T」はオブジェクト指向プログラミングのアイディアを運用して UART に関する関数、変数を全てパッケージされます、他の函数から見ると、uart という構造体しか見えません。

```
#ifndef UART_H_
#define UART_H_
/*-------------------------------------------------------------------------------
 *  Include
 *-------------------------------------------------------------------------------*/
#include "../inc/sopc.h"

/*-------------------------------------------------------------------------------
 *  Define
 *-------------------------------------------------------------------------------*/
#define BUFFER_SIZE 200

/*-------------------------------------------------------------------------------
 *  Struct
 *-------------------------------------------------------------------------------*/
typedef struct{
    unsigned int receive_flag;
    unsigned int receive_count;
    unsigned char receive_buffer[BUFFER_SIZE];
    int (* send_byte)(unsigned char data);
    void (* send_string)(unsigned int len, unsigned char *str);
    int (* init)(void);
    unsigned int (* baudrate)(unsigned int baudrate);
}UART_T;

extern UART_T uart;

#endif /*UART_H_*/
```

２．ドライバーファイル作成

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「uart.c」を作成します。


Nios II C/C++ Projects
altera.components
hello_world
Binaries
Includes
Debug
drivers
uart.c
inc
sopc.h
uart.h
main
main.c
application.stf
hello_world.c.bak
readme.txt
hello_world_syslib [KERNEL]

uart.c 内容：

```
/*
 * ===================================================================================
 *
 *       Filename:  uart.c
 *
 *    Description:
 *
 *        Version:  1.0.0
 *        Created:  2012.2.3
 *       Revision:  none
 *       Compiler:  Nios II 11.1 IDE
 *
 * ===================================================================================
 */

/*-----------------------------------------------------------------------------------
 *  Include
 *-----------------------------------------------------------------------------------*/
#include "sys/alt_irq.h"
#include <stdlib.h>
#include <stdio.h>
#include "../inc/uart.h"
#include "../inc/sopc.h"

/*-----------------------------------------------------------------------------------
 *  Function Prototype
 *-----------------------------------------------------------------------------------*/
static int uart_send_byte(unsigned char data);
static void uart_send_string(unsigned int len, unsigned char *str);
static int uart_init(void);
static void uart_ISR(void);
static int set_baudrate(unsigned int baudrate);
```

```
/*-----------------------------------------------------------------------------------
 *  Struct initialize
 *-----------------------------------------------------------------------------------*/
UART_T uart={
    .receive_flag=0,
    .receive_count=0,
    .send_byte=uart_send_byte,
    .send_string=uart_send_string,
    .init=uart_init,
    .baudrate=set_baudrate
    };

/*
 * ===  FUNCTION  ====================================================================
 *         Name:  uart_send_byte
 *  Description:
 * ===================================================================================
 */
int uart_send_byte(unsigned char data)
{
    UART->TXDATA.BITS.TRANSMIT_DATA = data;
    while(!UART->STATUS.BITS.TRDY);

    return 0;
}
```

```
/*
 * ===  FUNCTION  ======================================================================
 *         Name:  uart_send_string
 *  Description:
 * =====================================================================================
 */
void uart_send_string(unsigned int len, unsigned char *str)
{
    while(len--)
    {
        uart_send_byte(*str++);
    }
}

/*
 * ===  FUNCTION  ======================================================================
 *         Name:  uart_init
 *  Description:
 * =====================================================================================
 */
int uart_init(void)
{
    set_baudrate(115200);

    UART->CONTROL.BITS.IRRDY=1;
    UART->STATUS.WORD=0;

    alt_irq_register(RS232_IRQ, NULL, uart_ISR);

    return 0;
}
```

```
/*
 * ===  FUNCTION  ======================================================================
 *         Name:  uart_ISR
 *  Description:
 * =====================================================================================
 */
static void uart_ISR(void)
{
    while(!(UART->STATUS.BITS.RRDY));

    uart.receive_buffer[uart.receive_count++] = UART->RXDATA.BITS.RECEIVE_DATA;

    if(uart.receive_buffer[uart.receive_count-1]=='\n'){
        uart.receive_buffer[uart.receive_count]='\0';
//      uart_send_string(uart.receive_count,uart.receive_buffer);
        uart.receive_count=0;
        uart.receive_flag=1;
    }


}

/*
 * ===  FUNCTION  ======================================================================
 *         Name:  set_baudrate
 *  Description:
 * =====================================================================================
 */
int set_baudrate(unsigned int baudrate)
{
    UART->DIVISOR.WORD=(unsigned int)(ALT_CPU_FREQ/baudrate+0.5);

    return 0;
}
```

３．メイン関数修正

下記のように修正、修正後、全てファイルを保存しコンパイルしましょう。

コンパイル後、実際のシリアルポートと接続し、プログラムをダウンロードしたら、ハイパーターミナルで結果を見えます。

最初「Hello FPGA」が表示されます、キーボードで何にか入力すれば、ハイパーターミナル同じ内容も表示されます。

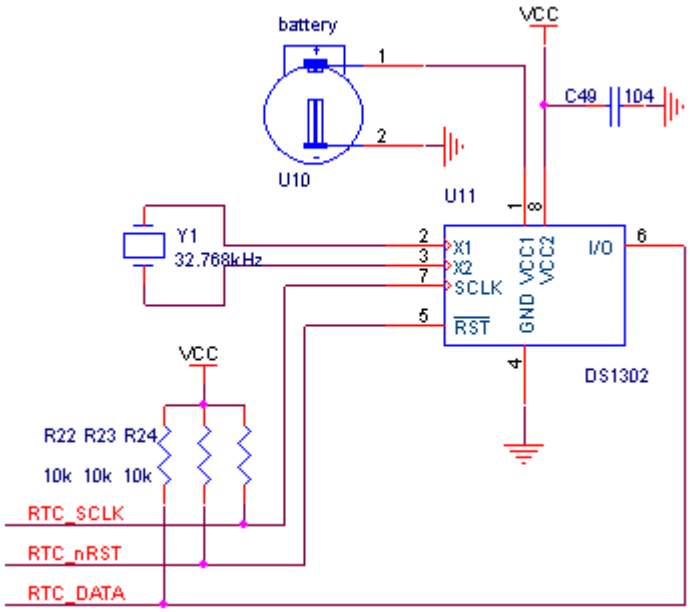
```
/*---------------------------------------------------------------------------
 * Include
 *-------------------------------------------------------------------------*/
#include "system.h"
#include "sys/alt_irq.h"
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include "../inc/uart.h"
#include "../inc/sopc.h"

/*
 * ===  FUNCTION  ======================================================================
 *         Name:  main
 *  Description:
 * =====================================================================================
 */
int main(void)
{
    unsigned char buffer[50]="Hello FPGA!\n";

    uart.init();

    while(1){
        if(uart.receive_flag){
            memset(buffer,0,50);// clear buffer

            strcpy(buffer,uart.receive_buffer);//copy uart.receive_buffer to buffer

            uart.receive_flag = 0;//clear flags
        }
        uart.send_string(sizeof(buffer),buffer);

        usleep(500000);
    }

    return 0;
}
```

7.4 RTC

本節から NIOS II でリアルタイムクロックチップ DS1302 をどのように実現するかを説明します。

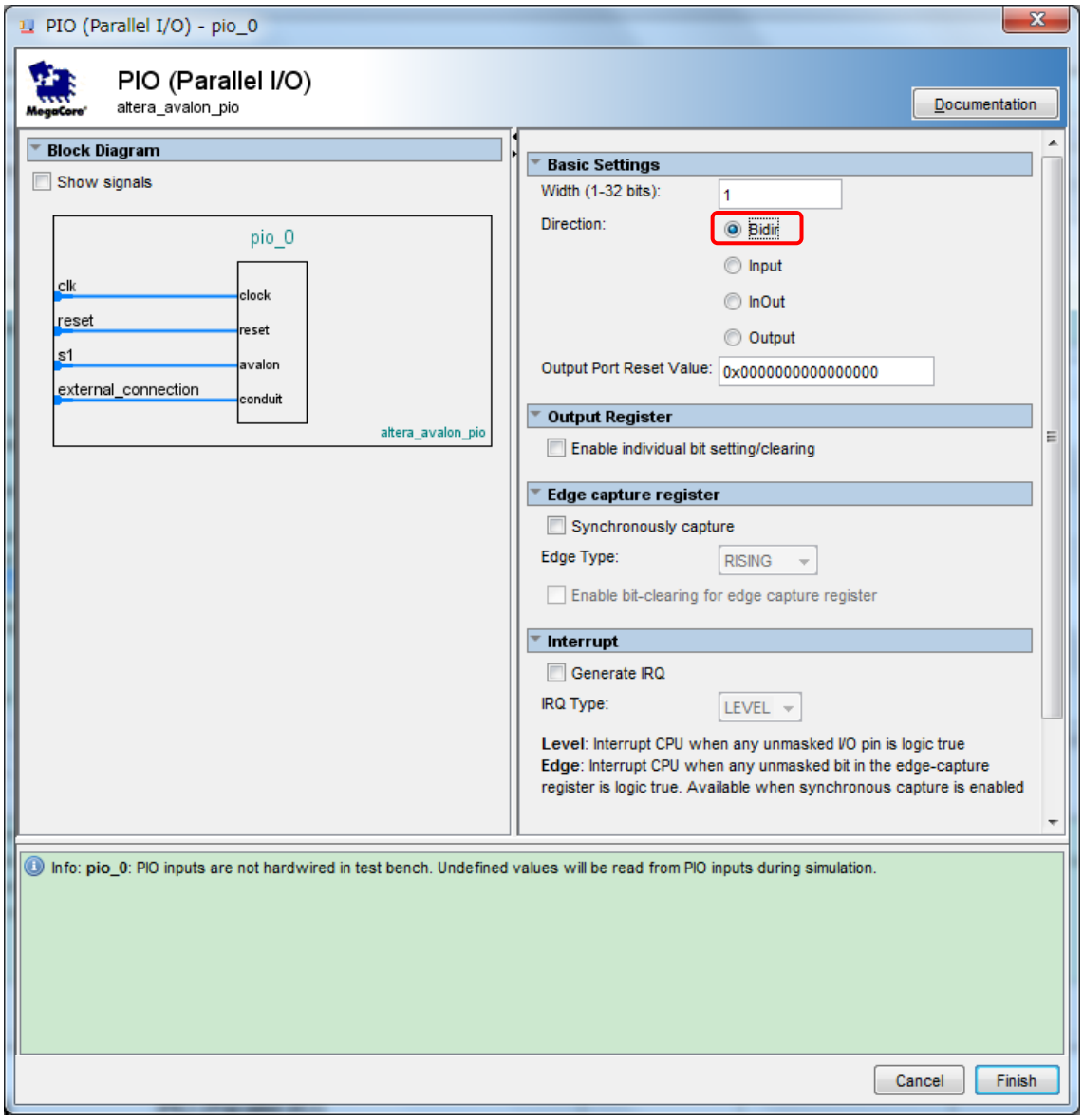
7.4.1 概要

DS1302 は DALLAS 社の充電リアルタイムクリックチップです、リアル時計/カレンダーを含み、31byte の静的 RAM もあります。使う時に、3 ピンのみ：RES（リセット）、I/O(データバス)、SCLK（シリアル時計）。時計/RAM が 1byte または 31byte 文字セットでデータを読む/書きます、動作時消費電力が低く、データと時計情報を保持する際、1mw より低いです。詳しくはデータシートを参照してください。

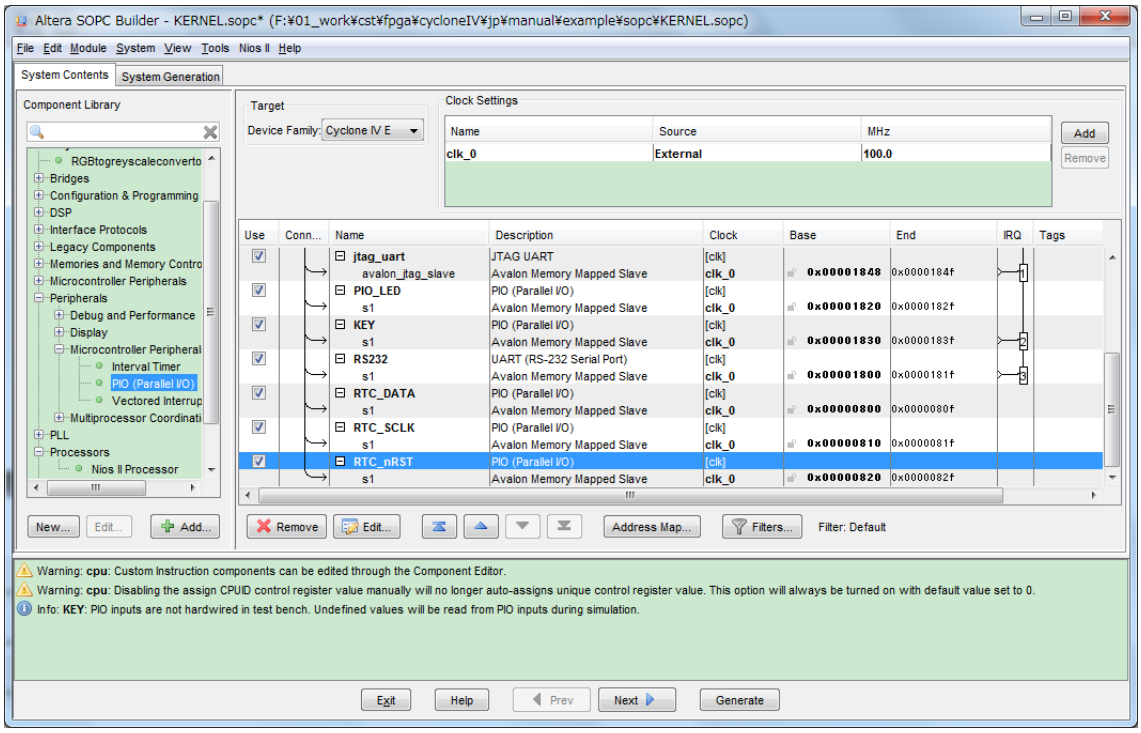


7.4.2 ハードウエア開発

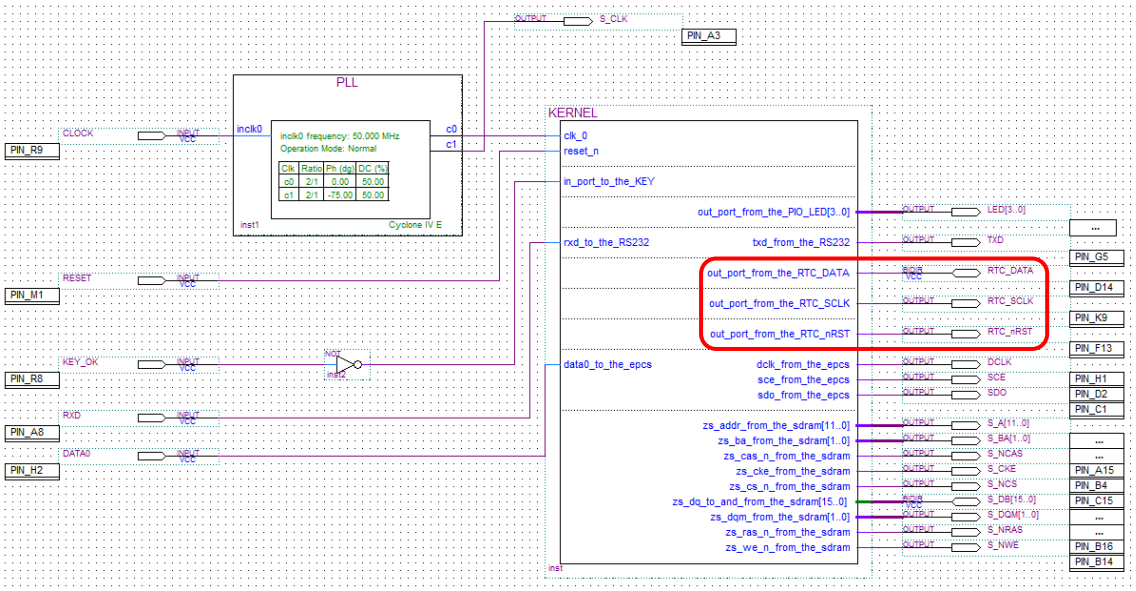
まず、三つ PIO モジュール（RTC_DATA、RTC_SCLK、RTC_nRST）を作成必要です、PIO モジュール作成方法は１． PIO モジュールを IP コアに追加を参照してください。
特に注意必要の所は RTC_DATA を作成する時、データバスとして入力と出力があります、双方向の IO を設定必要です。他の二つ PIO モジュールは出力を設定します。


PIO (Parallel I/O) - pio_0
PIO (Parallel I/O)
altera_avalon_pio
Documentation
Block Diagram
Show signals
pio_0
clk
reset
s1
external_connection
clock
reset
avalon
conduit
altera_avalon_pio
Basic Settings
Width (1-32 bits):
1
Direction:
Bidir
Input
InOut
Output
Output Port Reset Value:
0x0000000000000000
Output Register
Enable individual bit setting/clearing
Edge capture register
Synchronously capture
Edge Type:
RISING
Enable bit-clearing for edge capture register
Interrupt
Generate IRQ
IRQ Type:
LEVEL
Level: Interrupt CPU when any unmasked I/O pin is logic true
Edge: Interrupt CPU when any unmasked bit in the edge-capture register is logic true. Available when synchronous capture is enabled
Info: pio_0: PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
Cancel
Finish

作成後の様子：



ベースアドレス、割り込み番号を自動割り当てにします、その後、コンパイルします。
コンパイル完了後、Quartus II 画面に KERNEL を更新、ピンを調整、ピンの名前を修正、入力、出力も付けます。最後、TCL ファイルを実行してからコンパイルします。
※ピン RTC_DATA が双方向に設定してください。



7.4.3 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、RTC に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
/*
 * RTC_DATA configuration
 *
 */

#define RTC_DATA_NAME "/dev/RTC_DATA"
#define RTC_DATA_TYPE "altera_avalon_pio"
#define RTC_DATA_BASE 0x00001840

......

/*
 * RTC_SCLK configuration
 *
 */

#define RTC_SCLK_NAME "/dev/RTC_SCLK"
#define RTC_SCLK_TYPE "altera_avalon_pio"
#define RTC_SCLK_BASE 0x00001850

......

/*
 * RTC_nRST configuration
 *
 */

#define RTC_NRST_NAME "/dev/RTC_nRST"
#define RTC_NRST_TYPE "altera_avalon_pio"
#define RTC_NRST_BASE 0x00001860

......
```

これらのソースコードの中、以下のベースアドレスを使われます。

```
#define RTC_DATA_BASE 0x00001840
#define RTC_SCLK_BASE 0x00001850
#define RTC_NRST_BASE 0x00001860
```

次はプログラムを修正していきましょう。

１．ヘッダファイル修正

①sopc.h ファイルは下記のソースを追加します。

```
#define _RTC
#ifdef _RTC #define RTC_SCLK ((PIO_STR *) RTC_SCLK_BASE)
#define RTC_DATA ((PIO_STR *) RTC_DATA_BASE)
```

```
#define RTC_RST ((PIO_STR *) RTC_NRST_BASE) #endif /* _RTC */
```

②ds1302.h 新規作成

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック

```
/*
 *
=====================================================================
 *
 *       Filename:  ds1302.c
 *
 *    Description:
 *
 *        Version:  1.0.0
 *        Created:  2010.4.16
 *       Revision:  none
 *       Compiler:  Nios II 9.0 IDE
 *
 *
=====================================================================
 */

#ifndef DS1302_H_
#define DS1302_H_

/*---------------------------------------------------------------------
*  Include

*--------------------------------------------------------------------*/
#include "../inc/sopc.h"

/*---------------------------------------------------------------------
 *  Define
```

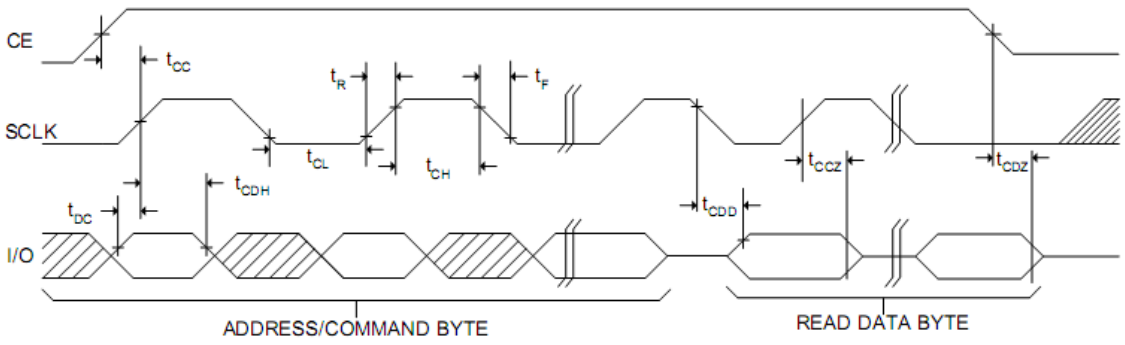
```
*-----------------------------------------------------------------------*/
#define RTC_DATA_OUT    RTC_DATA->DIRECTION = 1
#define RTC_DATA_IN     RTC_DATA->DIRECTION = 0

/*-----------------------------------------------------------------------
 *  Struct

*-----------------------------------------------------------------------*/
typedef struct{
   void (* set_time)(unsigned char *ti);
   void (* get_time)(char * ti);
}DS1302_STR;

/*-----------------------------------------------------------------------
 *  Extern Variable

*-----------------------------------------------------------------------*/
extern DS1302_STR ds1302;

#endif /*DS1302_H_*/
```

RS232 と同じです、中身には構造体があります、全て関数と変数をパッケージされます。

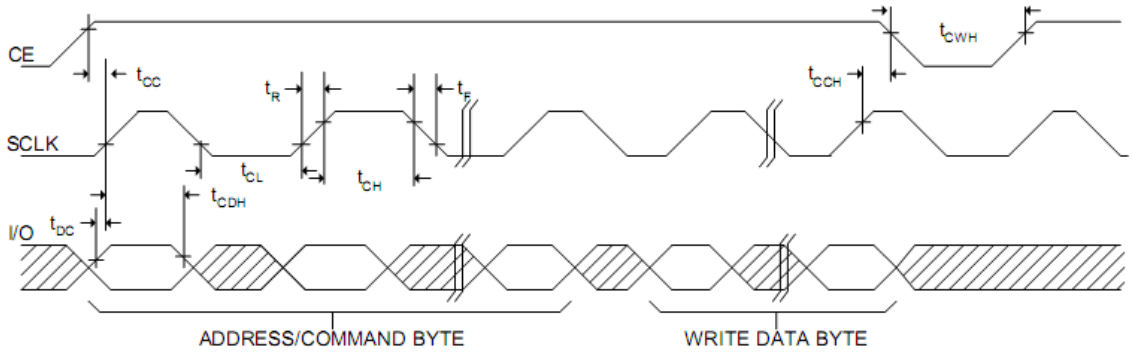
２．ドライバーファイル作成

チップのタイミング図により、ds1302 ドライバーを作成します。

読み込み時のタイミング図：



書込み時のタイミング図：



それ以外、レジスタは以下のテーブル通りです。左側の 2 列は読み込みと書込み用のアドレスとなります、毎回操作時、先にアドレスを書き込み、後はデータ転送です。

READ	WRITE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	RANGE
81h	80h	CH	10 Seconds			Seconds				00–59
83h	82h		10 Minutes			Minutes				00–59
85h	84h	12/$\overline{24}$	0	10 $\overline{\text{AM/PM}}$	Hour	Hour				1–12/0–23
87h	86h	0	0	10 Date		Date				1–31
89h	88h	0	0	0	10 Month	Month				1–12
8Bh	8Ah	0	0	0	0	0	Day			1–7
8Dh	8Ch	10 Year				Year				00–99
8Fh	8Eh	WP	0	0	0	0	0	0	0	—
91h	90h	TCS	TCS	TCS	TCS	DS	DS	RS	RS	—

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「ds1302.c」を作成します。

```
/*
 *
=====================================================================
 *
 *      Filename:  ds1302.c
 *
 *    Description:
 *
 *       Version:  1.0.0
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *
 *
```

```
================================================================================
 */

#include "../inc/ds1302.h"

static void delay(unsigned int dly);
static void write_1byte_to_ds1302(unsigned char da);
static unsigned char read_1byte_from_ds1302(void);
static void write_data_to_ds1302(unsigned char addr, unsigned char da);
static unsigned char read_data_from_ds1302(unsigned char addr);
void set_time(unsigned char *ti);
void get_time(char *ti);

DS1302_STR ds1302={
    .set_time = set_time,
    .get_time = get_time
};

/*
 * ===  FUNCTION
================================================================================
 *        Name:  delay
 *  Description:
 *
================================================================================
 */
void delay(unsigned int dly)
{
    for(;dly>0;dly--);
}

/*
 * ===  FUNCTION
================================================================================
 *        Name:  write_1byte_to_ds1302
 *  Description:  ds1302にデータを1 byte書き込む
```

```
 *
=====================================================================
 */
void write_1byte_to_ds1302(unsigned char da)
{
    unsigned int i;

    RTC_DATA_OUT;

    for(i=8; i>0; i--){
        if((da&0x01)!= 0)
            RTC_DATA->DATA = 1;
        else
            RTC_DATA->DATA = 0;

        delay(10);
        RTC_SCLK->DATA = 1;
        delay(20);
        RTC_SCLK->DATA = 0;
        delay(10);

        da >>= 1; //ASMのRRCと似ている
    }
}

/*
 * ===  FUNCTION
=====================================================================
 *        Name:  read_1byte_from_ds1302
 *  Description:  ds1302から1 byteデータを読み込む
 *
=====================================================================
 */
unsigned char read_1byte_from_ds1302(void)
{
    unsigned char i;
```

```
    unsigned char da = 0;

    RTC_DATA_IN;

    for(i=8; i>0; i--){
        delay(10);
        da >>= 1; //ASMのRRCと似ている
        if(RTC_DATA->DATA !=0 )
            da += 0x80;

        RTC_SCLK->DATA = 1;
        delay(20);
        RTC_SCLK->DATA = 0;
        delay(10);
    }

    RTC_DATA_OUT;

    return(da);
}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  write_data_to_ds1302
 *  Description:  ds1302にデータを書き込む
 *
======================================================================
 */
void write_data_to_ds1302(unsigned char addr, unsigned char da)
{
    RTC_DATA_OUT;
    RTC_RST->DATA = 0;//リセット、低れベール有効
    RTC_SCLK->DATA = 0;
    delay(40);
```

```
    RTC_RST->DATA = 1;

    write_1byte_to_ds1302(addr); // アドレス、コマンド
    write_1byte_to_ds1302(da); // 1Byteデータを書き込む

    RTC_SCLK->DATA = 1;
    RTC_RST->DATA = 0;

    delay(40);
}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  read_data_from_ds1302
 *  Description:  ds1302からデータを読み込み
 * ======================================================================
 */
unsigned char read_data_from_ds1302(unsigned char addr)
{
    unsigned char da;

    RTC_RST->DATA = 0;
    RTC_SCLK->DATA = 0;

    delay(40);

    RTC_RST->DATA = 1;

    write_1byte_to_ds1302(addr);
    da = read_1byte_from_ds1302();

    RTC_SCLK->DATA = 1;

    RTC_RST->DATA = 0;
```

```
    delay(40);

    return(da);
}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  set_time
 *  Description:  日付設定
 *
======================================================================
 */
void set_time(unsigned char *ti)
{
    unsigned char i;
    unsigned char addr = 0x80;

    write_data_to_ds1302(0x8e,0x00); // コントロールコマンド、WP=0,書込み操作

    for(i =7;i>0;i--){
        write_data_to_ds1302(addr,*ti); // 秒 分 時 日 月 曜日 年

        ti++;
        addr +=2;
    }

    write_data_to_ds1302(0x8e,0x80);// コントロールコマンド、WP=1、書込みロック
}
/*
 * ===  FUNCTION
======================================================================
 *        Name:  get_time
 *  Description:  日付取得、読み込んだ日付がBCDコード、十進に変換必要
 * ======================================================================
```

```
 */
void get_time(char *ti)
{
    unsigned char i;
    unsigned char addr = 0x81;
    char time;

    for (i=0;i<7;i++){
        time=read_data_from_ds1302(addr);//読み込んだ日付がBCDコード
        ti[i] = time/16*10+time%16;//秒 分 時 日 月 曜日 年
        addr += 2;
    }
}
```

３．メイン関数修正

下記のように修正します。

```
/*
 *
=====================================================================
===============
 *
 *      Filename:  main.c
 *
 *    Description:  RTC実験
 *
 *       Version:  1.0.0
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *
 *
=====================================================================
===============
 */
```

```
/*---------------------------------------------------------------------
---------
 * Include

*---------------------------------------------------------------------
--------*/
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include "../inc/uart.h"
#include "../inc/ds1302.h"


/*---------------------------------------------------------------------
---------
 * Variable

*---------------------------------------------------------------------
--------*/
unsigned char time[7] = {0x00,0x19,0x14,0x17,0x03,0x17,0x10};//フォーマッ
ト: 秒 分 時 日 月 曜日 年

/*
 * ===  FUNCTION
======================================================================
 *        Name:  main
 *  Description:  メイン関数
 *
======================================================================
===============
 */
int main()
{
   unsigned char buffer[50]="¥0";
```

```
    ds1302.set_time(time); //RTCの時間を設定

    while(1){
        ds1302.get_time(time);          //捕捉時間
        //日付をフォーマット、例：2010-4-4 15:25:00
        sprintf(buffer,"20%d-%d-%d %d:%d:%d¥n",
time[6],time[4],time[3],time[2],time[1],time[0]);
        //シリアルポートで送信
        uart.send_string(sizeof(buffer),buffer);

//printf("20%d-%d-%d %d:%d:%d¥n",time[6],time[4],time[3],time[2],time[
1],time[0]);
        usleep(100000);
    }

    return 0;
}
```

上記のソースで日付を取得してからシリアルポートで送信します（前節の内容を復習しますね)、勿論、printf を使ってコンソールに出力しても OK です。
全てファイルを保存しコンパイルしてから開発ボードで結果を見てください。

7.5 SPI

本節から NIOS II で SPI バスの使い方を説明します。

7.5.1 概要

SPI 知識概要：
SPI は英語 Serial Peripheral Interface の略です、即ちシリアル･ペリフェラル･インタフェースです、Motorola 社から開発した同期式シリアル通信モードとなり、これは 4 線式同期バスです、強いハードウエア機能を持つため、SPI に関わるソフトウエアが簡単になり、CPU はより多くの時間で他の作業を処理できます。
SPI 通信の原理は非常にシンプルですが、それはマスター･スレーブモードで動作します、このモードは一般に一つマスタ･デバイスと一つまた複数のスレーブデバイスを持つ、少なくとも 4 つのライン必要、場合により 3 ラインでよいです（一方向の伝送時、即ち半二重モードである)。
下記はすべての SPI デバイスがあるべきものです、MISO（マスターイン･スレーブアウト)、

MOSI（マスターアウト・スレーブイン）、SCK（クロック）、CS（チップセレクト）があります。

■MISO - マスターデバイスのデータで出力、スレーブデバイスから入力;

■MOSI - マスターデバイスのデータから入力、スレーブデバイスで出力;

■SCK - マスタによって生成されるクロック信号;

■CS - スレーブ有効信号、マスタデバイスから制御される。

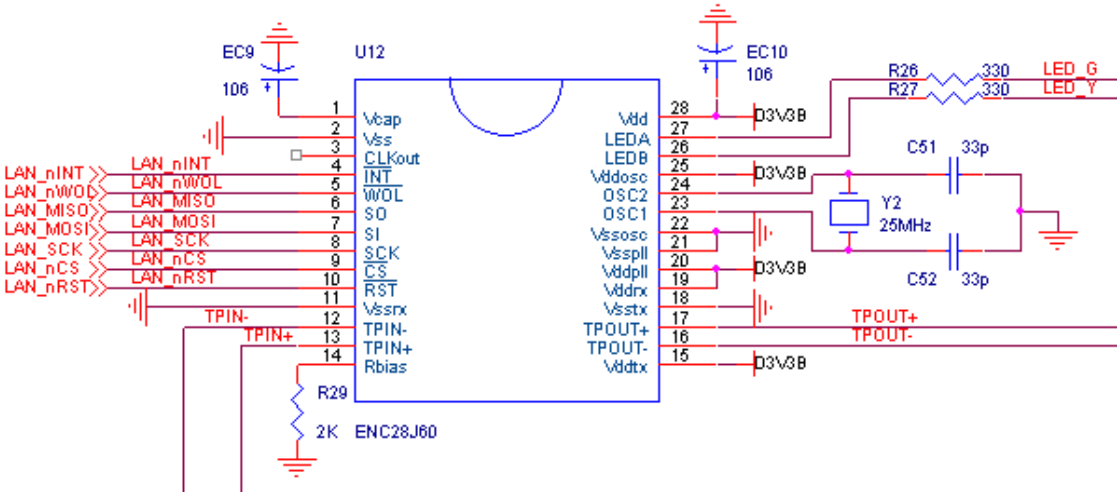
ここで、CS はチップが選択されるかを制御します、即ちチップ･セレクト信号が予め所定のイネーブル信号になる時（高電位または低電位）、このチップの動作が有効であることです。これにより、同じバス上に複数の SPI デバイスを接続することができます。

7.5.2 ハードウエア開発

CycloneIV 開発ボードの LAN インタフェースは SPI バスで実現されます、LAN チップは Microchip 社の ENC28J60 です。まず、この部分の回路を見てみましょう。下記のように、

SPI バスと関連の所：LAN_MISO，LAN_MOSI，LAN_SCK

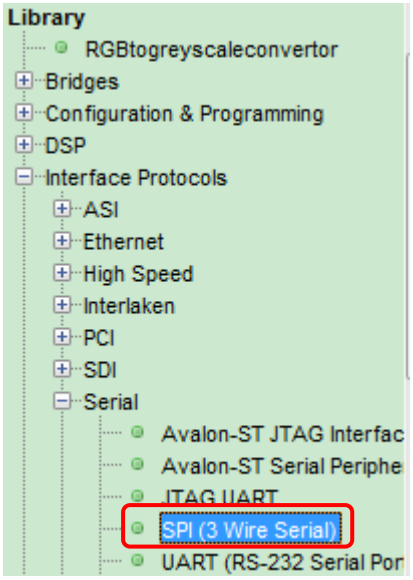
他には PIO モジュールで実現されます、あるピンも使わないです、例えば、LAN_nWOL。



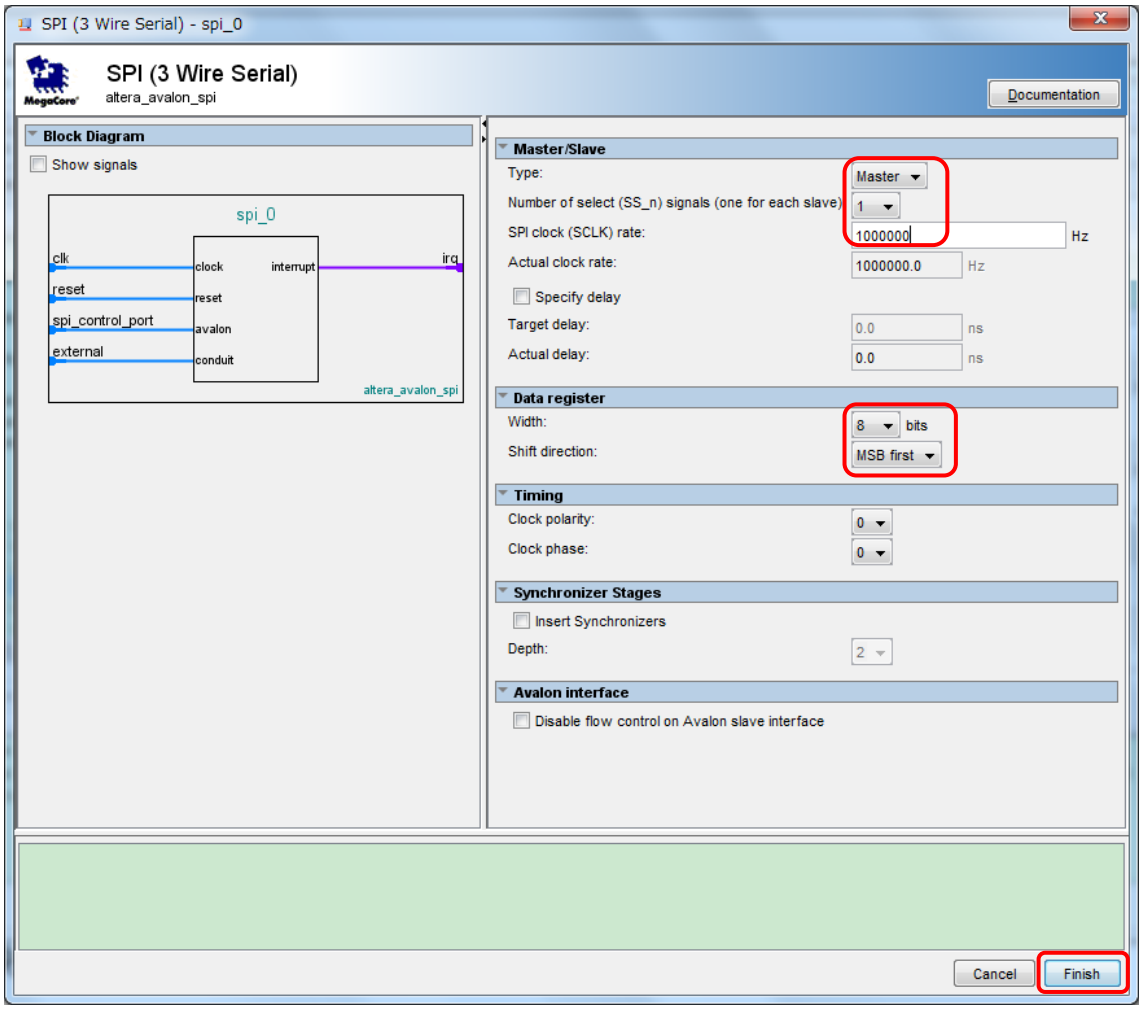
本節は SPI バスの使い方を説明しますので、ENC28J60 はちょっと複雑ですので、ここ説明しません。では、SPI モジュールを構築して行きましょう。

PIO モジュール作成方法は[1．PIO モジュールを IP コアに追加]を参照してください。

RTC の章に続きまして、Quartus II のプロジェクトを開き、SOPC Builder を立ち上げ、下図のようにダブルクリック



Type：「Master」；Number of select：「1」；SPI clock rate：「1000000」；Width：「8」
Shift direction：「MSB first」を設定してから「Finish」ボタン押す

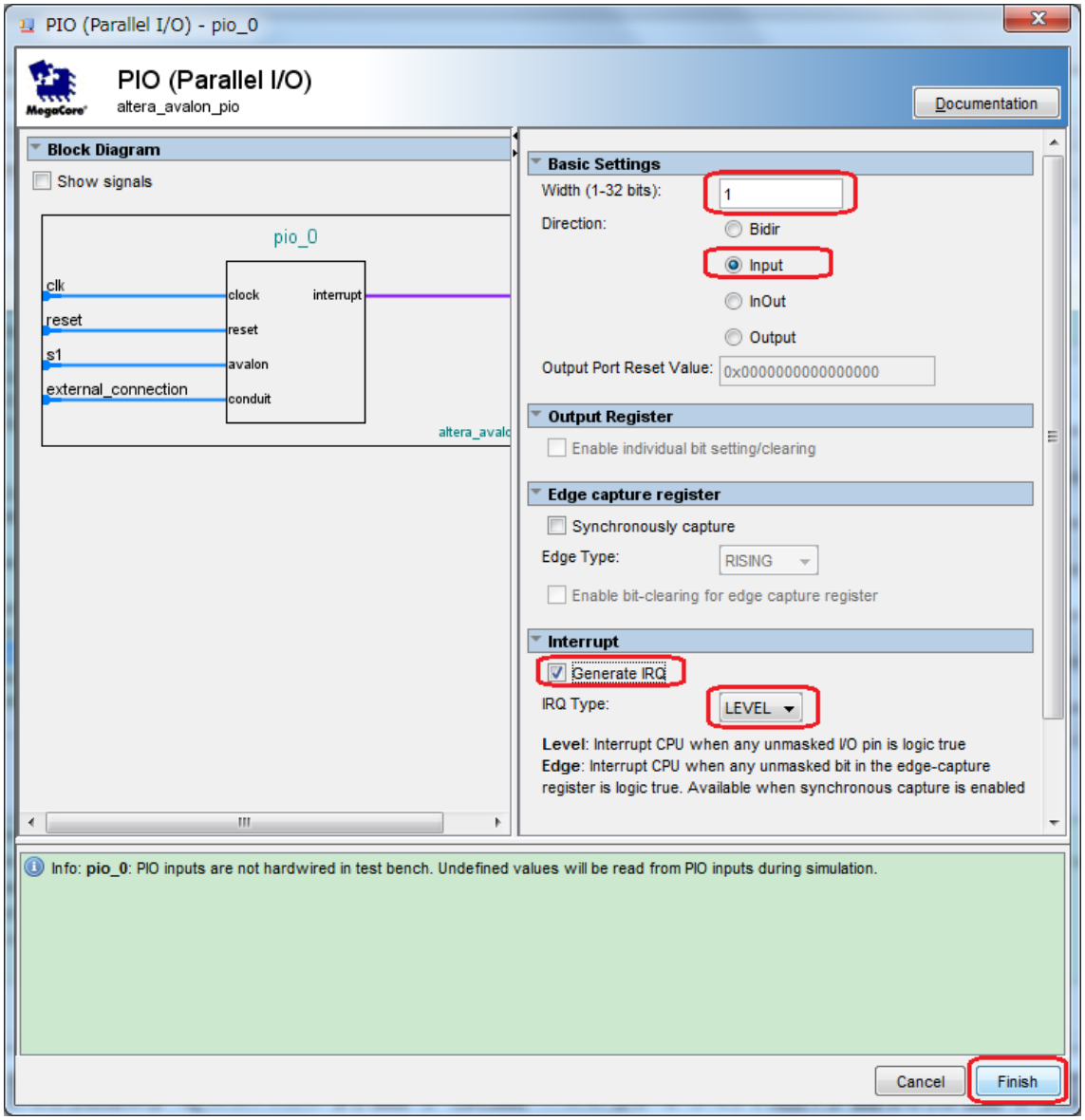


それ以外、他の二つ PIO モジュールも作成必要です。一つは CS 信号制御用です、もう一

つは割り込み信号です。なぜ SPI バス自身の CS 信号を使われないか？理由はソフトウエアの処理により決めらることです。

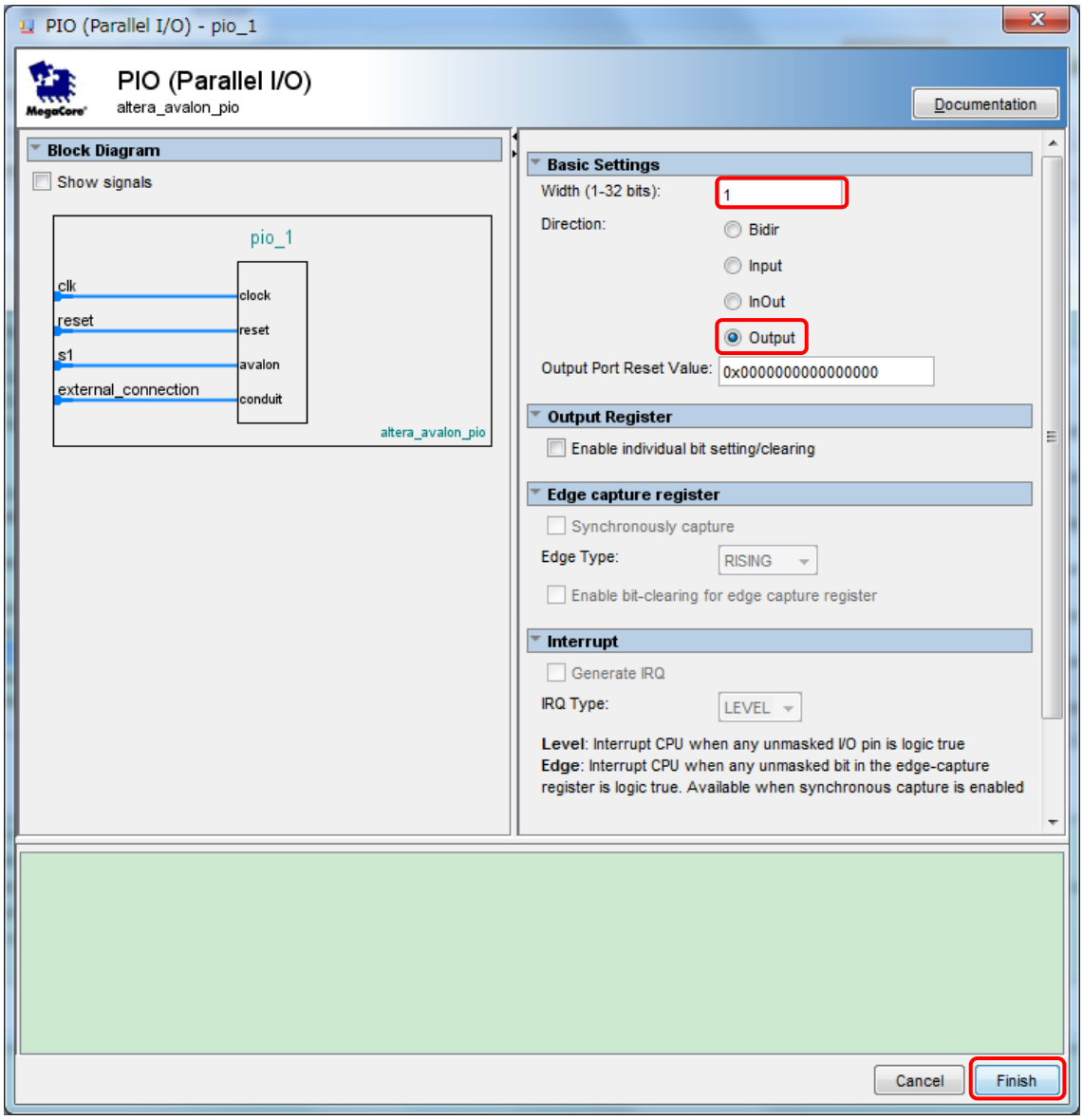
割り込み PIO モジュール：

Width：「1」；Direction：「Input」；Interrupt：「Generate IRQ」；IRQ Type：「LEVEL」を設定してから「Finish」ボタンを押す



CS 制御用の PIO モジュール：

Width：「1」；Direction：「Output」を設定してから「Finish」ボタンを押す

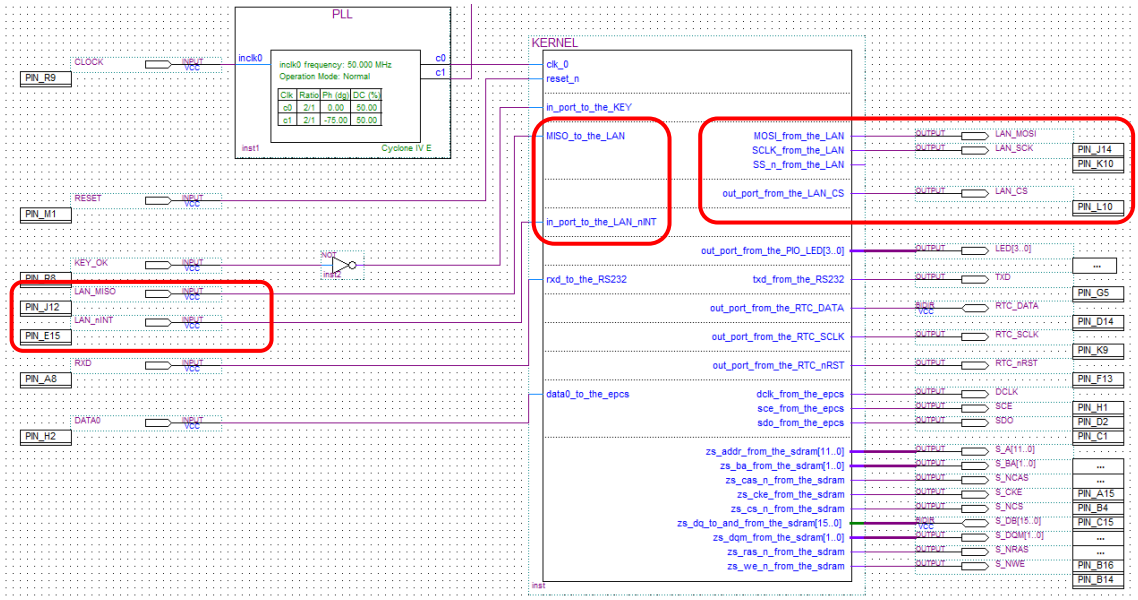


モジュールを作成後、名前を修正します。

最後、ベースアドレス、IRQ を自動割り当てにし、保存してからコンパイルしましょう。

コンパイル完了後、Quartus II メイン画面に戻します。その後、TCL ファイルによりピンの名前を修正して TCL ファイルを実行します。

実行後の様子：



それでは、すべてファイルを保存してからコンパイルします。

7.5.3 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、LAN に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
 /*
 * LAN configuration
 *
 */

#define LAN_NAME "/dev/LAN"
#define LAN_TYPE "altera_avalon_spi"
#define LAN_BASE 0x00001820
......

/*
 * LAN_nINT configuration
 *
 */

#define LAN_NINT_NAME "/dev/LAN_nINT"
```

```
#define LAN_NINT_TYPE "altera_avalon_pio"
#define LAN_NINT_BASE 0x00001890
......
/*
 * LAN_CS configuration
 *
 */

#define LAN_CS_NAME "/dev/LAN_CS"
#define LAN_CS_TYPE "altera_avalon_pio"
#define LAN_CS_BASE 0x000018a0
......
```

これらのソースコードの中、以下のベースアドレスを使われます。

```
#define LAN_BASE 0x00001820
#define LAN_NINT_BASE 0x00001890
#define LAN_CS_BASE 0x000018a0
```

次はプログラムを修正していきましょう。

１．ヘッダファイル修正

①sopc.h ファイルは下記のソースを追加します。

```

/*-------------------------------------------------------------------
---------
 *  Define

*-------------------------------------------------------------------
--------*/
#define _LAN

/*-------------------------------------------------------------------

/*-------------------------------------------------------------------
---------
 *  Peripheral registers structures
```

```
*-------------------------------------------------------------------------
--------*/
/*--------------------------LAN--------------------------------------
---*/

typedef struct{

   volatile unsigned long int RXDATA;

   volatile unsigned long int TXDATA;

   union{
      struct{
         volatile unsigned long int NC            :3;
         volatile unsigned long int ROE           :1;
         volatile unsigned long int TOE           :1;
         volatile unsigned long int TMT           :1;
         volatile unsigned long int TRDY          :1;
         volatile unsigned long int RRDY          :1;
         volatile unsigned long int E            :1;
         volatile unsigned long int NC1           :23;
      }BITS;
      volatile unsigned long int WORD;
   }STATUS;

   union{
      struct{
         volatile unsigned long int NC            :3;
         volatile unsigned long int IROE          :1;
         volatile unsigned long int ITOE          :1;
         volatile unsigned long int NC1           :1;
         volatile unsigned long int ITRDY         :1;
         volatile unsigned long int IRRDY         :1;
         volatile unsigned long int IE           :1;
         volatile unsigned long int NC2           :1;
```

```
            volatile unsigned long int SSO            :21;
        }BITS;
        volatile unsigned long int CONTROL;
    }CONTROL;

    unsigned long int RESERVED0;
    unsigned long int SLAVE_SELECT;

}SPI_ST;



/*-------------------------------------------------------------------------
---------
 *  Peripheral declaration

*-------------------------------------------------------------------------
--------*/
#ifdef _LAN
#define LAN             ((SPI_ST *) LAN_BASE)
#define LAN_CS          ((PIO_STR *) LAN_CS_BASE)
#define LAN_NINT        ((PIO_STR *) LAN_NINT_BASE)
#endif /*_LAN*/
```

これらのソースは下記資料（P93）により作られます、構造体の順番は下記テーブルの順番により設定されます、RS232 と同じです。

《Embedded Peripherals IP User Guide》:

http://www.altera.com/literature/ug/ug_embedded_ip.pdf

構造体以外、ベースアドレスの定義も追加されます。

Table 8–3. Register Map for SPI Master Device

Internal Address	Register Name	Type [R/W]	32..11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata *(1)*	R	RXDATA (n-1..0)											
1	txdata *(1)*	W	TXDATA (n-1..0)											
2	status *(2)*	R/W				E	RRDY	TRDY	TMT	TOE	ROE			
3	control	R/W		SSO *(3)*		IE	IRRDY	ITRDY		ITOE	IROE			
4	Reserved	—												
5	slaveselect *(3)*	R/W	Slave Select Mask											

Notes to Table 8–3:

(1) Bits 31 to *n* are undefined when *n* is less than 32.
(2) A write operation to the status register clears the ROE, TOE, and E bits.
(3) Present only in master mode.

②enc28j60.h 新規作成

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック

enc28j60.h ファイル内容（一部ソースが抜粋、他にはマイクロ定義等があります。弊社から提供のサンプルソースをそのままコピーするのが楽になります。）

```
/*---------------------------------------------------------------------
----------
 *  Data Struct

*---------------------------------------------------------------------
---------*/
typedef const struct{
    unsigned char (* read_control_register)(unsigned char address);
    void (* initialize)(void);
    void (* packet_send)(unsigned short len,unsigned char * packet);
    unsigned int (* packet_receive)(unsigned short maxlen,unsigned char
* packet);
}ENC28J60;

/*---------------------------------------------------------------------
----------
 *  external variable

*---------------------------------------------------------------------
---------*/
extern ENC28J60 enc28j60;
```

２．ドライバーファイル作成

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「enc28j60.c」を作成します。（一部ソースが抜粋、弊社から提供のサンプルソースをそのままコピーするのが楽になります。）

```
/*
 *
====================================================================
================
 *
 *       Filename:  enc28j60.c
 *
 *    Description:
 *
 *        Version:  1.0.0
 *        Created:  2012.1.31
 *       Revision:  none
 *       Compiler:  Nios II 11.1 IDE
 *           URL:  http://www.csun.co.jp
 *
 *
====================================================================
================
 */


/*--------------------------------------------------------------------
----------
 *  Include

*--------------------------------------------------------------------
---------*/

#include "../inc/enc28j60.h"
```

```
#include "../inc/sopc.h"
#include <stdio.h>


/*---------------------------------------------------------------------
----------
 *  Function Prototype

*---------------------------------------------------------------------
---------*/

static unsigned char enc28j60_read_control_register(unsigned char
address);
static void enc28j60_initialize(void);
static void enc28j60_packet_send(unsigned short len,unsigned char *
packet);
static unsigned int enc28j60_packet_receive(unsigned short
maxlen,unsigned char * packet);

/*---------------------------------------------------------------------
----------
 *  Variable

*---------------------------------------------------------------------
---------*/
//構造体初期化
ENC28J60 enc28j60={
    .read_control_register  = enc28j60_read_control_register,
    .initialize           = enc28j60_initialize,
    .packet_send          = enc28j60_packet_send,
    .packet_receive        = enc28j60_packet_receive
};

static unsigned char enc28j60_bank = 1;
static unsigned short next_packet_pointer;
```

```
/*
 * ===  FUNCTION
======================================================================
=
 *        Name:  set_cs
 *  Description:
 *
======================================================================
================
 */
static void set_cs(unsigned char level)
{
    if(level)
        LAN_CS->DATA = 1;
    else
        LAN_CS->DATA = 0;
}

/*
 * ===  FUNCTION
======================================================================
=
 *        Name:  enc28j60_write_operation
 *  Description:
 *
======================================================================
================
 */
static void enc28j60_write_operation(unsigned char op, unsigned char
address, unsigned char data)
{
    //ｃSを低電位に設定、低電位が有効のため
    set_cs(0);
    //まず、ライトコマンドを発行、ステータスレジスタのTMTを検知しています、TMTが0
になる時、発送中の状態です。
    //TMTが1になったら、発送完了、その時、レジスタが空いている
```

```
    LAN->TXDATA = (op | (address & 0x1F)); // write command
    while(!(LAN->STATUS.BITS.TMT));
    //データ書き込み、ステータスレジスタのTMTを検知しています、TMTが0になる時、発送中の状態です。
    //TMTが1になったら、発送完了、その時、レジスタが空いている
    LAN->TXDATA = data;         // write data
    while(!(LAN->STATUS.BITS.TMT));
    //発送完了後、ｃSを高電位に設定
    set_cs(1);
}

/*
 * ===  FUNCTION
======================================================================
=
 *        Name:  enc28j60_read_operation
 *  Description:
 *
======================================================================
================
 */
static unsigned char enc28j60_read_operation(unsigned char op,unsigned
char address)
{
    unsigned char data;
    //ｃSを低電位に設定、低電位が有効のため
    set_cs(0);
    //まず、ライトコマンドを発行、ステータスレジスタのTMTを検知しています、TMTが0になる時、発送中の状態です。
    //TMTが1になったら、発送完了、その時、レジスタが空いている
    LAN->TXDATA = op|(address&0x1f);
    while(!(LAN->STATUS.BITS.TMT));
    //データ書込み、0x00を発送します。0x00がラダムデータ、これは時計を有効するためです。
    LAN->TXDATA = 0x00;  //0x00 is random number ,to enable clock
    while(!(LAN->STATUS.BITS.TMT));
```

```
    //MACとMIIのレジスタから読み込んだ一番目のバイトが無効なので、2回書き込む必要
    if(address&0x80){
        LAN->TXDATA = 0x00;
        while(!(LAN->STATUS.BITS.TMT));    //The first byte that MAC and
MII registers read  is invalid,so they need to read twice
    }

    //データを読み込み
    data = LAN->RXDATA;
    //読み込み完了後、ｃSを高電位に設定
    set_cs(1);

    return data;
}
```

３．メイン関数修正

ここ初期化だけを行います、LAN で発送信プログラムは TCP/IP プロトコルでしか実現できませんので、TCP/IP プロトコルを説明時に一緒に紹介します。

```
/*
 *
=====================================================================
================
 *
 *      Filename:  main.c
 *
 *    Description:  SPIバス実例
 *
 *       Version:  1.0.0
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *
 *          URL:  http://www.csun.co.jp
 *
```

```
 *
=====================================================================
================
 */

/*---------------------------------------------------------------------
----------
 *  Include

*---------------------------------------------------------------------
---------*/
#include "../inc/enc28j60.h"

/*
 * ===  FUNCTION
=====================================================================
=
 *        Name:  main
 *  Description:
 *
=====================================================================
================
 */
int main()
{
    enc28j60.initialize();

    return 0;
}
```

保存してコンパイルしましょう。現時点、プログラムは開発ボードにダウンロードしなくてよいです、ダウンロードしても LAN の初期化だけです。

7.6 IIC

本節から NIOS II で IIC バスの使い方を説明します。

7.6.1 概要

IIC バスの使い方を説明するため、ハードウエアは 512 バイトの EEPROM：24LC04 を使

われます。NIOS II の中、IIC インタフェースがありません。この機能を実現するため、二つ方法があります。一つは IP マクロの作成あるいは既存 IP マクロから移植します、もう一つは IO で IIC バスプロトコルを模擬します、本節は後者の方法を利用して 24LC04 に読み込み・書込みをコントロールします。

IIC バスの原理：

IIC(Inter－Integrated Circuit)バスは Philips 社から開発した２線式シリアルバスです、このバスでマイクロコントローラと周辺デバイスを接続できます。データ SDA と時計 SCL で構成されるシリアルバスとなります、送信と受信が両方もできます。CPU と制御される IC の間、IC と IC の間は双方向で送受信です、最高スピードが 100kbps です。

送受信の間に三種類信号があります：

開始信号、終了信号、応答信号を分けられます。

開始信号：SCL が高電位の場合、SDA が高電位から低電位に変更、送信を始めます。

終了信号：SCL が高電位の場合、SDA が低電位から高電位に変更、送信を終わらせます。

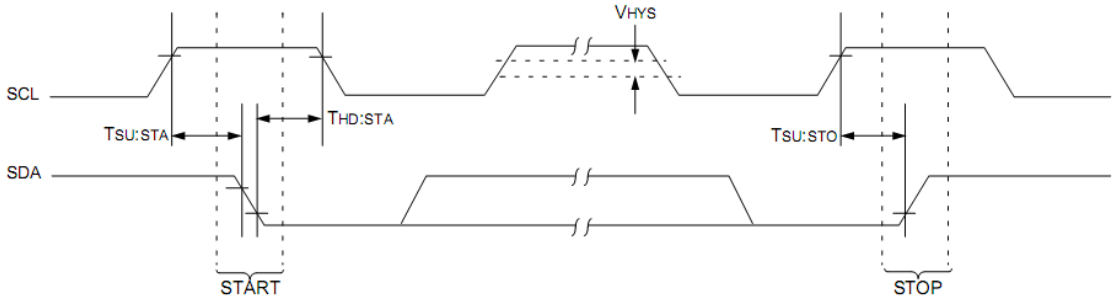
応答信号：受信の IC は 8bit データを受けた後、送信の IC に特定の低電位プラスを送ってデータを受けた事を応答します。

CPU は制御されるユニットにある信号を送信し、制御されるユニットから応答信号を待ちます、CPU が応答信号を受信した後、実際の状況により継続送信するかを判断します。

応答信号を受信していない場合、制御されるユニットが故障になると判断します。

これらの信号の中、開始信号が必要です、終了信号と応答信号がなくても良いです。

下記は IIC バスのタイミング図を示します。

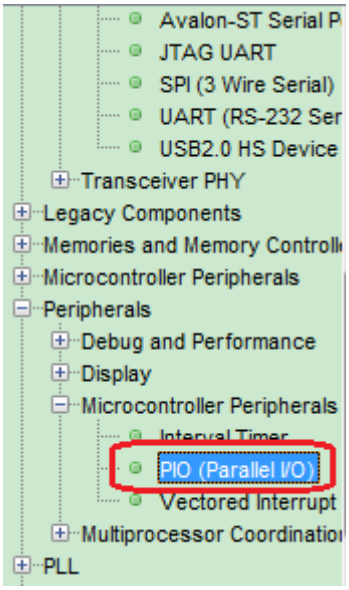


7.6.2 ハードウエア開発

まず、IP マクロに二つ IO モジュールを追加必要です、名前が SCL と SDA を付けられます。SCL は output(出力)、SDA は Bidirection（双方向）です。

※PIO モジュール作成方法は１．PIO モジュールを IP コアに追加を参照してください。

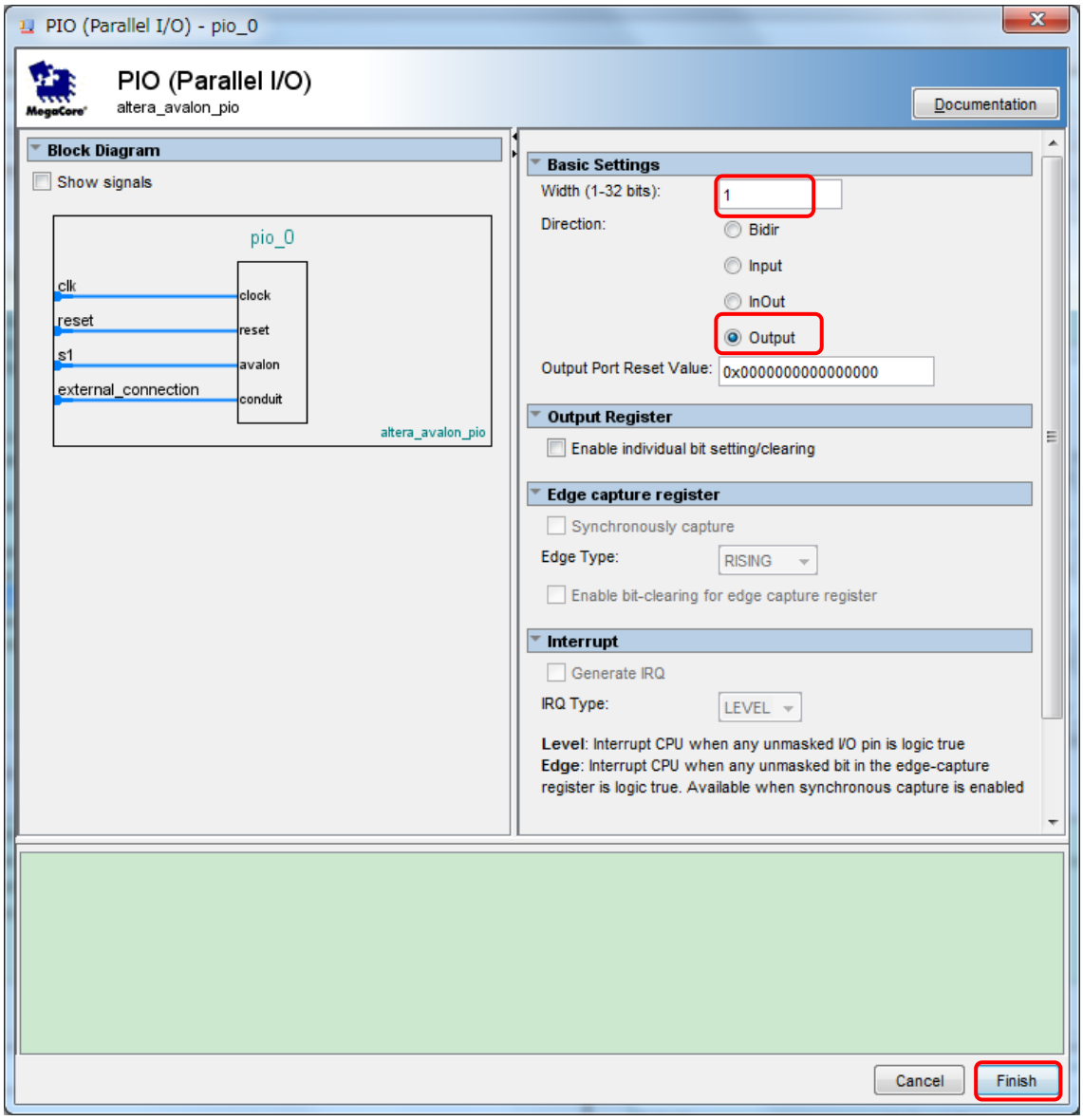
SPI 実例の章に続きまして、Quartus II のプロジェクトを開き、SOPC Builder を立ち上げ、下図のようにダブルクリック



Width：「1」

Direction：「output」を設定してから「Finish」ボタン押す

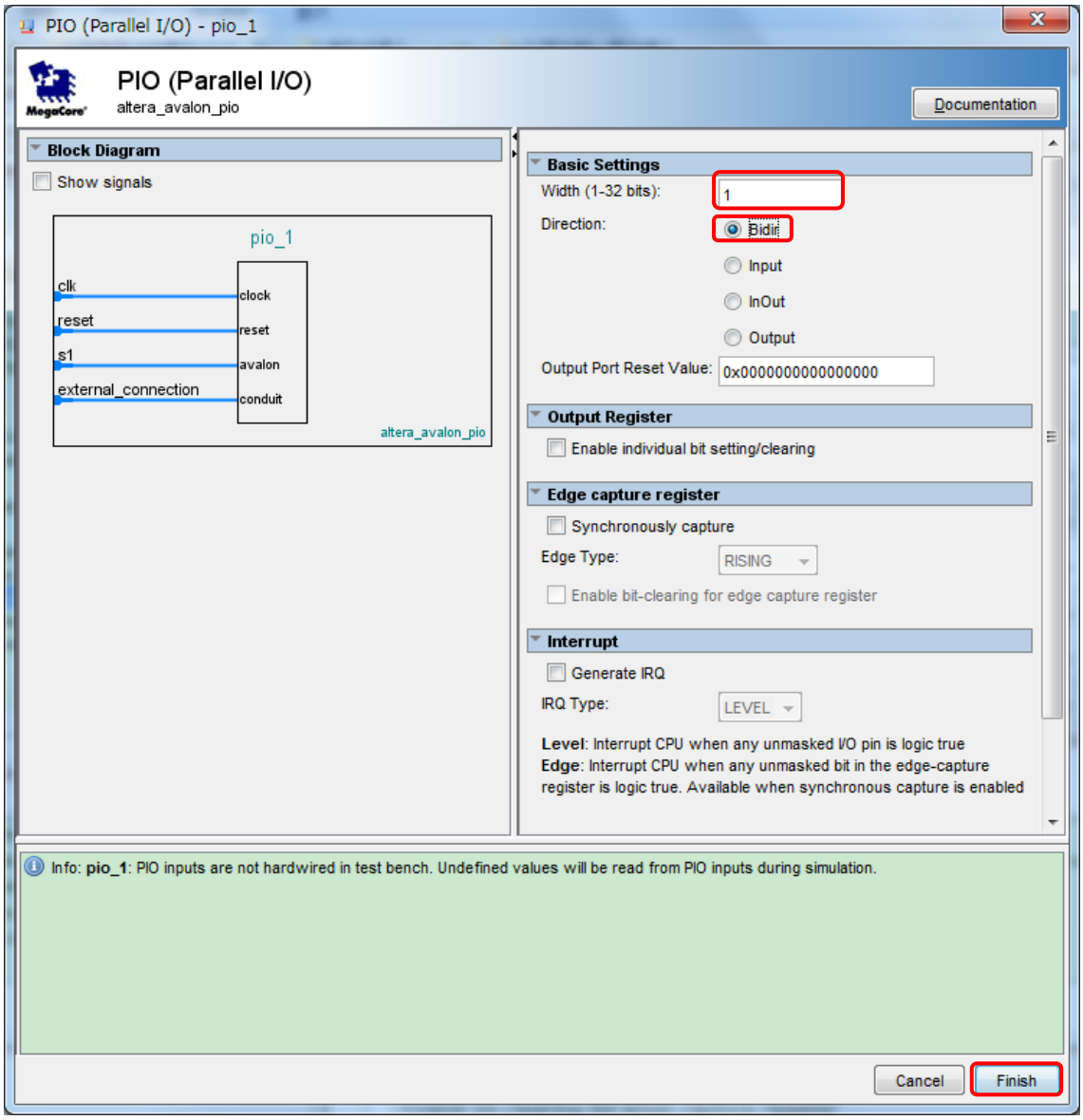
※SCL モジュール



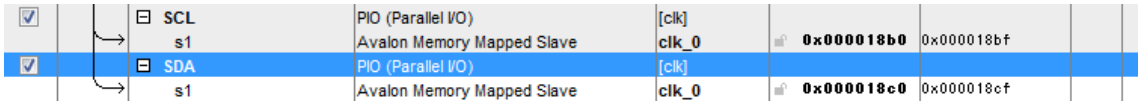
次は SDA モジュールを作成

Width：「1」

Direction：「Bidir」を設定してから「Finish」ボタン押す



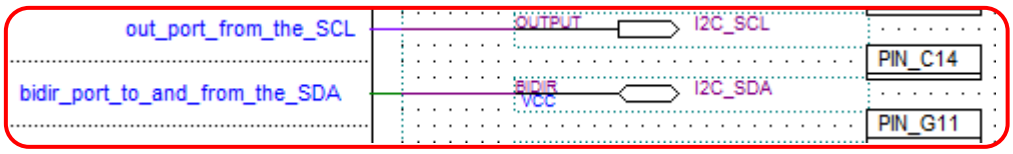
名前を修正後、ベースアドレスと IRQ を自動に割り当てにした後様子：



保存してから「Generate」をクリックしコンパイルしましょう。

コンパイル完了後、Quartus II メイン画面に戻します。その後、TCL ファイルによりピンの名前を修正して TCL ファイルを実行します。

実行後の様子：



それでは、すべてファイルを保存してからコンパイルします。

7.6.3 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、SCL と SDA に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
 /*
 * SCL configuration
 *
 */
#define SCL_NAME "/dev/SCL"
#define SCL_TYPE "altera_avalon_pio"
#define SCL_BASE 0x000018b0
……
/*
 * SDA configuration
 *
 */
……
#define SDA_NAME "/dev/SDA"
#define SDA_TYPE "altera_avalon_pio"
#define SDA_BASE 0x000018c0
……
```

次はプログラムを修正していきましょう。

NIOS II API の使用方法を紹介するため、本節は NIOS II の API を使って IIC バスの使い方を説明します。（NIOS II API をあまりお勧めしません、理由は前述した通りです。）

１．ヘッダファイル修正

iic.h 新規作成

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック

iic.h ファイル内容（弊社から提供のサンプルソースをそのままコピーしても良い。）

```
/*
 *
==========================================================================
================
```

```
 *
 *      Filename:  iic.h
 *
 *    Description:
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
================
 */

#ifndef IIC_H_
#define IIC_H_

/*-----------------------------------------------------------------------------
-------------------
 *  Define

*-----------------------------------------------------------------------------
-----------------*/
#define   OUT    1
#define   IN     0

typedef struct{
   void (* write_byte)(unsigned short addr, unsigned char dat);
   unsigned char (* read_byte)(unsigned short addr);
}IIC;

extern IIC iic;

#endif /*IIC_H_*/
```

２．ドライバーファイル作成

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「iic.c」を作成します。

※IOWR_ALTERA_AVALON_PIO_DATA という関数は NIOS II の API です、以前と違います、特に「sopc.h」ファイルに構造体等を追加必要ありません。

```
/*
 *
================================================================================
================
 *
 *      Filename:  iic.c
 *
 *    Description:
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
================
 */

/*-----------------------------------------------------------------------------
------------------
 *  Include

*-----------------------------------------------------------------------------
-----------------*/
#include <stdio.h>
#include <sys/unistd.h>
#include <io.h>
```

```
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "../inc/iic.h"

/*---------------------------------------------------------------------
------------------
 *  Variable

*---------------------------------------------------------------------
-----------------*/
static alt_u8 read_byte(alt_u16 addr);
static void write_byte(alt_u16 addr, alt_u8 dat);

/*---------------------------------------------------------------------
------------------
 *  Struct

*---------------------------------------------------------------------
-----------------*/
IIC iic ={
    .write_byte = write_byte,
    .read_byte = read_byte
};



/*
 * ===  FUNCTION  ======================================================
 *         Name:  start
 *  Description:  IIC起動
 * =====================================================================
 */
static void start(void)
{
    IOWR_ALTERA_AVALON_PIO_DIRECTION(SDA_BASE, OUT);
    IOWR_ALTERA_AVALON_PIO_DATA(SDA_BASE, 1);
```

```
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 1);
    usleep(10);
    IOWR_ALTERA_AVALON_PIO_DATA(SDA_BASE, 0);
    usleep(5);
}
/*
 * ===  FUNCTION  ======================================================
 *         Name:  uart_send_byte
 *  Description:  IIC停止
 * =====================================================================
 */
static void stop(void)
{
    IOWR_ALTERA_AVALON_PIO_DIRECTION(SDA_BASE, OUT);
    IOWR_ALTERA_AVALON_PIO_DATA(SDA_BASE, 0);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);
    usleep(10);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 1);
    usleep(5);
    IOWR_ALTERA_AVALON_PIO_DATA(SDA_BASE, 1);
    usleep(10);
}
/*
 * ===  FUNCTION  ======================================================
 *         Name:  ack
 *  Description:  IIC応答
 * =====================================================================
 */
static void ack(void)
{
    alt_u8 tmp;

    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);
    IOWR_ALTERA_AVALON_PIO_DIRECTION(SDA_BASE, IN);
    usleep(10);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 1);
```

```
    usleep(5);
    tmp = IORD_ALTERA_AVALON_PIO_DATA(SDA_BASE);
    usleep(5);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);
    usleep(10);

    while(tmp);
}
/*
 * ===  FUNCTION  ======================================================
 *         Name:  iic_write
 *  Description:  IICで1byteを書き込む
 * =====================================================================
 */
void iic_write(alt_u8 dat)
{
    alt_u8 i, tmp;

    IOWR_ALTERA_AVALON_PIO_DIRECTION(SDA_BASE, OUT);

    for(i=0; i<8; i++){
        IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);
        usleep(5);
        tmp = (dat & 0x80) ? 1 : 0;
        dat <<= 1;
        IOWR_ALTERA_AVALON_PIO_DATA(SDA_BASE, tmp);
        usleep(5);
        IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 1);
        usleep(10);
    }
}
/*
 * ===  FUNCTION  ======================================================
 *         Name:  read
 *  Description:  IICから1byteを読み込む
 * =====================================================================
```

```
 */
static alt_u8 iic_read(void)
{
    alt_u8 i, dat = 0;

    IOWR_ALTERA_AVALON_PIO_DIRECTION(SDA_BASE, IN);

    for(i=0; i<8; i++){
        IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);
        usleep(10);
        IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 1);
        usleep(5);
        dat <<= 1;
        dat |= IORD_ALTERA_AVALON_PIO_DATA(SDA_BASE);
        usleep(5);
    }

    usleep(5);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);
    usleep(10);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 1);
    usleep(10);
    IOWR_ALTERA_AVALON_PIO_DATA(SCL_BASE, 0);

    return dat;
}

/*
 * ===  FUNCTION  ======================================================
 *         Name:  write_byte
 *  Description:  EEPROMに1byteを書き込む
 * =====================================================================
 */
static void write_byte(alt_u16 addr, alt_u8 dat)
{
    alt_u8 cmd;
```

```
    cmd = (0xa0 | (addr >> 7)) & 0xfe;

    start();
    iic_write(cmd);
    ack();
    iic_write(addr);
    ack();
    iic_write(dat);
    ack();
    stop();
}
/*
 * ===  FUNCTION  ======================================================
 *         Name:  read_byte
 *  Description:   EEPROMから1byteを読み込む
 * =====================================================================
 */
static alt_u8 read_byte(alt_u16 addr)
{
    alt_u8 cmd, dat;
    cmd = (0xa0 | (addr >> 7)) & 0xfe;

    start();
    iic_write(cmd);
    ack();
    iic_write(addr);
    ack();
    start();
    cmd |= 0x01;
    start();
    iic_write(cmd);
    ack();
    dat = iic_read();
    stop();

    return dat;
```

```
}
```

３．メイン関数修正

```
/*
 *
=====================================================================
================
 *
 *      Filename:  main.c
 *
 *    Description:  IICバス実例
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
================
 */

/*---------------------------------------------------------------------
-------------------
 *  Include

*---------------------------------------------------------------------
-----------------*/
#include <unistd.h>
#include "../inc/iic.h"
#include <stdio.h>
#include "alt_types.h"

/*---------------------------------------------------------------------
```

```
-------------------
 *  Variable
 *
*-----------------------------------------------------------------------
-----------------*/
alt_u8 write_buffer[512], read_buffer[512];

/*
 * ===  FUNCTION
=======================================================================
=
 *         Name:  main
 *  Description:
 *
=======================================================================
================
 */
int main()
{
    alt_u16 i, err;
    alt_u8 dat;

    printf("¥nWriting data to EEPROM!¥n");

    //512btyeのデータを書き込み、前の256byteが数字0から255、後ろの256byteが1
    for(i=0; i<512; i++){
        if(i<256)
            dat = i;
        else
            dat = 1;

        iic.write_byte(i, dat);
        write_buffer[i] = dat;
        printf("0x%02x ", dat);
        usleep(10000);
    }
```

```
    printf("¥nReading data from EEPROM!¥n");

    //512byteのデータを読み込んでプリンタ
    for(i=0; i<512; i++){
        read_buffer[i] = iic.read_byte(i);
        printf("0x%02x ", read_buffer[i]);
        usleep(1000);
    }

    err = 0;
    printf("¥nVerifing data!¥n");

    //データが同じかを比較し、違いたっら、書込み・読み込みが何にかエラーが発生した事を明らかにする
    for(i=0; i<512; i++){
        if(read_buffer[i] != write_buffer[i])
            err ++;
    }

    if(err == 0)
        printf("¥nData write and read successfully!¥n");
    else
        printf("¥nData write and read failed!--%d errors¥n", err);

    return 0;
}
```

プログラムは簡単ですが、IIC バスの使い方をある程度理解できるでしょう。保存してコンパイルしましょう。コンパイル後、ボードにダウンロードして実行すれば、
コンソールから結果を見ましょう。

7.7 タイマー

本節から NIOS II でタイマーの使い方を説明します。

7.7.1 概要

タイマーは良く使われます、時計サイクルを数えて定期的な割り込み信号を生成するハー

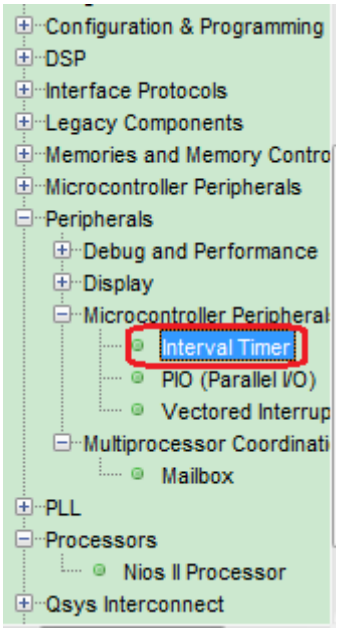
ドウエアです。定期的なイベントを処理する場合良く使われます、例えば、サンプリング周波数、アラム等。多くの資料はシステムクロック、Timestamp、ウォッチドッグを紹介されますが、タイマーの本当の機能をどう使うかをあまり説明していないです。そうすると、「IP マクロが一つシステムクロックしかありません、二つタイマーを使えないか」こういう問題がぶつかったら、どのように対応しますか？システムクロックで実現できません、NIOS II のタイマーを使うしかありません。

7.7.2 ハードウエア開発

まず、タイマーモジュールを構築して行きましょう。

PIO モジュール作成方法は１．PIO モジュールを IP コアに追加を参照してください。

IIC バスの章に続きまして、Quartus II のプロジェクトを開き、SOPC Builder を立ち上げ、下図のようにダブルクリック



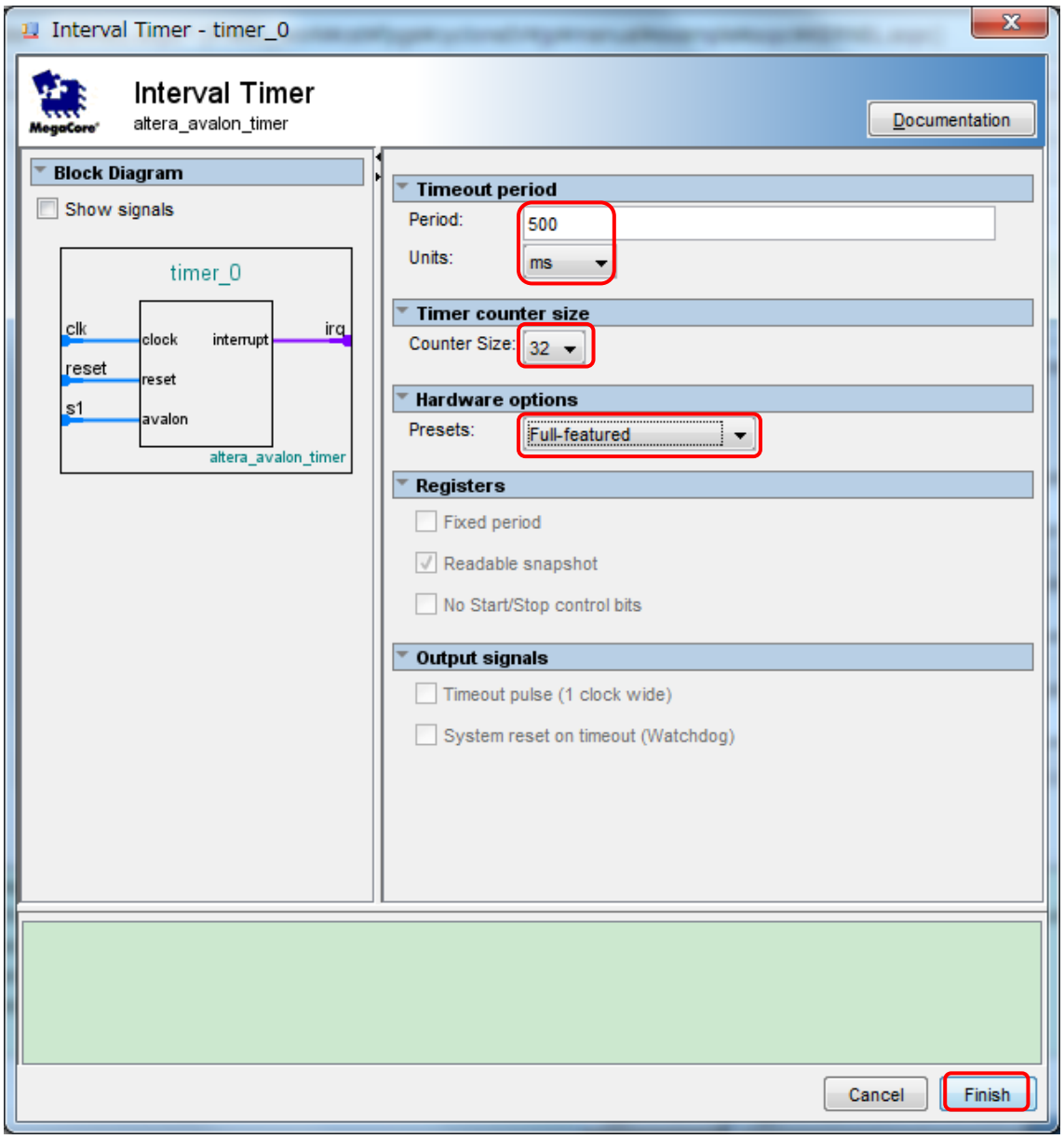
Period：「500」；

Units：「ms」；

Counter Size：「32」；

Preset：「Full-featured」

を設定してから「Finish」ボタン押す



ここ二つタイマーを使いますので、もう一つは同じ様に作成してください。

この二つタイマーでは４つ LED も使う必要ですので、前にも作成しましたので、飛ばします。

モジュール名前も timer1,timer2 に修正します。

次はベースアドレス、IRQ を自動割り当てにします、保存しコンパイルします。

コンパイル完了後、Quartus II メイン画面に戻します。KERNEL を更新しても特にピンの変更がありません。

それでは、すべてファイルを保存してからコンパイルします。

7.7.3 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、タイマーに関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
/*
 * timer1 configuration
 *
 */

#define TIMER1_NAME "/dev/timer1"
#define TIMER1_TYPE "altera_avalon_timer"
#define TIMER1_BASE 0x00001840
……
/*
 * timer2 configuration
 *
 */

#define TIMER2_NAME "/dev/timer2"
#define TIMER2_TYPE "altera_avalon_timer"
#define TIMER2_BASE 0x00001860
……
```

次はプログラムを修正していきましょう。

タイマーのレジスタは下記資料（P256）を参照します。

《Embedded Peripherals IP User Guide》：

http://www.altera.com/literature/ug/ug_embedded_ip.pdf

下記テーブルにより、タイマーにはステータスレジスタ、コントロールレジスタ、タイマーの上位 16 桁、下位 16 桁、それ以外、snap の上位 16 桁、下位 16 桁もあります。本節は snap 以外の三つレジスタを使います。

Table 28–3. Register Map—32-bit Timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	*(1)*					RUN	TO
1	control	RW	*(1)*			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Note to Table 28–3:

(1) Reserved. Read values are undefined. Write zero.

１．メイン関数修正

```
/*
 *
=====================================================================
================
 *
 *       Filename:  main.c
 *
 *    Description:  タイマー実例
 *
 *        Version:  1.0.1
 *        Created:  2012.1.31
 *       Revision:  none
 *       Compiler:  Nios II 11.1 IDE
 *           URL:  http://www.csun.co.jp
 *
 *
=====================================================================
================
 */

/*---------------------------------------------------------------------
----------
 *  Include

*---------------------------------------------------------------------
```

```
---------*/
#include <stdio.h>
#include <sys/unistd.h>
#include <io.h>
#include <string.h>

#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "../inc/sopc.h"

/*-----------------------------------------------------------------------
----------
 *  Variable

*-----------------------------------------------------------------------
---------*/
static void timer_init(void);    //割り込み初期化を宣言

int i = 0,j = 0,flag;
alt_u32 timer_prd[4] = {5000000, 10000000, 50000000, 100000000};//タイ
マーのサイクルを定義



/*
 * ===  FUNCTION
======================================================================
=
 *         Name:  main
 *  Description:
 *
======================================================================
================
 */
```

```
int main(void)
{
    //Timerを初期化
    timer_init();

    while(1);

    return 0;
}

/*
 * ===  FUNCTION
======================================================================
=
 *        Name:  ISR_timer
 *  Description:
 *
======================================================================
================
 */
static void ISR_timer1(void *context, alt_u32 id)
{
    //水ランプが点滅します、全部４つLED
    LED->DATA = 1<<i;

    i++;

    if(i == 4)
        i = 0;

    //タイマーの割り込みフラグレジストをクリア
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER1_BASE, 0x00);
}

/*
 * ===  FUNCTION
```

```
/*
 * ===============================================================================
 *
 *         Name:  ISR_timer2
 *  Description:  Timer2でTimer1のサイクルを変更、変更後、タイマーを再起動必要
 *
 * ===============================================================================
 */
static void ISR_timer2(void *context, alt_u32 id)
{
    //Timer1のサイクルを変更
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER1_BASE, timer_prd[j]);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER1_BASE, timer_prd[j] >> 16);

    //タイマーを再起動
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER1_BASE, 0x07);

    //点滅頻度が先に高くにし、その後、低くにし、最後、また高くにする
    if(j == 0)
        flag = 0;
    if(j == 3)
        flag = 1;

    if(flag == 0){
        j++;
    }
    else{
        j--;
    }

    //割り込みフラグをクリア
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER2_BASE, 0);
}

/*
 * ===  FUNCTION
```

```
/*============================================================================
 *        Name:  timer_init
 *  Description:
 *
 *==========================================================================*/
void timer_init(void)    //割り込み初期化
{
    //Timer1の割り込みフラグレジストをクリア
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER1_BASE, 0x00);
    //Timer1サイクルを設定
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER1_BASE,80000000);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER1_BASE, 80000000 >> 16);
    //Timer1割り込みを有効にする
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER1_BASE, 0x07);
    //Timer1割り込みをレジスト
    alt_irq_register(TIMER1_IRQ, (void *)TIMER1_BASE, ISR_timer1);

    //Timer2の割り込みフラグレジストをクリア
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER2_BASE, 0x00);
    //Timer2サイクルを設定
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER2_BASE,400000000);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER2_BASE, 400000000 >> 16);
    //Timer2割り込みを有効にする
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER2_BASE, 0x07);
    //Timer2割り込みをレジスト
    alt_irq_register(TIMER2_IRQ, (void *)TIMER2_BASE, ISR_timer2);
}
```

上記ソースは HAL の API を使って実現します、勿論、構造体を作成して以前の方法でも実現できます。構造体の作成は下記テーブルにより決められます。構造体を提供しますが、ドライバーの実現は皆様から作成して貰いましょう。

Table 28–3. Register Map—32-bit Timer

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	(1)					RUN	TO
1	control	RW	(1)			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Note to Table 28–3:

(1) Reserved. Read values are undefined. Write zero.

構造体：

```
typedef struct{
   union{
      struct{
         volatile unsigned long int TO :1;
         volatile unsigned long int RUN :1;
         volatile unsigned long int NC :30;
      }BITS;
      volatile unsigned long int WORD;
   }STATUS;
   union{
      struct{
         volatile unsigned long int ITO :1;
         volatile unsigned long int CONT :1;
         volatile unsigned long int START:1;
         volatile unsigned long int STOP :1;
         volatile unsigned long int NC :28;
      }BITS; volatile unsigned long int WORD;
   }CONTROL;
   volatile unsigned long int PERIODL;
   volatile unsigned long int PERIODH;
   volatile unsigned long int SNAPL;
   volatile unsigned long int SNAPH;
}TIMER;
```

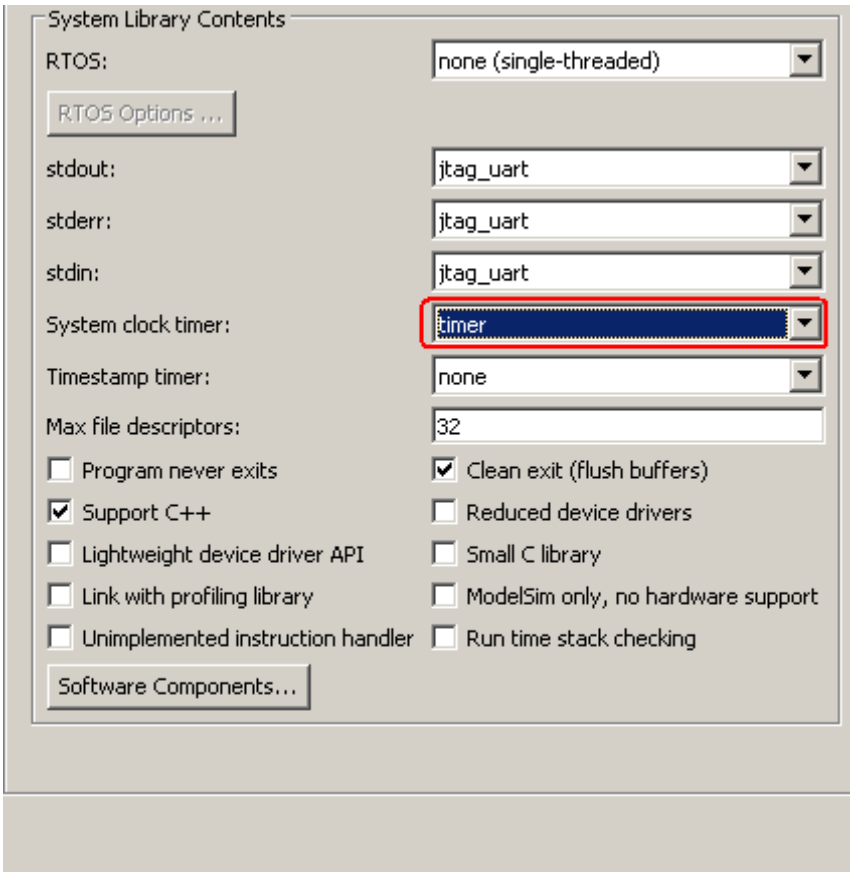
上記の構造体を使って HAL の API を使わなくて前章を参照して皆様がドライバーファイルを実現できます。ここまで同時二つタイマーを使う問題も解決しました。

■システムクロックの使用

ここシステムクロックを使ってタイマーの使い方も説明します。

この関数で 1 秒ごとに一つ LED を点灯し、４つ LED で繰り返します。まず、ソフトウエアから設定必要です。

プロジェクト「hello」を右クリック、「System Library Properties」をクリック



システムクロックを使うメイン関数：

```
/*
 *
===================================================================
=======
 *
 *      Filename:  main.c
 *
 *    Description:
 *
 *       Version:  1.0.1
```

```
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
========
 */

/*-----------------------------------------------------------------------------
---------
 *  Include

*-----------------------------------------------------------------------------
--------*/
#include <stdio.h>
#include <stdlib.h>
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
#include "../inc/sopc.h"
#include "sys/alt_alarm.h"

/*-----------------------------------------------------------------------------
---------
 *  Function prototypes

*-----------------------------------------------------------------------------
--------*/
alt_u32 my_alarm_callback(void *context);



/*-----------------------------------------------------------------------------
---------
```

```
 *  Variable
*-------------------------------------------------------------------------
--------*/
unsigned int i = 0;
unsigned int alarm_flag;

/*-------------------------------------------------------------------------
---------
 *  Define
*-------------------------------------------------------------------------
--------*/
#define INTEVAL_TICK 1
#define DEBUG

/*
 * ===  FUNCTION
======================================================================
 *         Name:  main
 *  Description:
 *
=========================================================================
========
 */
int main(void)
{
    //API関数を呼び出す用の変数
    alt_alarm alarm;

    printf("hello world!¥n");

    //システムクロックサービスを呼び出す
    if(alt_alarm_start(&alarm,INTEVAL_TICK,my_alarm_callback,NULL) < 0){
            #ifdef DEBUG
            printf("Error: No system clock available¥n");
```

```
            #endif
            exit(0);
    }
    else{
            #ifdef DEBUG
            printf("Success: System clock available¥n");
            #endif
    }

    while(1){
        if(alarm_flag != 0){
            LED->DATA = 1<<i;
            i++;

            if(i == 4)
                i = 0;

            alarm_flag = 0;
        }

    };

    return 0;
}


/* ===  FUNCTION
======================================================================
 *         Name:  my_alarm_callback
 *  Description:
 *
======================================================================
=====
 */
alt_u32 my_alarm_callback(void *context)
{
```

```
        alarm_flag = 1;
        //INTEVAL_TICKの値がタイマー割り込み関数に入る時間を決めます。
        return INTEVAL_TICK;
}
```

7.8 SDRAM 開発

本節から NIOS II で SDRAM の内容と使い方を説明します。

7.8.1 概要

NIOS システムの最も重要な外部デバイスとして、それが重要な役割を果たします。

電源オン度に、高速でプログラムを実行するため、FPGA は Flash のプログラムを SDRAM に送ってから実行します、但し、電源オフ時データが失ったので、データがフラッシュにを保存必要です。SDRAM の理論知識を紹介しませんが、Nios II 開発プロセスで SDRAM の理論知識を理解していない場合でも、使用上にも問題がありません。なぜなら、SOPC Builder には SDRAM を完璧に駆動していますからです。我々はそれを使用する方法を知っていれば良いです。ここでは、SDRAM の使い方を説明します、それは本当に非常に単純です。

「ハードウエア開発」という章にも SDRAM モジュールを作成しましたので、本節はハードウエアの開発を省略します。

7.8.2 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、まず、SDRAM に関する内容（ベースアドレスと IRQ）を system.h ファイルにあるかどうかを確認します。

```
/*
 * sdram configuration
 *
 */

#define SDRAM_NAME "/dev/sdram"
#define SDRAM_TYPE "altera_avalon_new_sdram_controller"
#define SDRAM_BASE 0x01000000
……
```

次はプログラムを修正していきましょう。

１．メイン関数修正

```
/*
```

```
 *
================================================================================
================
 *
 *      Filename:  main.c
 *
 *    Description:  SDRAM実例
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
================
 */

/*-----------------------------------------------------------------------------
----------
 * Include

*----------------------------------------------------------------------------
---------*/
#include <stdio.h>
#include "../inc/sopc.h"
#include "system.h"
#include "string.h"

/*-----------------------------------------------------------------------------
----------
 *  Variable

*----------------------------------------------------------------------------
---------*/
```

```
unsigned short * ram = (unsigned short *)(SDRAM_BASE+0x10000); //SDRAM
アドレス

/*
 * ===  FUNCTION
======================================================================
=
 *         Name:  main
 *  Description:  メイン関数
 *
======================================================================
================
 */
int main(void)
{
    int i;

    memset(ram,0,100);

    //ramにデータを書き込み、書込み完了後、ramのアドレスは(SDRAM_BASE+0x10100)
に変わります。
    for(i=0;i<100;i++){
        *(ram++) = i;
    }

    //逆にramからデータを読み込み
    for(i=0;i<100;i++){
        printf("%d¥n",*(--ram));
    }

    return 0;
}
```

プログラムが簡単です、特定の SDRAM に値を付けます。

注意点：

■unsigned short タイプのポインタ変数 ram を定義します、そして、SDRAM+0x10000 に初期化しています。理由は SDRAM は 16 桁のバスを使います、初期化値が「SDRAM+0x10000」を設定するのは NIOS II を実行時一部 SDRAM を使うためです。 この NIOS II 使う容量を使わなくて成功にコンパイルを確保します。「0x10000」この値は固定ではなく、NIOS II が使う容量を避ければよいです。

■SDRAM に値に付けた後、ポインタは後ろに移動して次のアドレスにポイントします。プラス度に、アドレスが後ろに 16 桁を移動します、例えば、カレントが「SDRAM+0X10000」にポイントされ、一回移動したら、アドレスが「SDRAM+0X10002」に変わります、さらに移動すれば、「SDRAM+0X10004」に変わります。これらの移動処理は内部で自動処理しますので、原理を理解のうえ、我々から操作必要ありません。

開発ボードには 64MbitSDRAM があります、数少ない部分は NIOS システムをあげ、多くのスペースは空いています。但し、以下の場合、64MbitSDRAM を利用できます。

C 言語で大きなサイズのデータを受信する時、ヒープを使います。ヒープとスタックを間違い安いかも知れませんが、スタックはシステムから自動割り当てられます、例えば、関数のローカル変数 int b を宣言する時、システムから関数 b のためスペースを割り当てます。逆にヒープはプログラマから申請かつ大きさを設定必要です。

下記のソースはヒープのサンプルです。

```
/* * === FUNCTION ====================================== *
    Name: main *
    Description: メイン関数
* ================================================== */
int main()
{
    char *ram = (char *)malloc ( sizeof(char)* 500000 );
    if ( ram==NULL ){
        fprintf ( stderr, "¥ndynamic memory allocation failed¥n" );
        exit (EXIT_FAILURE);
    }
    uart.init();
    /*------------------ FLASH ----------------------------*/
    while(1){
        if(uart.mode_flag){
            xmodem.xmodem_rx();
            printf("ram_cnt:%d¥n",ram_cnt);
            parse_srecord_buf(ram,ram_cnt);
            printf("successful!");
```

```
            ram_cnt = 0;
            uart.mode_flag = 0;
        }
    }
    free (ram);
    return 0;
}
```

上記二つ SDRAM の使い方を説明しました、皆様が一つ質問があるかもしれません、なぜスタックを使わないですか？これはコンパイルと関係があります、コンパイル時に、スタックに必要の容量もコードに加えます、即ち、大きなサイズのデータを処理する時、もし、スタックを使用すると、コンパイル後のコードが大きくなります、Flash にダウンロードした後、SDRAM にロードする時間も長くかかります。逆に、ヒープを使う場合、この状況を起こさないです、ヒープを使う時に割り当てますからです。

7.9 Flash プログラム

本節から NIOS II で Flash プログラミングに関数内容、例えば、パラレル Flash と EPCSX シリーズ、を説明します。

7.9.1 概要

Flash の特徴は読み込み安いですが、書き込みにくいです。一般に、Flash から内容を直接読み込むことが出来ます、書き込む時、多くのコマンドを発行必要です、例えば、555、AA、2AA、55、555、A0、PA。PD は PA にデータ「PD」を書き込みます。NIOS II 開発プロセスの中、Flash の煩雑のクリア処理は Altera 社から実現しますので、Altera 社が提供している API 関数を呼び出して Flash の読み込み・書込みを完成できます。これらの操作の前提は Flash が CFI 標準あるいは EPCSX シリアル Flash（EPCSX と相性がある Flash でも OK）に満たすことです、満たさない場合、皆様が自身で処理必要です。

「ハードウエア開発」という章にも Flash と EPCS モジュールを作成しましたので、本節はハードウエアの開発を省略します。

7.9.2 ソフトウエア開発

CFI Flash と EPCS のソフトウエア操作方法は同じです、Altera 社は二種類関数を提供しています、Simple Flash Access（簡単な Flash アクセス）と Fine-Grained Flash Access（細かい Flash アクセス）があります。

Fine-Grained Flash Access とういう種類関数をお勧めします、Simple Flash Access よりあまり複雑ではないですが、通常のブロック間の消去問題を回避できます。Flash はブロックで構成されます、クリアすれば、ブロック全体をクリアします。

もし Flash に書き込む時、二つブロックが跨る場合、特にの二つブロックの後ろにデータがあれば、二つブロックをクリアすると、データも同時クリアされた可能性があります、そうすると、データが失ったことを招きます。

本節は良く使われる関数を紹介します、詳しくは Altera 社の資料を参照してください。

①C 言語でハードデスクのデータの操作と同じ様に、使う前オープン必要です。

まず、alt_flash_open_dev()関数を使って Flash をオープンします、正常にオープンするとハンドルを返します。下記ソースはこの関数を使う時一部です。

```
    alt_flash_fd *fd;
    //Flashデバイスをオープン
    fd = alt_flash_open_dev(EPCS_NAME);
```

EPCS_NAME は system.h ファイルにあります。

```
#define EPCS_NAME "/dev/epcs"
```

データを読み込む関数

```
int alt_read_flash( alt_flash_fd* fd,
                    int offset,
                    void* dest_addr,
                    int length ) ;
```

操作完了後、Flash をクローズ必要です。クローズ関数のプロタイプ：

```
void alt_flash_close_dev(alt_flash_fd* fd ) ;
```

Fine-Grained Flash Access には以下の関数もあります。

alt_get_flash_info()

alt_erase_flash_block()

alt_write_flash_block()

alt_get_flash_info()は Flash に関する情報を取得できます、例えば、いくつかゾーン、ゾーンごとにいくつかブロック、ブロックの大きさ等、関数のプロタイプは下記通りです。

```
int ret_code = 0;
int number_of_regions=0;
flash_region* regions;
ret_code = alt_get_flash_info(fd, &regions, &number_of_regions);
```

ここ flash_region とういう構造体を使われます。flash_region のプロタイプは下記です。

```
typedef struct flash_region {
        int offset; /* region が Flash のベースアドレスによりオフセット*/
        int region_size; /* region のサイズ*/
        int number_of_blocks; /* region ごとのブロックの数*/
        int block_size;/* block のサイズ*/
}flash_region;
```

ブロックを 1 個クリアする時、alt_flash_fd を使います、プロタイプは下記です。

```
int alt_erase_flash_block( alt_flash_fd* fd, int offset, int length) ;
```

データを 1 ブロック書き込む時、alt_write_flash_block を使います、プロタイプは下記です。

```
int alt_write_flash_block( alt_flash_fd* fd,
                           int block_offset,
                           int data_offset,
                           const void *data,
                           int length);
```

上記関数は良く使われるものです、それでは、これら関数を使って Flash を操作しましょう。(NIOS II IDE でメイン関数を修正)

```
/*
 *
===================================================================
===============
 *
 *      Filename:  main.c
 *
 *    Description:  Flashプログラム実例
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
===================================================================
===============
 */

/*-------------------------------------------------------------------
---------
 * Include

*-------------------------------------------------------------------
--------*/
#include <stdio.h>
```

```
#include <string.h>
#include "system.h"
#include "sys/alt_flash.h"

/*---------------------------------------------------------------------
---------
 *  Define

*---------------------------------------------------------------------
--------*/
#define BUF_SIZE 100
#define FLASH_OFFSET 0x50000

/*
 * ===  FUNCTION
=====================================================================
 *         Name:  main
 *  Description:  メイン関数
 *
=====================================================================
===============
 */
int main(void)
{
    alt_flash_fd *fd;
    int i;
    int ret_code;
    char source[BUF_SIZE];
    char dest[BUF_SIZE];

    for(i=0;i<100;i++){
        source[i] = i;
    }

    //Flashデバイスをオープン
    fd = alt_flash_open_dev(EPCS_NAME);
```

```
if(fd == NULL){
    printf("Can't open flash device¥n");
    exit(-1);
}
else{
    printf("Open Flash Device Successfully.¥n");
}

//FLASHにデータを書き込み、成功の場合、0を返す
ret_code = alt_write_flash(fd,FLASH_OFFSET,source,BUF_SIZE);

if(ret_code != 0){
    printf("Can't write flash device¥n");
    exit(-1);
}
else{
    printf("Write Flash Device Successfully.¥n");
}

//FLASHからデータを読み込み、成功の場合、0を返す
ret_code = alt_read_flash(fd,FLASH_OFFSET,dest,BUF_SIZE);

if(ret_code != 0){
    printf("Can't read flash device¥n");
    exit(-1);
}
else{
    printf("Read Flash Device Successfully.¥n");
}

//読み込んだデータをプリンタ
for(i=0;i<100;i++){
    printf("%d¥n",dest[i]);
}
```

```
    alt_flash_close_dev(fd);

    return 0;
}
```

上記プログラムは EPCS を操作しますが、CFI Flash を操作する場合、alt_flash_open_dev(EPCS_NAME)のパラメータから"/dev/cfi_flash"に変更してよいです。

7.10 AVALON

本節から NIOS II システムでの AVALON IP カスタマイズの方法を説明します。

7.10.1 概要

NIOS II は FPGA に基づき作成した組み込み IP マクロです、ニーズに応じ周辺デバイスを追加出来る以外、ユーザーロジックデバイスとユーザーコマンドをカスタマイズしてさまざまアプリのニーズを満たす事が出来ます。本節は Avalon バスに基づきユーザーデバイスをどのようにカスタマイズするかを説明します。

SOPC Builder には電気部品エディタがあります、このエディタでカスタマイズしたロジックデバイスをパッケージして SOPC Builder の一つ電気部品として使えます。

では、PWM を例として説明して行きます。

これから説明しよう PWM は Avalon バスの Avalon Memory Mapped Interface (Avalon-MM)に基づき作成するものです、Avalon バスには他の種類のデバイスもあります、例えば、Avalon Streaming Interface (Avalon-ST)、Avalon Memory Mapped Tristate Interface 等、

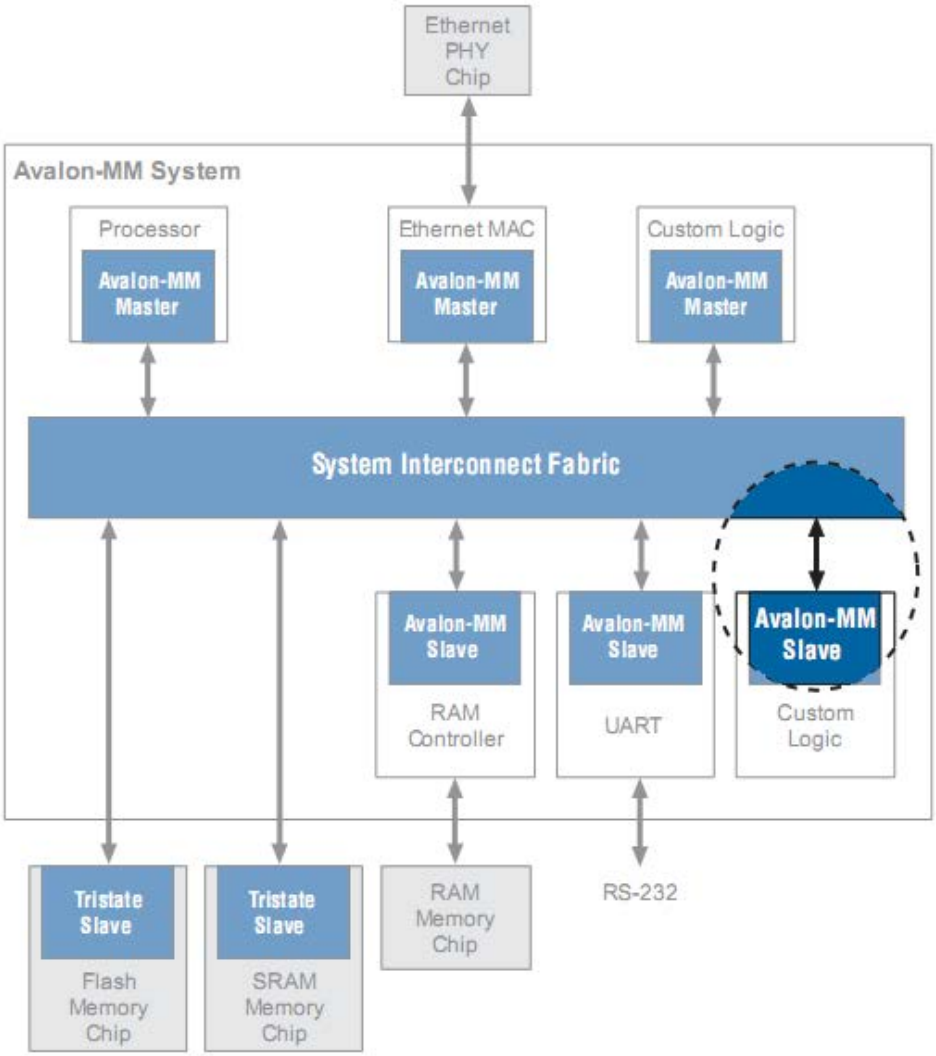
詳しくは下記 Altera 社の資料を参照してください。

《Avalon Interface Specifications》

http://www.altera.com/literature/manual/mnl_avalon_spec_1_3.pdf

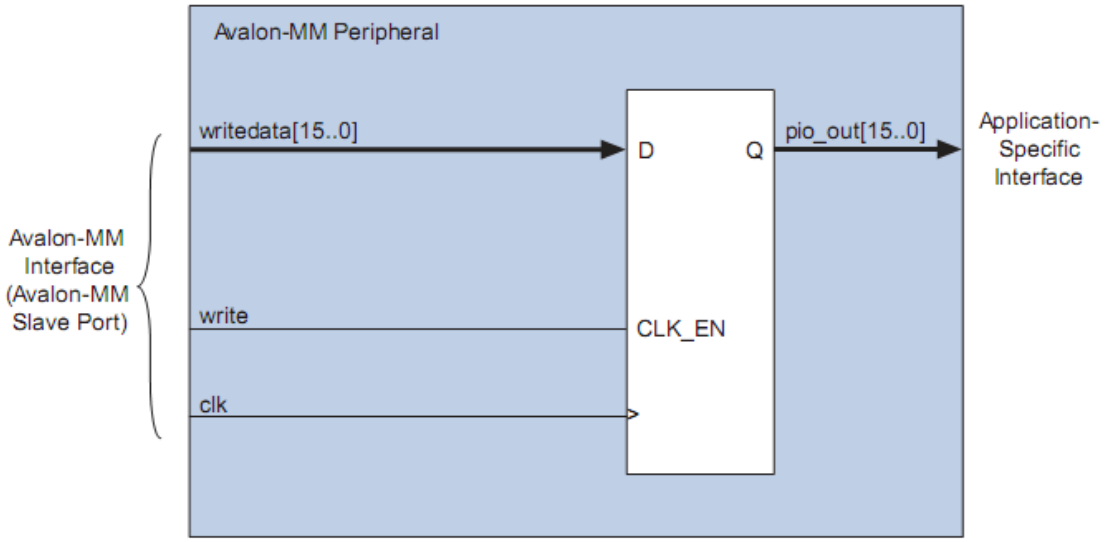
Avalon-MM インタフェースはメモリマッピングシステムにあるマスタースレーブデバイス間のライトリード操作のインタフェースとなります、下図は Avalon-MM に基づきマスタースレーブデバイスシステムを示します。

本節で実現内容は下図のハイライトの部分です、UART、RAM Controller 等のモジュールと同じ位置を付けます。



Avalon-MM インタフェースは沢山の特徴があります、一番の特徴はニーズにより信号ラインを選択できます。勿論、ある程度条件があります、本節を読む前、《Avalon Interface Specifications》を一読するのをお勧めします。

下図は Avalon-MM 周辺デバイスのチャートです。



理論知識の紹介はここまでです、次は実例を説明します。

7.10.2 DHL モジュール設計

PWM を例としますから、まず、Avalon-MM Slave インタフェース仕様に準拠する PWM 機能を実現した時相ロジックを作成します。ここでは Verilog 言語でプログラミングします、プログラムの中 Avalon 信号に関わります、信号の意味は下記テーブルで示します。

※Verilog は専門資料をご参照ください。

HDL の信号	Avalon 信号種類	幅	方向	説明
clk	clk	1	input	同期クロック信号
reset_n	reset_n	1	input	リセット信号、低電位で有効
chipselect	chipselect	1	input	チップ･セレクト信号
address	address	2	input	2 桁アドレス、デ―コード後レジスタオフセットを確定
write	write	1	input	書込み有効信号
writedata	writedata	32	input	32 桁書込み用のデータ
read	read	1	input	読み込み有効信号
byteenable	byteenable	1	input	バイト有効信号
readdata	readdata	32	input	32 桁読み込み用のデータ

他に、このプログラムには PWM_out 信号を含まれます、この信号は PWM から出力します、Avalon 信号に属しません。

PWM 内部は有効コントロールレジスタ、サイクル設定レジスタとデューティ･サイクル設定レジスタを含みます。各レジスタを Avalon Slave ポートアドレスの中の単一のオフセットアドレスにマッピングします。すべてレジスタは読み込み・書込み操作を行えます、ソフトウェアではレジスタのカレント値を読み込むことが出来ます。

レジスタとオフセットは以下通りです。

レジスタ名	オフセット	アクセス属性	説明
clock_divide_reg	00	R/W	PWM 出力サイクルのクロック数を設定
duty_cycle_reg	01	R/W	一つサイクル内の PWM 低電位出力の数
control_reg	10	R/W	PWM 出力をクローズと有効、1 が有効

Verilog プログラムは下記です。

※Verilog プログラムファイル「PWM.v」を作成してから Quartus II プロジェクト「SOPC_T.qpf」の所属フォルダーに「pwm」フォルダを作成してから置いてください。

```
module PWM(
       reset_n,
       clk,
       chipselect,
       address,
       write,
       writedata,
       read,
       byteenable,
       readdata,
       PWM_out);

input clk;
input reset_n;
input chipselect;
input [1:0]address;
input write;
input [31:0] writedata;
input read;
input [3:0] byteenable;
output [31:0] readdata;
output PWM_out;

reg [31:0] clock_divide_reg; //Internal register  clock divide
reg [31:0] duty_cycle_reg; //Internal register ;the clock less than
duty_cycle_reg pwm_out will be output 1.otherwise will be 0
reg control_reg;
reg clock_divide_reg_selected;
```

```
reg duty_cycle_reg_selected;
reg control_reg_selected;
reg [31:0] PWM_counter;
reg [31:0] readdata;
reg PWM_out;
wire pwm_enable;

//アドレスデ–コード
always @ (address)
begin
	clock_divide_reg_selected<=0;
	duty_cycle_reg_selected<=0;
	control_reg_selected<=0;
	case(address)
		2'b00:clock_divide_reg_selected<=1;
		2'b01:duty_cycle_reg_selected<=1;
		2'b10:control_reg_selected<=1;
		default:
		begin
			clock_divide_reg_selected<=0;
			duty_cycle_reg_selected<=0;
			control_reg_selected<=0;
		end
	endcase
end

//PWM出力サイクルのクロックレジスタを書き込む
always @ (posedge clk or negedge reset_n)
begin
	if(reset_n==1'b0)
		clock_divide_reg=0;
	else
	begin
		if(write & chipselect & clock_divide_reg_selected)
		begin
			if(byteenable[0])
```

```
                        clock_divide_reg[7:0]=writedata[7:0];
                if(byteenable[1])
                        clock_divide_reg[15:8]=writedata[15:8];
                if(byteenable[2])
                        clock_divide_reg[23:16]=writedata[23:16];
                if(byteenable[3])
                        clock_divide_reg[31:24]=writedata[31:24];
        end
    end
end

//PWMデューティ□Eサイクルレジスタを書き込む
always @ (posedge clk or negedge reset_n)
begin
    if(reset_n==1'b0)
        duty_cycle_reg=0;
    else
    begin
        if(write & chipselect & duty_cycle_reg_selected)
        begin
            if(byteenable[0])
                duty_cycle_reg[7:0]=writedata[7:0];
            if(byteenable[1])
                duty_cycle_reg[15:8]=writedata[15:8];
            if(byteenable[2])
                duty_cycle_reg[23:16]=writedata[23:16];
            if(byteenable[3])
                duty_cycle_reg[31:24]=writedata[31:24];
        end
    end
end

//コントロールレジスタを書き込む
always @ (posedge clk or negedge reset_n)
begin
    if(reset_n==1'b0)
```

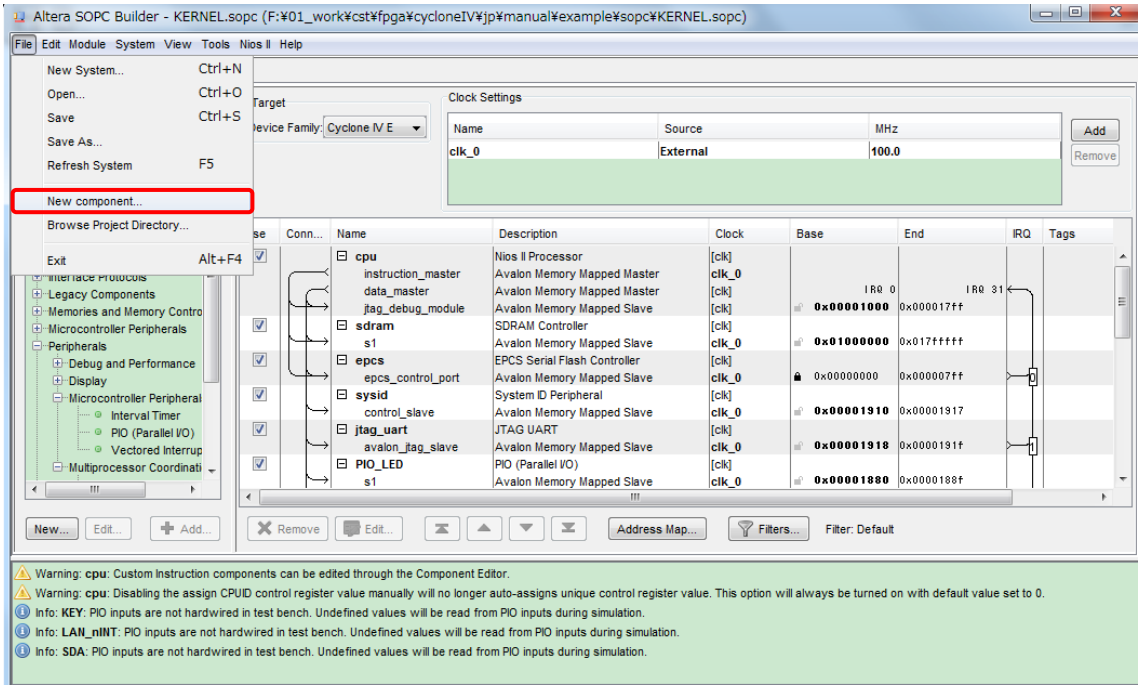
```
                control_reg=0;
        else
        begin
                if(write & chipselect & control_reg_selected)
                begin
                        if(byteenable[0])
                                control_reg=writedata[0];
                end
        end
end

//レジスタを読み込む
always @ (address or read or clock_divide_reg or duty_cycle_reg or
control_reg or chipselect)
begin
        if(read & chipselect)
                case(address)
                        2'b00:readdata<=clock_divide_reg;
                        2'b01:readdata<=duty_cycle_reg;
                        2'b10:readdata<=control_reg;
                        default:readdata=32'h8888;
                endcase
end

//コントロールレジスタ
assign pwm_enable=control_reg;

always @ (posedge clk or negedge reset_n)
begin
                if(reset_n==1'b0)
                        PWM_counter=0;
                else
                begin
                        if(pwm_enable)
                        begin
                                if(PWM_counter>=clock_divide_reg)
```

```
                                        PWM_counter<=0;
                                end
                                        PWM_counter<=PWM_counter+1;
                        end
                        else
                                PWM_counter<=0;
                end
end

//PWM機能部分
always @ (posedge clk or negedge reset_n)
begin
                if(reset_n==1'b0)
                        PWM_out<=1'b0;
                else
                begin
                        if(pwm_enable)
                        begin
                                if(PWM_counter<=duty_cycle_reg)
                                        PWM_out<=1'b1;
                                else
                                        PWM_out<=1'b0;
                        end
                        else
                                PWM_out<=1'b0;
                end
end

endmodule
```

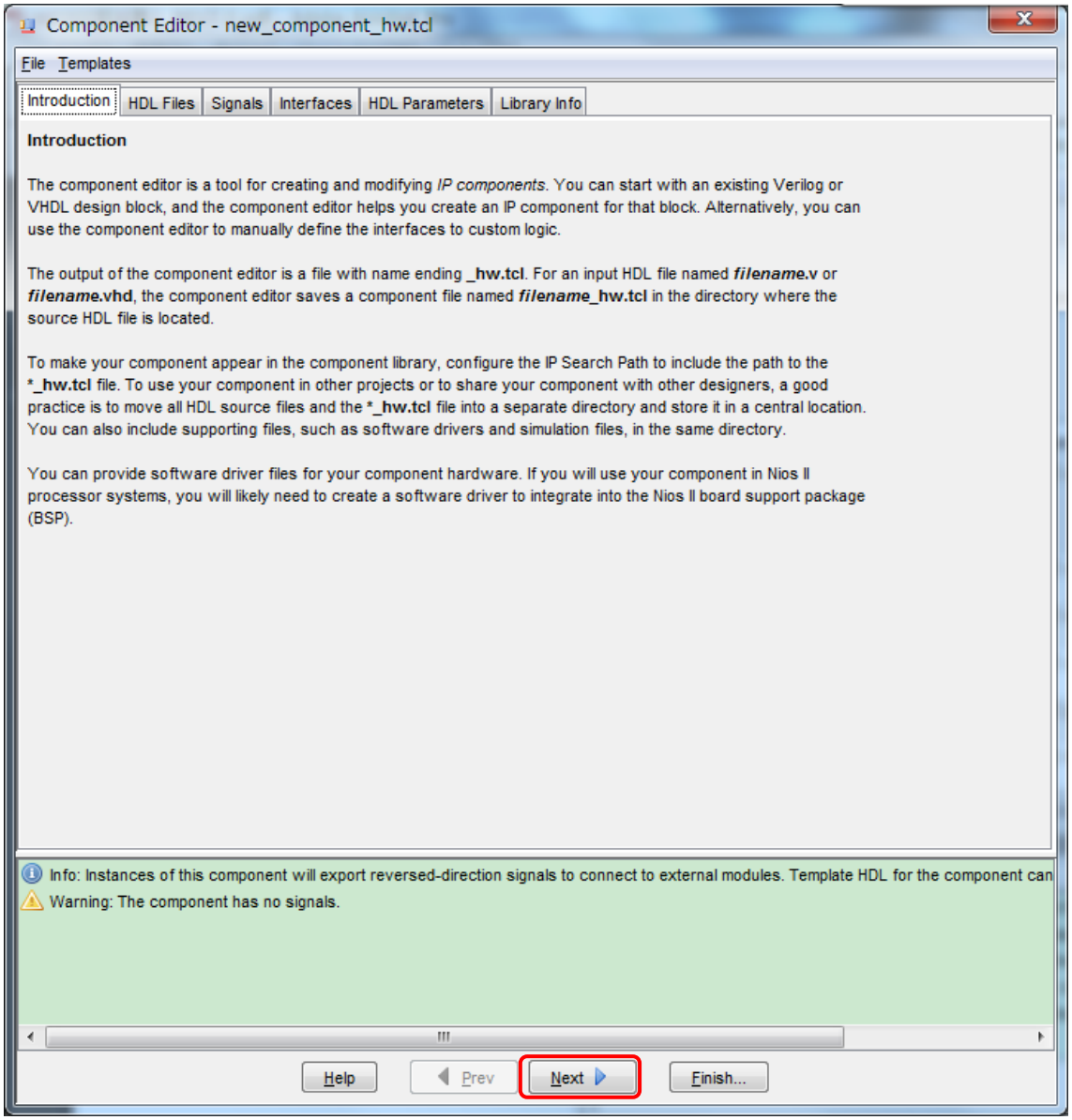
7.10.3 ハードウエア開発

Quartus II でプロジェクト「SOPC_T.qpf」を開き、SOPC Builder を起動させてから PWM モジュールを作成します。

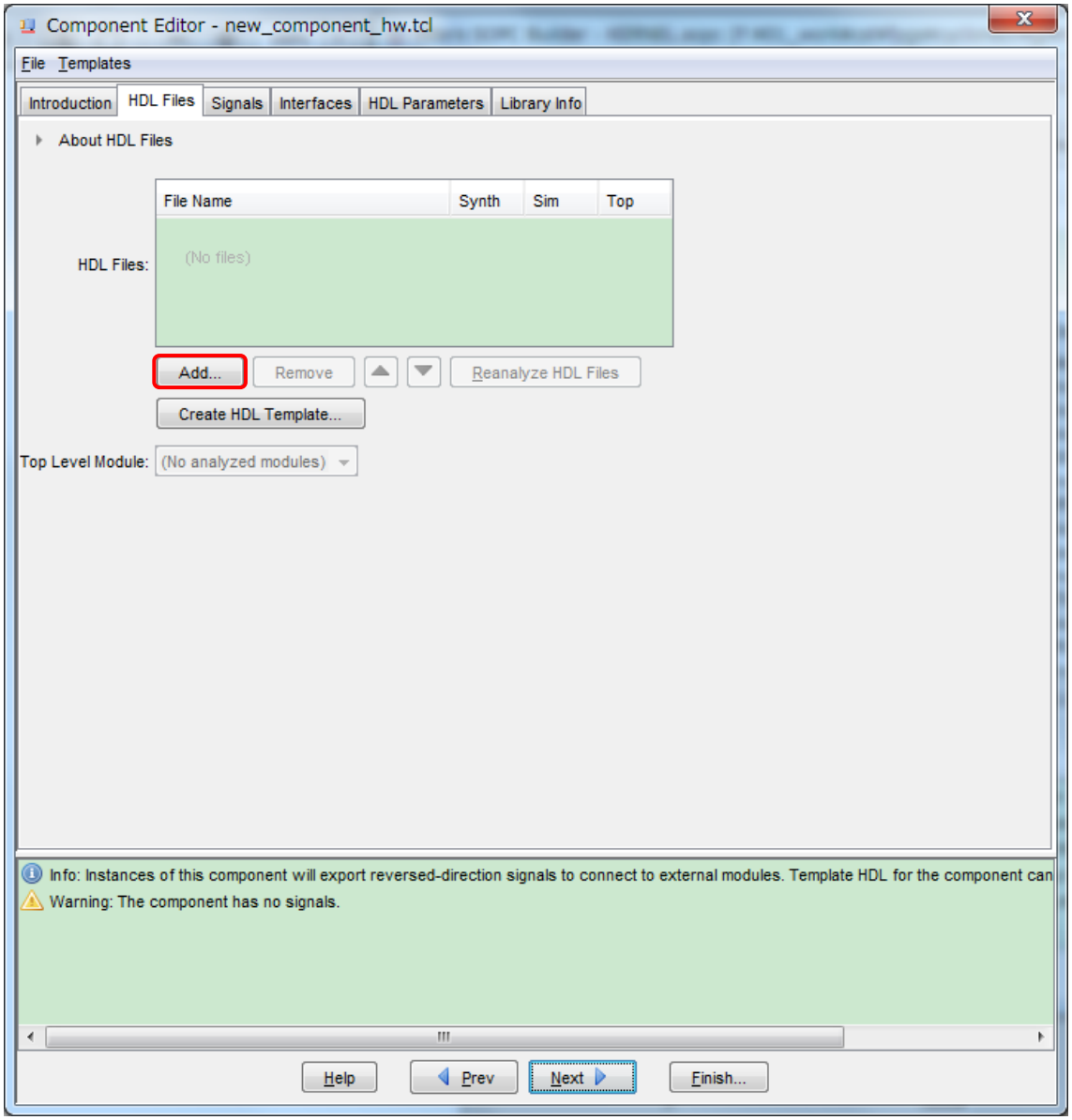
SOPC Builder 画面でメニュー「File」→「New component」をクリック



そのまま「Next」ボタンをクリック

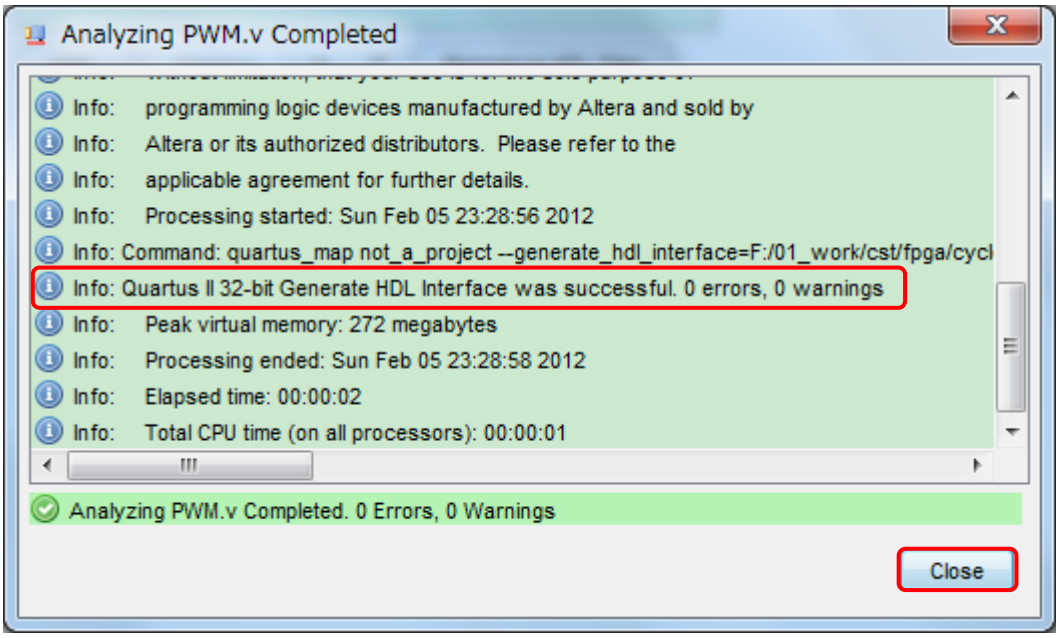


「Add」ボタンを押し先程作った Verilog プログラムファイル「PWM.v」をインポートします。



インポート後、Quartus II は自動に分析します、下図のメッセージが出てれば、成功に分析したことになります。

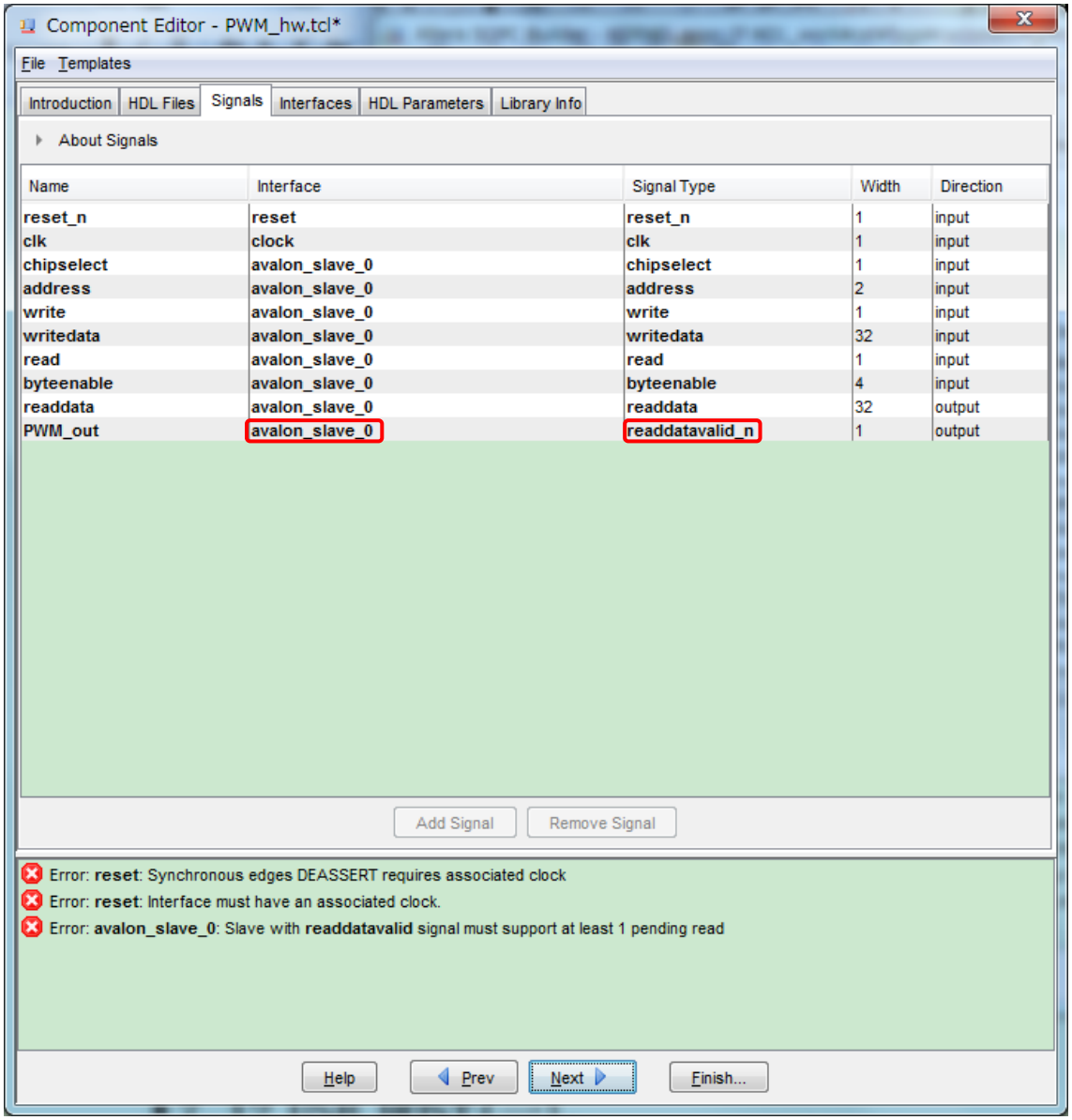
成功分析の後、「Close」ボタンを押す

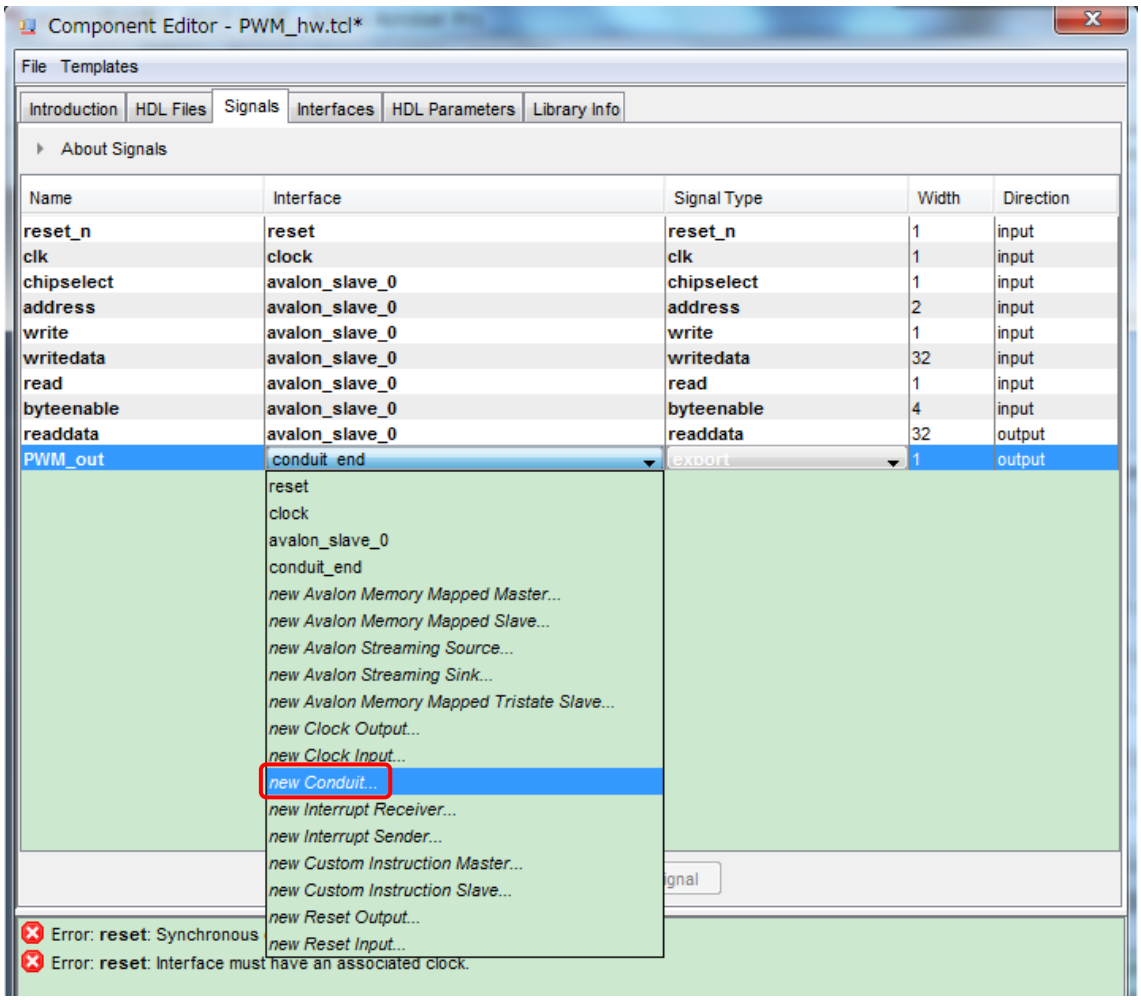


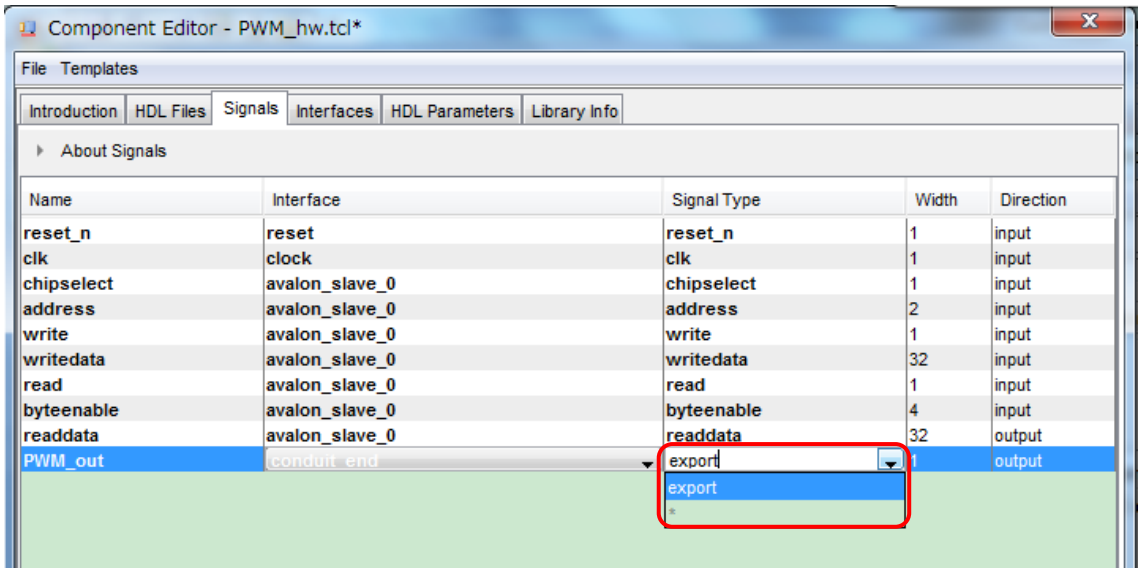
「Component Editor」画面に戻ったら、「Next」ボタンを押す

下記のような画面が出ます、画面内容により「PWM.v」ファイルの信号は全て表示されます。ニーズによりこれらの信号を設定できます、「Interface」は Avalon インタフェースです、Avalon-MM、Avalon-ST、Avalon Memory Mapped Tristate Interface 等があります。「Signal Type」は Avalon インタフェースの中の信号種類です、PWM.v の中の信号はこの前に説明しました、デフォルトで「PWM_out」のみが修正必要です。

下記赤□をそれぞれクリックしてプルダウンリストが表示されます、表示されたら、「new Conduit」、「export」を選択します。

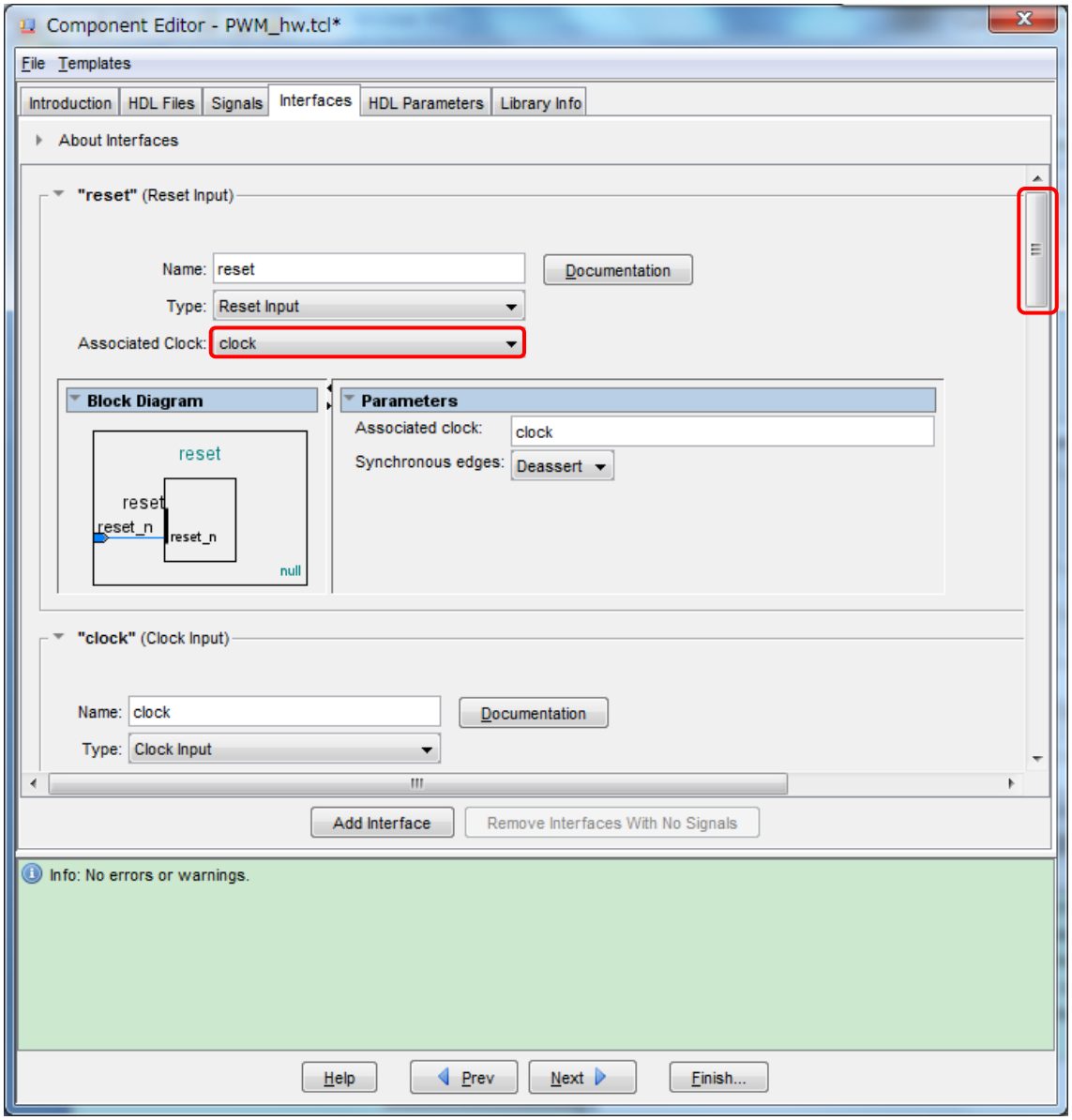

Component Editor - PWM_hw.tcl*
File Templates
Introduction HDL Files Signals Interfaces HDL Parameters Library Info
About Signals
Name Interface Signal Type Width Direction
reset_n reset reset_n 1 input
clk clock clk 1 input
chipselect avalon_slave_0 chipselect 1 input
address avalon_slave_0 address 2 input
write avalon_slave_0 write 1 input
writedata avalon_slave_0 writedata 32 input
read avalon_slave_0 read 1 input
byteenable avalon_slave_0 byteenable 4 input
readdata avalon_slave_0 readdata 32 output
PWM_out avalon_slave_0 readdatavalid_n 1 output
Add Signal
Remove Signal
Error: reset: Synchronous edges DEASSERT requires associated clock
Error: reset: Interface must have an associated clock.
Error: avalon_slave_0: Slave with readdatavalid signal must support at least 1 pending read
Help
Prev
Next
Finish...



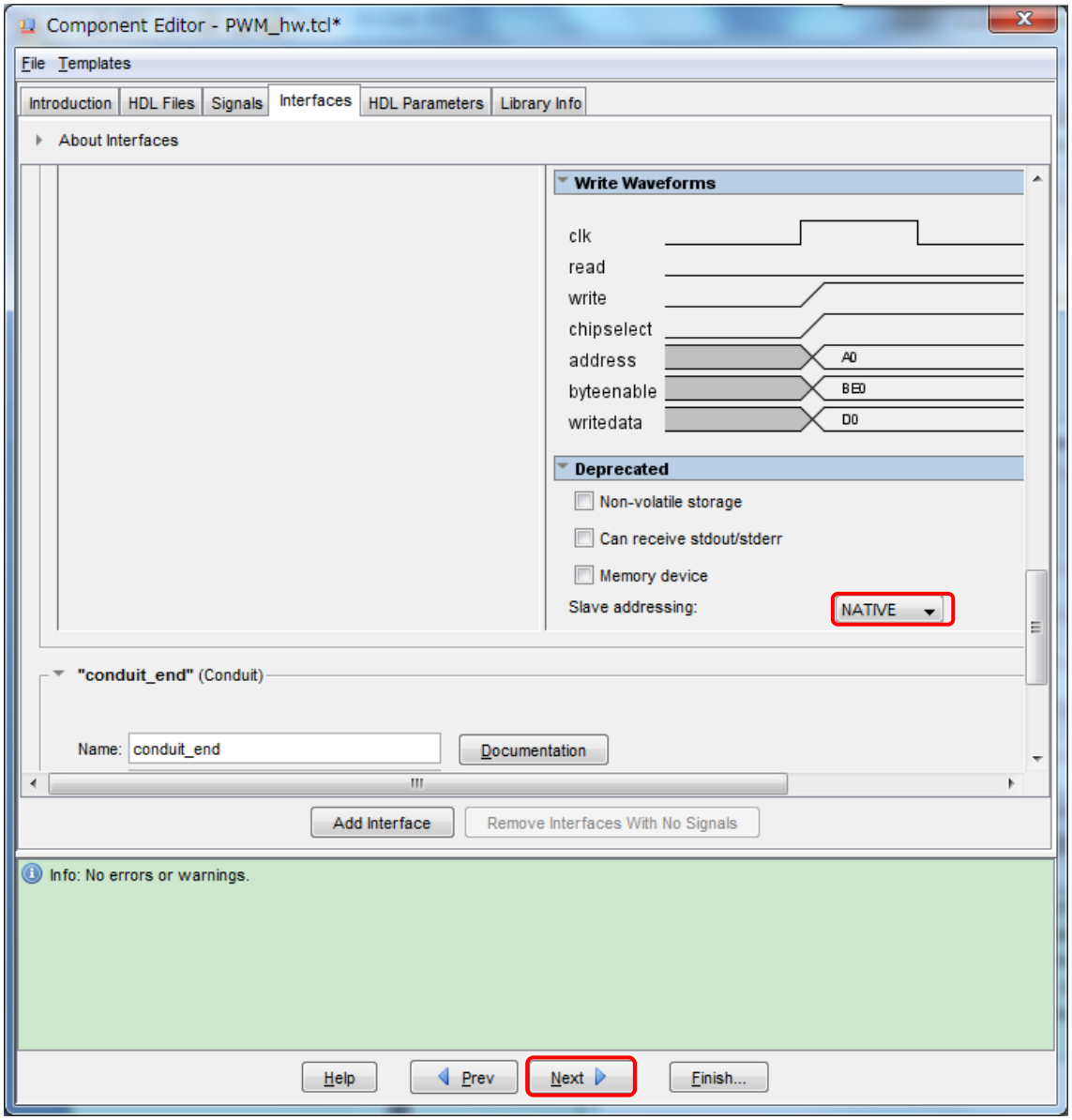


設定してから「Next」ボタンを押して次の画面が出ます、まず、reset 信号にクロックと関係を付けてください。(clock を選択)、それから、ドロップダウンしてください。

※関連のクロックを付けたら、エラーがなくなった。



下図のように「NATIVE」を選択

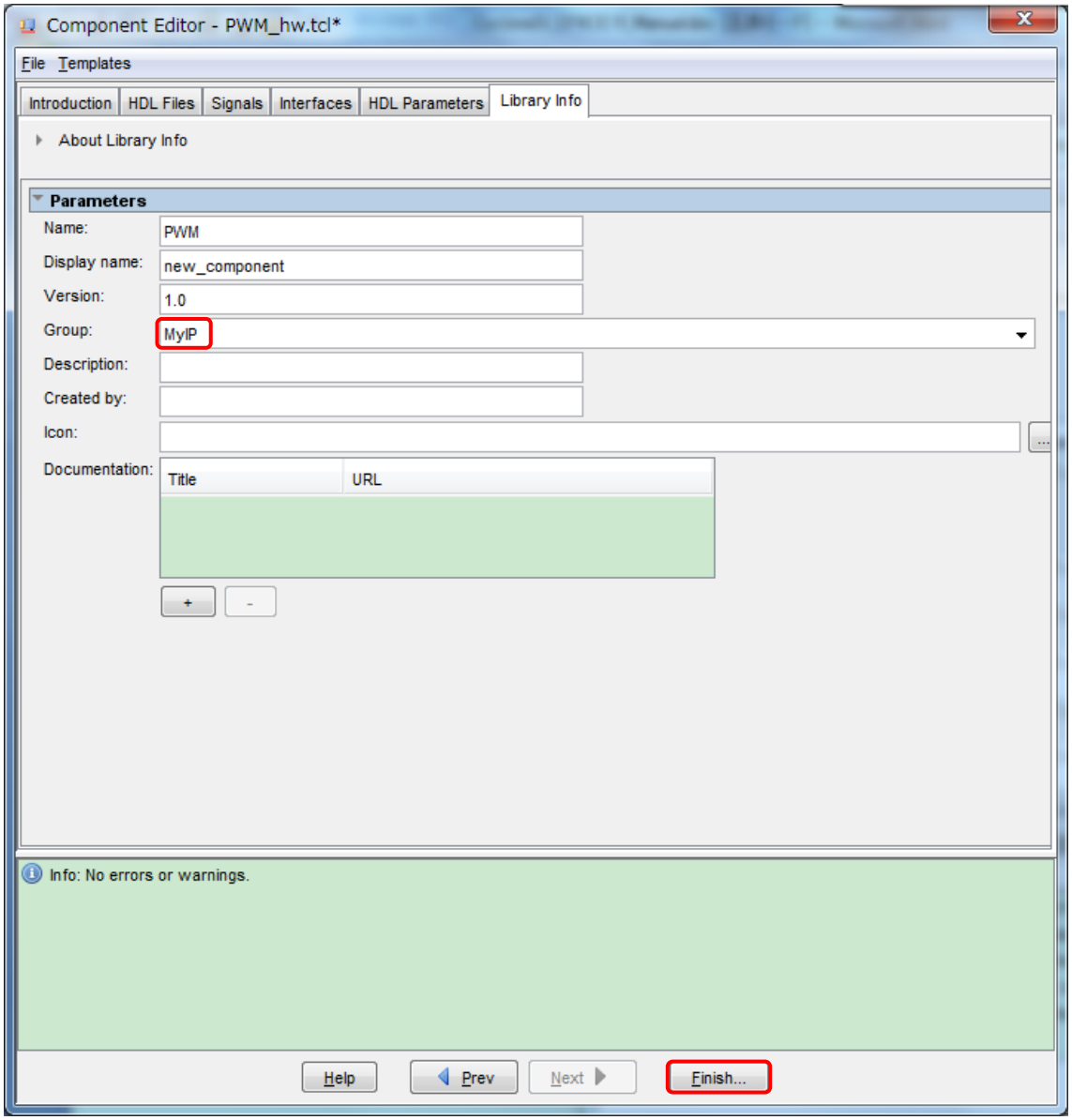


他に沢山のオプションがあります、Timing 部分を少し説明します。PWM の Avalon Slave ポートが Avalon Slave ポートクロック信号と同期します､R/W 時の作成時間が 0 にします、R/W レジスタは一つサイクルのみが掛りますから、R/W はノーウエイトになり、読み込み遅延も必要がありません。

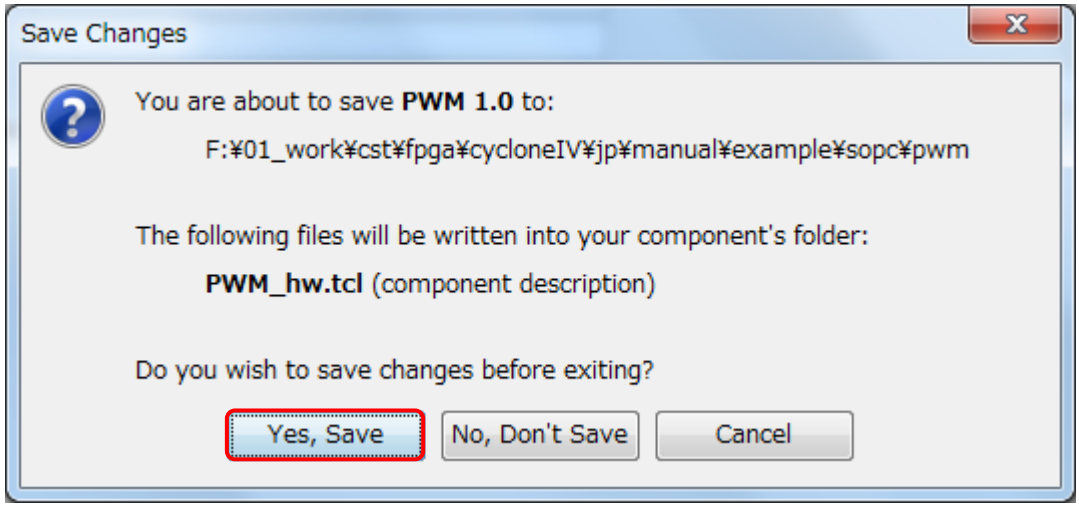
上図に設定が終わってから「Next」ボタンをクリック

次の画面が出ます、この画面で「Group」に「MyIP」を入力してから「Finish」ボタンをクリック

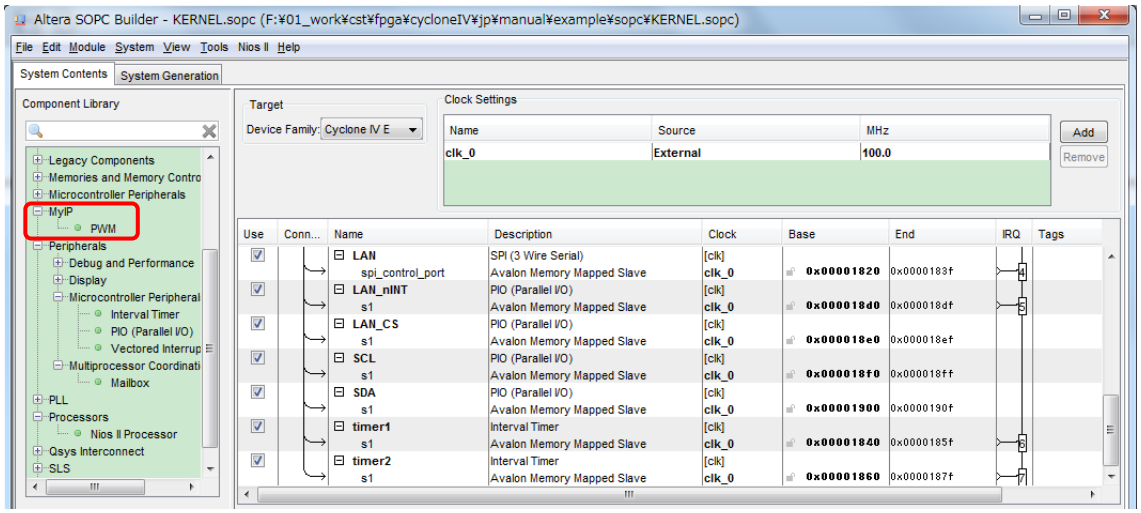
※Group を入力したら、SOPC Builder に反映できます。



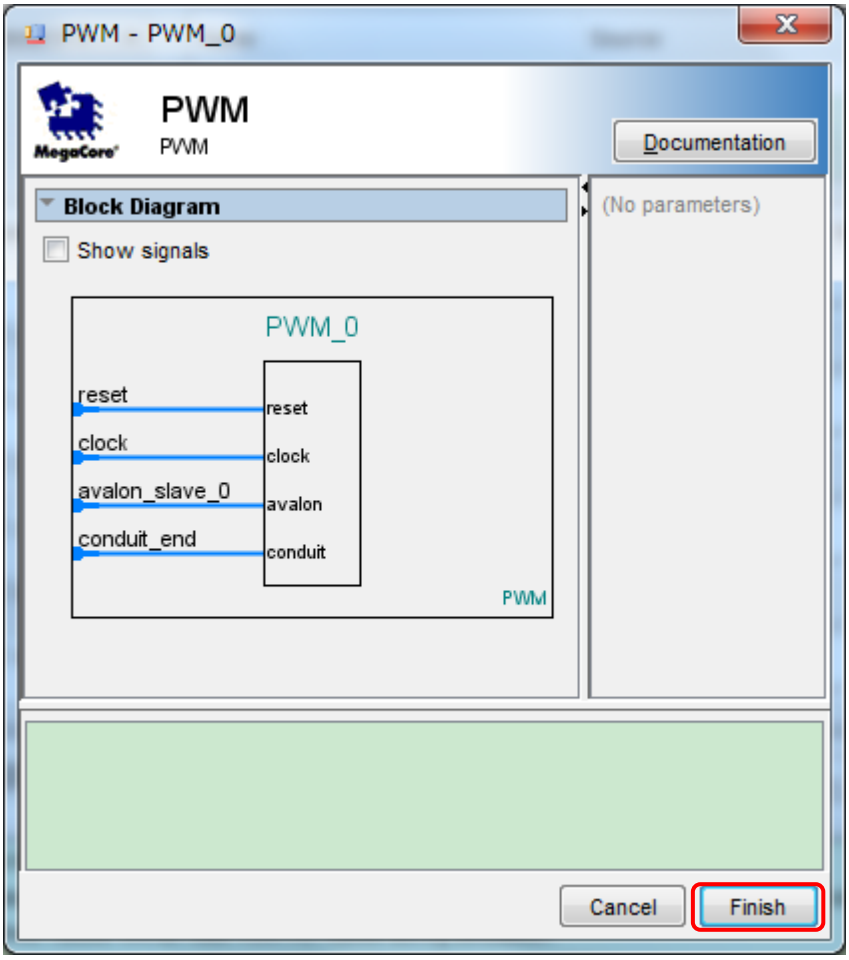
次に「PWM_hw.tcl」とういうファイルが生成される画面が出ます。「Yes,Save」をクリックします。



SOPC Builder メイン画面に戻ります、左側に先程作った「MyIP」が表示されますね。

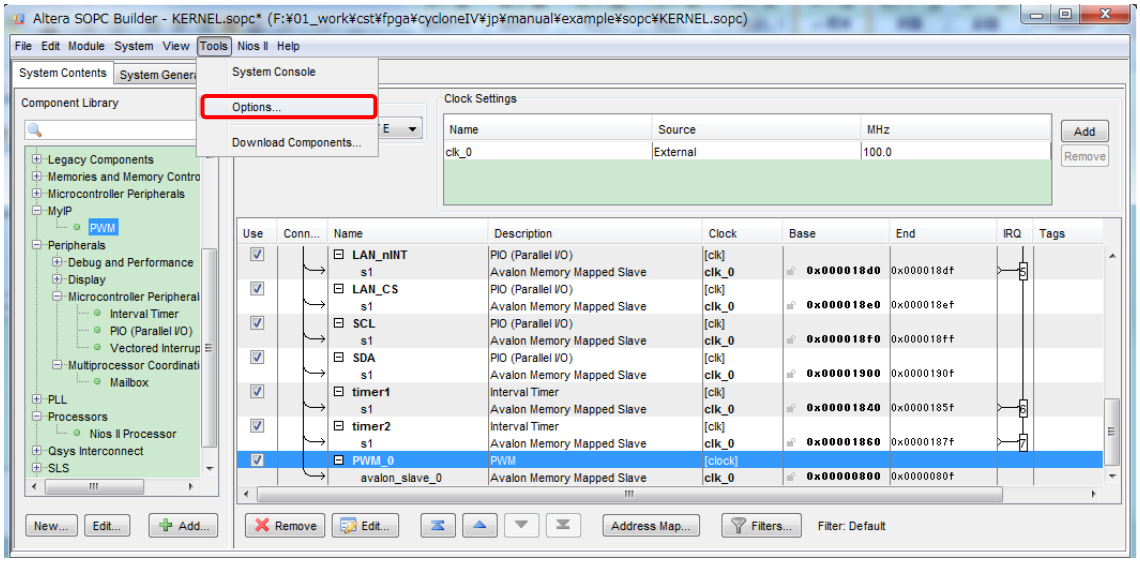


上図の PWM をダブルクリックすると、下記の PWM モジュール作成画面が出ます。

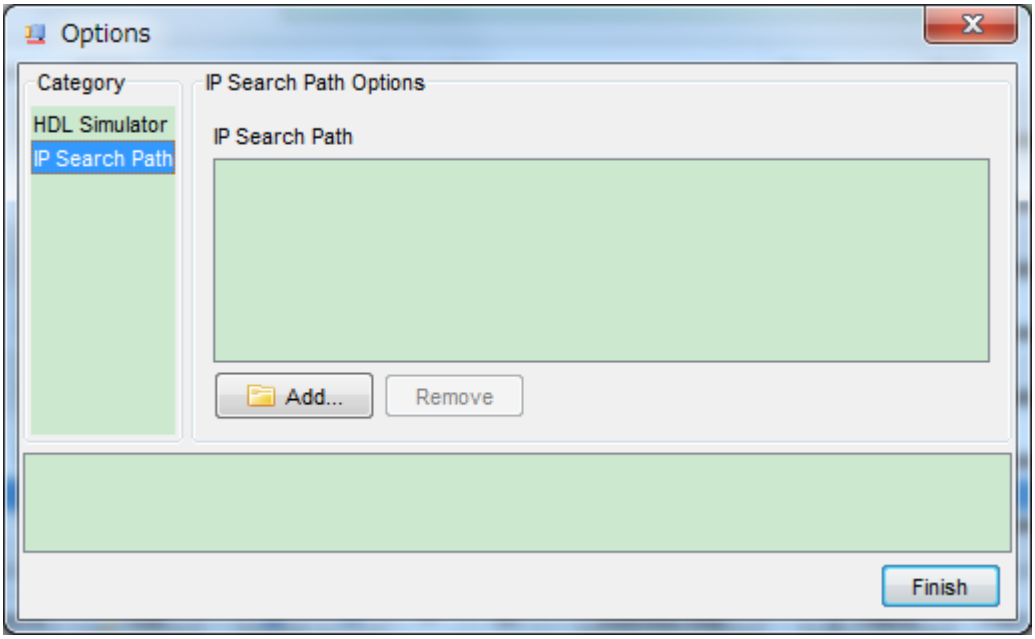


そのまま「Finish」をクリックしてモジュール作成を終わらせます。

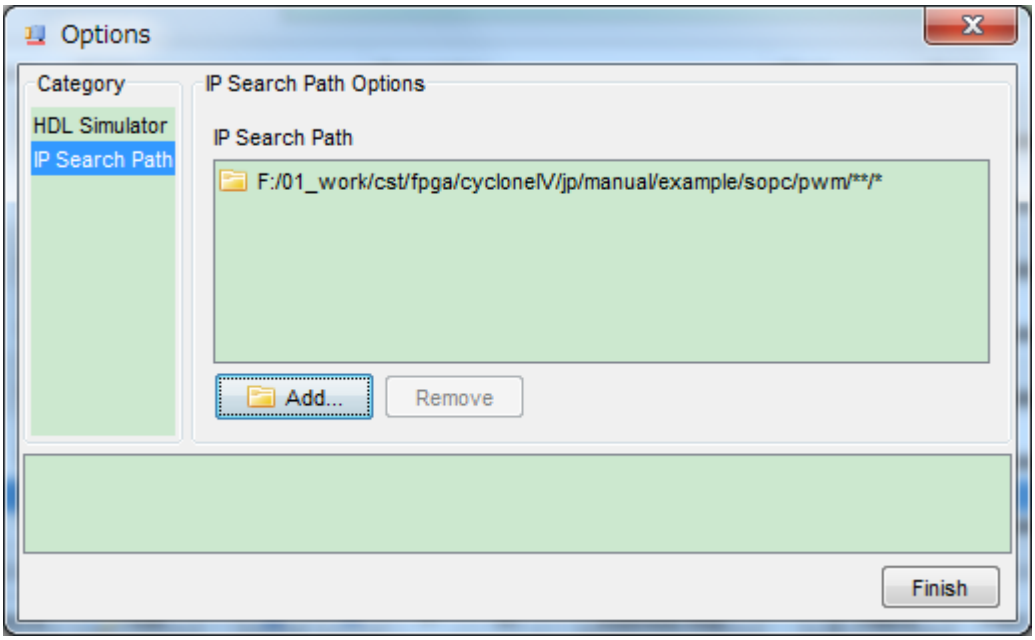
それ以外、もう一つ設定が必要です。メニュー「Tools」→「Options」をクリック



「IP Search Path」を選択、「Add」をクリックして「PWM.v」ファイルのパスを追加します。

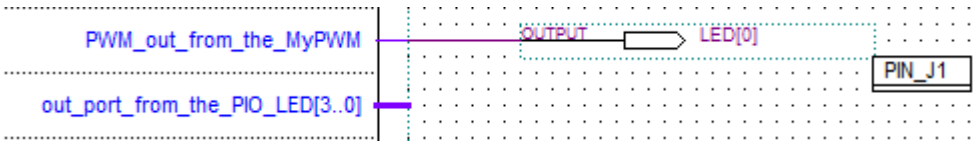


追加後の様子：(「Finish」をクリック)



IP Search Path を追加理由：次回 SOPC Builder が起動しても PWM.v を見つけるための設定です。設定しないと、PWM モジュールが無効になります。
では、PWM 名前を「MyPWM」に修正し、ベースアドレス、IRQ を自動割り当てにし、保存してから「Generate」ボタンをクリックしてからコンパイルします。
コンパイル後、Quartus II メイン画面に戻ります。KERNEL を更新して「MyPWM」が表示されます、一つ LED と接続にし、PWM を使って LED の明るさを変更できます、徐々点灯、徐々消灯を実現できます。(PIO_LED との接続は一時解除必要です。)

TCL ファイル（osh_new.tcl）を実行した後の様子：



最後は全てファイルを保存してコンパイルしましょう。

コンパイル後、開発ボードに AS モードあるいは JTAG モードでダウンロードしてもよいです。

7.10.4 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、PWM に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
/*
 * MyPWM configuration
 *
 */


#define MYPWM_NAME "/dev/MyPWM"
#define MYPWM_TYPE "PWM"
#define MYPWM_BASE 0x00000800
#define MYPWM_SPAN 16
#define ALT_MODULE_CLASS_MyPWM PWM
```

次はメイン関数を修正していきましょう。

```
/*
 * ================================================================
 *
 *      Filename:  main.c
 *
 *    Description:  AVALONバス実例
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
```

```
 * ===============================================================
 */


/*-----------------------------------------------------------
 * Include
 *-----------------------------------------------------------*/
#include <stdio.h>
#include <unistd.h>
#include "system.h"

//レジスタのオフセットにより、構造体PWMを定義
typedef struct{
    volatile unsigned int divi;
    volatile unsigned int duty;
    volatile unsigned int enable;
}PWM;

/*
 * ===  FUNCTION
================================================================================
 *         Name:  main
 *  Description:
 *
================================================================================
===============
 */
int main()
{
    int dir = 1;

    //PWMはMyPWM_BASEとういうベースアドレスをポインタにする
    PWM *pwm = (PWM *)MYPWM_BASE;
    //PWM初期化、diviの最大値が2^32-1。
    pwm->divi = 1000;
    pwm->duty = 0;
```

```
    pwm->enable = 1;

    //dutyの値を絶えず変更させてLEDの一つサイクル内の点灯時間の長さを変更
    while(1){
        if(dir > 0){
            if(pwm->duty < pwm->divi)
                pwm->duty += 100;
            else
                dir = 0;
        }
        else{
            if(pwm->duty > 0)
                pwm->duty -= 100;
            else
                dir = 1;
        }

        usleep(100000);
    }

    return 0;
}
```

ここまで Avalon バスの説明が終わります、プログラムをコンパイルして開発ボードにダウンロードして結果も見れます。

7.11 デジタルチューブ

本節ではデジタルチューブの操作方法及びタイマーの運用方法を説明していきます。

7.11.1 概要

これからデジタルチューブをダイナミックスキャンするプログラムを説明します、このプログラムはタイマーの運用例です。

7.11.2 ハードウエア開発

まず、IP マクロに二つ 6bit の PIO モジュールと一つタイマーを追加必要です、一つ PIO はデータバスとして使われます、もう一つはスウィッチとして使われます、PIO の

※PIO モジュール作成方法は１．PIO モジュールを IP コアに追加を参照してください。

作成後、ベースアドレス、IRQ の自動割り当てを忘れないでください。

保存、「Generate」ボタンをクリックしてコンパイルします。

コンパイル完了後、KERNEL を更新してからピンを追加し名前も修正します、TCL ファイルの実行も行います。最後、コンパイルを実施してください。

コンパイル完了後、開発ボードに AS モードあるいは JTAG モードでダウンロードしましょう。

7.12.3 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

コンパイル完了後、メイン関数（main.c）を修正しましょう。

```
/*
 *
===============================================================================
 *
 *       Filename:  main.c
 *
 *    Description:  デジタルチューブ実例
 *
 *        Version:  1.0.1
 *        Created:  2012.1.31
 *       Revision:  none
 *       Compiler:  Nios II 11.1 IDE
 *           URL:  http://www.csun.co.jp
 *
 *
===============================================================================
 */
```

```
/*---------------------------------------------------------------------
-----------------
 *  Include

*---------------------------------------------------------------------
----------------*/
#include <stdio.h>
#include <sys/unistd.h>
#include <io.h>
#include <string.h>

#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"
/*---------------------------------------------------------------------
-----------------
 *  Define

*---------------------------------------------------------------------
----------------*/
#define   TRUE    1
#define   FALSE   0

/*---------------------------------------------------------------------
-----------------
 *  Variable

*---------------------------------------------------------------------
----------------*/
alt_u8 segtab[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
//0~9
unsigned char led_buffer[8]={0};
unsigned char bittab[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf};
static unsigned char cnt=0;
```

```
/*-----------------------------------------------------------------------
------------------
 *  Function

*-----------------------------------------------------------------------
----------------*/
static void timer_init(void);

/*
 * ===  FUNCTION
======================================================================
 *        Name:  main
 *  Description:
 *
======================================================================
===============
 */
int main(void)
{
    unsigned char i=0,k=0;
    unsigned char buf[20];
    int j=0;

    timer_init();

    while(1){
        sprintf(buf,"%06u",j++);

        for(i=0;i<6;i++){
            led_buffer[i] = buf[5-i]-'0';
        }
        usleep(500000);
    }

    return 0;
```

```
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  timer_handler
 *  Description:
 *
======================================================================
===============
 */
static void timer_handler(void *context, alt_u32 id)
{
    IOWR_ALTERA_AVALON_PIO_DATA(SEG_SEL_BASE, 0xff);
    IOWR_ALTERA_AVALON_PIO_DATA(SEG_SEL_BASE, bittab[cnt]);

    IOWR_ALTERA_AVALON_PIO_DATA(SEG_DAT_BASE, segtab[led_buffer[cnt]]);

    cnt++;
    if(cnt==6)
       cnt=0;

    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0);
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  timer_init
 *  Description:
 *
======================================================================
===============
 */
static void timer_init(void)
{
```

```
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0);

    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE,200000);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE,200000 >> 16);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0x07);

    alt_irq_register(TIMER_IRQ, NULL, timer_handler);
}
```

上記のようにメイン関数を修正してから保存しコンパイルしてください。コンパイル後も開発ボードにダウンロードして結果を確認してください。

7.12 USB デバイス

本節から USB デバイスの操作方法とホストコンピュータに関する内容を説明します。

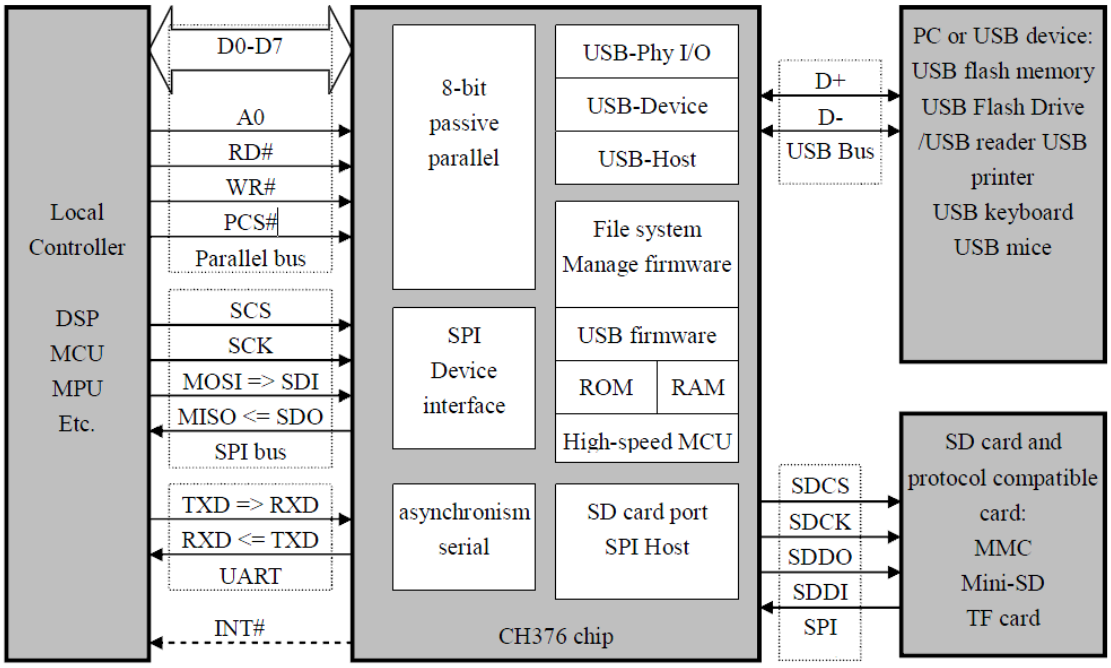
7.12.1 概要

これから開発ボードの USB に関する内容を説明します、開発ボードに USB チップ CH376 を使われます。

このチップは USB デバイスと USB ホスト方式をサポートします、USB 通信プロトコル用の基本フォームウエアも搭載しています、それ以外、Mass-Storage 専用の通信プロトコルフォームウエア、SD カード通信インタフェースフォームウエア、FAT16、FAT32 及び FAT32 ファイルシステム管理のフォームウエアもあります。

よく使われる USB デバイス、例えば、USB メモリ、USB ハードデスク、USB カードリーダ、SD カード（標準容量の SD カードと高容量 HC-SD カード及びプロトコル相性がある MMC と TC カード）をサポートします。

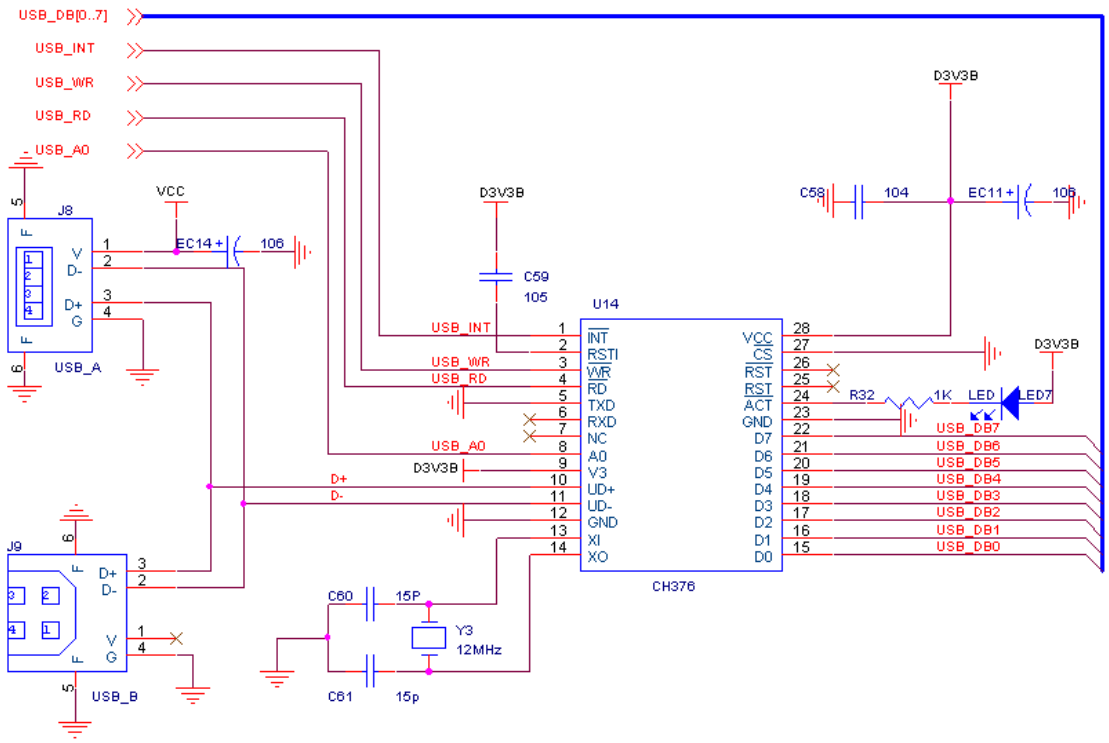
ブロック図は下記です。



チップ内部に USB 通信プロトコルの基本フォームウエアを統合されますので、USB 通信プロトコルをユーザーからプログラムが不要です。それ以外、ファイルシステムの管理用フォームウエアもありますので、USB メモリの内容も直接読み込むことが出来ます、非常に便利です。

7.12.2 ハードウエア開発

まず、USB 関連の回路を下図のように見ましょう。8bit バスを使いますので、回路がすごくシンプルです、FPGA と接続するピンは 12 個であります、中身に、8 つピンがデータバスです、1 つピンが割り込み用のもの、3 つピンが制御用です。



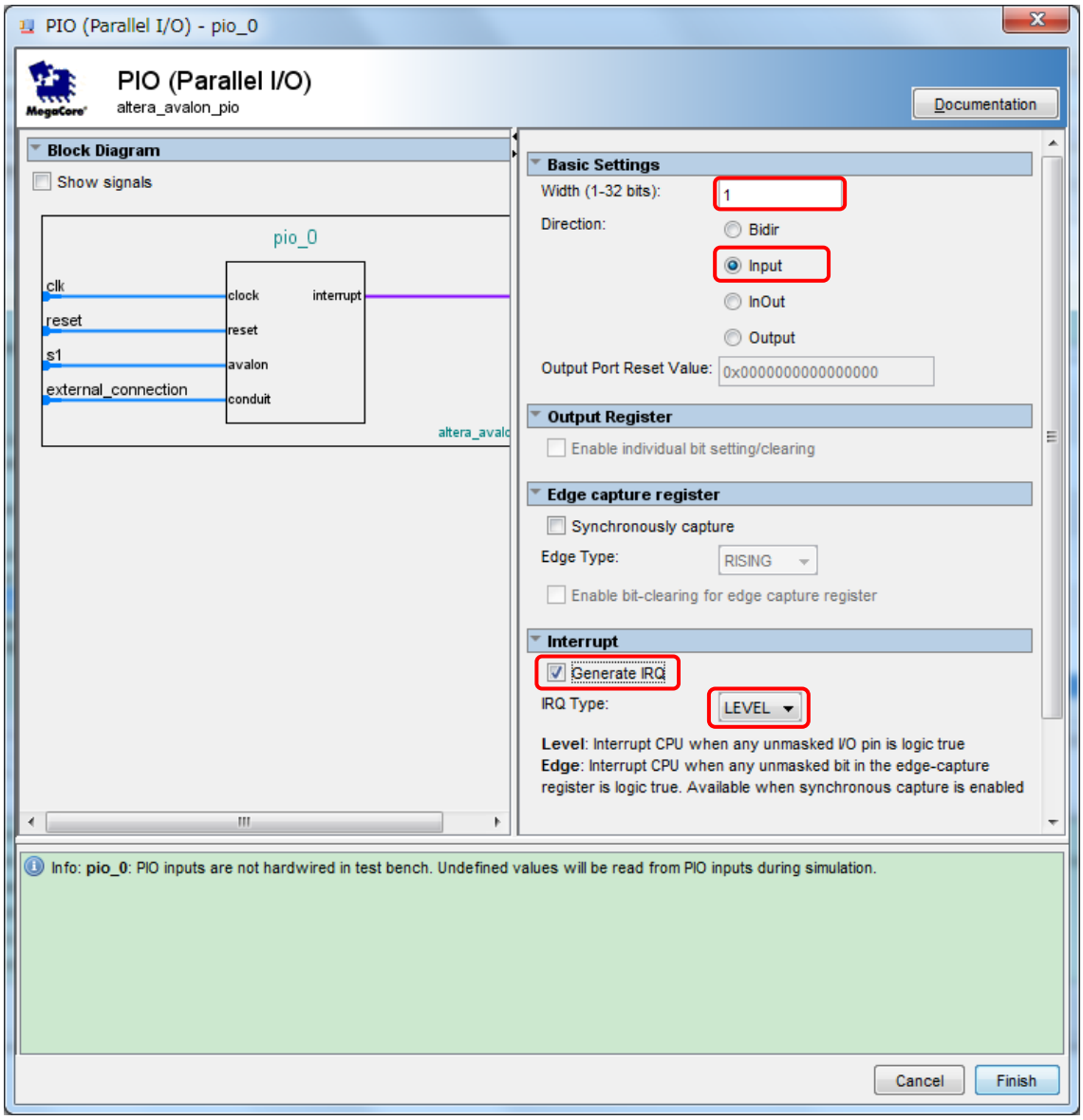
それでは、IP マクロに USB に関するモジュール（すべては PIO モジュール）を追加しましょう。USB_DB が 8bit の出力 PIO（双方向）、USB_WR、USB_RD、USB_A0 は 3 個 1bit の出力 PIO です、残りの USB_nINT は 1bit の入力 PIO です。

※PIO モジュール作成方法は1．PIO モジュールを IP コアに追加を参照してください。

作成後、ベースアドレス、IRQ の自動割り当てを忘れないでください。

☑	⊟ USB_DB	PIO (Parallel I/O)	[clk]		
	s1	Avalon Memory Mapped Slave	clk_0	0x00001960	0x0000196f
☑	⊟ USB_nINT	PIO (Parallel I/O)	[clk]		
	s1	Avalon Memory Mapped Slave	clk_0	0x00001970	0x0000197f
☑	⊟ USB_WR	PIO (Parallel I/O)	[clk]		
	s1	Avalon Memory Mapped Slave	clk_0	0x00001980	0x0000198f
☑	⊟ USB_RD	PIO (Parallel I/O)	[clk]		
	s1	Avalon Memory Mapped Slave	clk_0	0x00001990	0x0000199f
☑	⊟ USB_A0	PIO (Parallel I/O)	[clk]		
	s1	Avalon Memory Mapped Slave	clk_0	0x000019a0	0x000019af

USB_nINT 以外の PIO モジュールは Width と入出力のみと合わせて修正しても良いです、他にデフォルトのままで OK です。USB_nINT は下図のように設定してください。

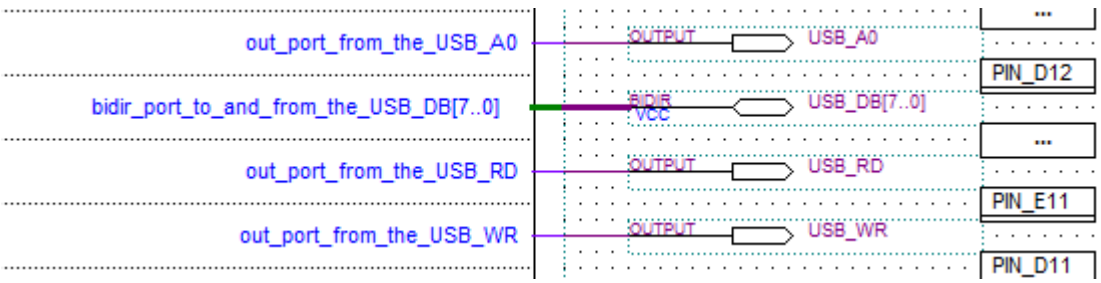


保存、「Generate」ボタンをクリックしてコンパイルします。

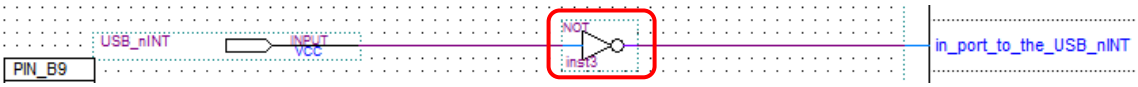
コンパイル完了後、Quartus II メイン画面に戻します。KERNEL を更新し、その後、ピンを追加して TCL ファイルによりピンの名前を修正して TCL ファイルを実行します。

※USB_DB は双方向なので、bidir のピンを使ってください。

実行後の様子：



USB_nINT 割り込みピンについて、低電位で割り込みが発生しますので、NAND ゲートを追加必要です。



最後、すべてファイルを保存してからコンパイルします。

7.12.3 ソフトウエア開発

USB はホストモードとデバイスモードを分けられます、これら二つモードのハードウエア部分は同じですが、ソフトウエアは違います。本節はまずデバイスモードを説明します、デバイスモードは開発ボードが USB インタフェースを経由でホスト PC と接続して開発ボードとホスト PC の間の通信を実現します。

それでは、NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、USB に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
/*
 * USB_DB configuration
 *
 */

#define USB_DB_NAME "/dev/USB_DB"
#define USB_DB_TYPE "altera_avalon_pio"
#define USB_DB_BASE 0x00001960
……
/*
 * USB_nINT configuration
 *
 */

#define USB_NINT_NAME "/dev/USB_nINT"
#define USB_NINT_TYPE "altera_avalon_pio"
#define USB_NINT_BASE 0x00001970
……
/*
 * USB_WR configuration
 *
 */
```

```
#define USB_WR_NAME "/dev/USB_WR"
#define USB_WR_TYPE "altera_avalon_pio"
#define USB_WR_BASE 0x00001980
……
/*
 * USB_RD configuration
 *
 */

#define USB_RD_NAME "/dev/USB_RD"
#define USB_RD_TYPE "altera_avalon_pio"
#define USB_RD_BASE 0x00001990
……
/*
 * USB_A0 configuration
 *
 */

#define USB_A0_NAME "/dev/USB_A0"
#define USB_A0_TYPE "altera_avalon_pio"
#define USB_A0_BASE 0x000019a0
……
```

次はプログラムを修正していきましょう。

１．ヘッダファイル usb.h 追加

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック

usb.h ファイル内容（ch376 のデータシートを参照）

```
/*
 *
================================================================================
 *
 *      Filename:  usb.h
 *
 *   Description:  USBドライバーヘッダファイル
```

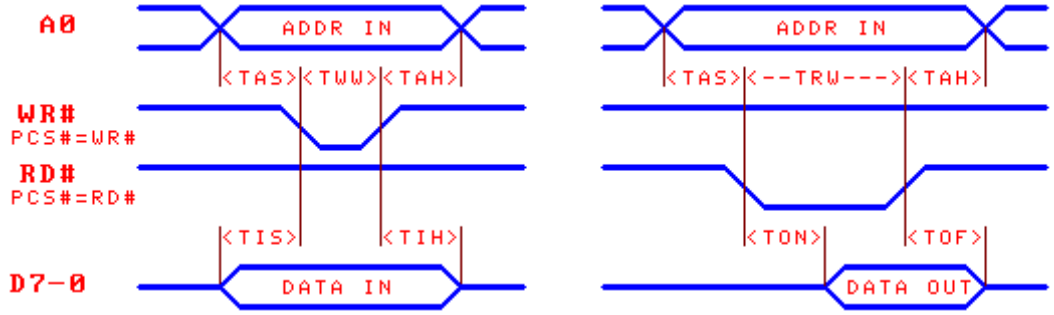
```
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
===============
 */

#ifndef __usb_h__
#define __usb_h__


/*------------------------------------------------------------------------------
------------------
 *  Include

*------------------------------------------------------------------------------
-----------------*/
#include "system.h"
/*------------------------------------------------------------------------------
------------------
 *  Define
 * CH375に関する定義、以下はUSBレジスタアドレスです、この部分について、CH376データ
シートを参照

*------------------------------------------------------------------------------
-----------------*/

//common
#define USB_HOST    0X06
#define USB_DEVICE  0x02
#define USB_DISABLE 0X00
```

```
#define RESET_ALL   0X05
#define CHECK_EXIST 0X06
#define SET_USB_ID  0X12
#define SET_USB_MODE    0X15
#define GET_STATUS  0X22
#define UNLOCK_USB  0X23
#define RD_USB_DATA 0X28
#define WR_USB_DATA5    0X2A
#define WR_USB_DATA7    0X2B
#define GET_IC_VER  0X01
#define ENTER_SLEEP 0X03
#define CHK_SUSPEND 0X0B
#define RD_USB_DATA0    0X27

#define RET_SUCCESS 0X51
#define RET_ABORT   0X5B

#define INT_EP2_OUT 0x02
#define INT_EP2_IN  0x0a

//host
#define DISK_READ   0X54
#define DISK_RD_GO  0X55

#define DISK_READY  0X59
#define DISK_INIT   0X51

//status
#define USB_INT_CONNECT 0x15
#define USB_INT_DISCONNECT 0X16
#define USB_INT_SUCCESS 0X14
#define USB_INT_DISK_READ   0X1D

//usb
//以下の内容はUSBインタフェースの定義です、さまざまプログラミング方法を説明するため
```

```
今回構造体を作成しないです。
//注意点：PIO_USB_DB_DIRの定義
#define PIO_USB_DB     *(volatile unsigned long int *)USB_DB_BASE
#define PIO_USB_WR     *(volatile unsigned long int *)USB_WR_BASE
#define PIO_USB_RD     *(volatile unsigned long int *)USB_RD_BASE
#define PIO_USB_A0     *(volatile unsigned long int *)USB_A0_BASE
#define PIO_USB_INT    *(volatile unsigned long int *)USB_INT_BASE

#define PIO_USB_DB_DIR  *(volatile unsigned long int *)(USB_DB_BASE+4)

#define VID 0X0FFE
#define PID 0X1000

/*------------------------------------------------------------------------
------------------
 *  Struct

*------------------------------------------------------------------------
----------------*/
typedef struct{
   char receive_buffer[200];
   int send_ok_flag;
   int receive_ok_flag;
}USB_T;

/*------------------------------------------------------------------------
------------------
 *  Extern

*------------------------------------------------------------------------
----------------*/
extern USB_T usb;
extern int initialize_usb(void);
extern int set_usb_mode(unsigned char);
extern int send_string_to_usb(char *str,int str_len);
extern void write_command_to_usb(unsigned char command);
```

```
extern void write_data_to_usb(unsigned char data);

#endif //__usb_h__
```

次は CH376 のタイミング図を見てましょう。



上記タイミング図により、USB ドライバーファイルを作成します。

２．ドライバーファイル作成

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「usb.c」を作成します。

usb.c ファイル内容：

```
/*
 *
=====================================================================
===============
 *
 *      Filename:  usb.c
 *
 *    Description:  USBドライバー
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
===============
 */
```

```
/*-----------------------------------------------------------------------
------------------
 *  Include

*-----------------------------------------------------------------------
-----------------*/
#include "../inc/usb.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include <unistd.h>
#include <stdio.h>

/*-----------------------------------------------------------------------
------------------
 *  Function

*-----------------------------------------------------------------------
-----------------*/
void write_command_to_usb(unsigned char command);
void write_data_to_usb(unsigned char data);
unsigned char read_data_from_usb(void);
void delay(void);

/*-----------------------------------------------------------------------
------------------
 *  Variable

*-----------------------------------------------------------------------
-----------------*/
USB_T usb;

/*
 * ===  FUNCTION
=======================================================================
 *       Name:  irq_usb
```

```
 *  Description:  割り込み関数
 *
================================================================================
 */
void irq_usb(void)
{
    unsigned int i;
    unsigned char interrupt_status,data_len;
    //  static int times=0;

    write_command_to_usb(GET_STATUS);

    interrupt_status=read_data_from_usb();

    switch(interrupt_status){
        //Device
        case INT_EP2_OUT:

            write_command_to_usb(RD_USB_DATA);
            data_len=read_data_from_usb();


for(i=0;i<data_len;i++)usb.receive_buffer[i]=read_data_from_usb();
            usb.receive_buffer[i]='¥0';

            usb.receive_ok_flag=1;

            break;

        case INT_EP2_IN:
            write_command_to_usb(UNLOCK_USB);
            usb.send_ok_flag=1;
            break;

        default :break;
```

```
    }

    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(USB_NINT_BASE,0x00);
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  send_string_to_usb
 *  Description:  文字列送信
 *
======================================================================
===============
 */
int send_string_to_usb(char *str,int str_len)
{
    int i;

    write_command_to_usb(WR_USB_DATA7);
    write_data_to_usb(str_len);

    for(i=0;i<str_len;i++)write_data_to_usb(str[i]);

    return 0;
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  initialize_usb
 *  Description:  USB初期化
 *
======================================================================
===============
 */
```

```
int initialize_usb(void)
{
    PIO_USB_RD=1;
    PIO_USB_WR=1;
    PIO_USB_A0=1;
    usb.receive_ok_flag=0;

    // enable the io interrupt

    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(USB_NINT_BASE,1);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(USB_NINT_BASE,0);

    alt_irq_register(USB_NINT_IRQ,NULL,irq_usb);


    set_usb_mode(USB_DEVICE);

    return 0;
}
/*
 * ===  FUNCTION
======================================================================
 *        Name:  set_usb_mode
 *  Description:  USBモード設定
 *
======================================================================
===============
 */
int set_usb_mode(unsigned char type)
{
    write_command_to_usb(SET_USB_MODE);
    write_data_to_usb(type);
    read_data_from_usb();

    if((read_data_from_usb())==0x51)return 0;
    else return -1;
```

```
}
/*
 * ===  FUNCTION
======================================================================
 *        Name:  write_command_to_usb
 *  Description:  ライトコマンド
 *
======================================================================
===============
 */
void write_command_to_usb(unsigned char command)
{
    //A0
    PIO_USB_A0=1;

    //DB DIR output
    PIO_USB_DB_DIR=0xff;

    PIO_USB_DB=command;

    PIO_USB_WR=0;
    PIO_USB_WR=1;

}
/*
 * ===  FUNCTION
======================================================================
 *        Name:  uart_send_byte
 *  Description:  遅延
 *
======================================================================
===============
 */
void delay(void)
{
```

```
    int i;
    for(i=0;i<1000;i++);
}
/*
 * ===  FUNCTION
======================================================================
 *         Name:  write_data_to_usb
 *  Description:  データを書き込む
 *
======================================================================
===============
 */
void write_data_to_usb(unsigned char data)
{
    //A0
    PIO_USB_A0=0;

    //DB DIR output
    PIO_USB_DB_DIR=0xff;

    PIO_USB_DB=data;

    usleep(20);
    PIO_USB_WR=0;
    delay();
    usleep(20);
    PIO_USB_WR=1;



}
/*
 * ===  FUNCTION
======================================================================
 *         Name:  read_data_from_usb
 *  Description:  データを読み込む
 *
```

```
================================================================================
===============
 */
unsigned char read_data_from_usb(void)
{
    unsigned char data=0;

    //A0
    PIO_USB_A0=0;



    //DB DIR output
    PIO_USB_DB_DIR=0;



    PIO_USB_RD=0;
    delay();
    data=PIO_USB_DB;
    PIO_USB_RD=1;

    return data;
}
```

３．メイン関数修正

下記のように修正：

```
/*
 *
================================================================================
===============
 *
 *      Filename:  main.c
 *
 *    Description:  USBデバイス実例
 *
 *       Version:  1.0.1
```

```
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
 */

/*-----------------------------------------------------------------------------
 *  Include
 *-----------------------------------------------------------------------------*/
#include <stdio.h>
#include <unistd.h>
#include "../inc/usb.h"

/*
 * ===  FUNCTION
================================================================================
 *       Name:  main
 *  Description:
 *
================================================================================
 */
int main()
{
    unsigned char tmp[] = "Hello USB!¥n";

    initialize_usb();

    while(1){
```

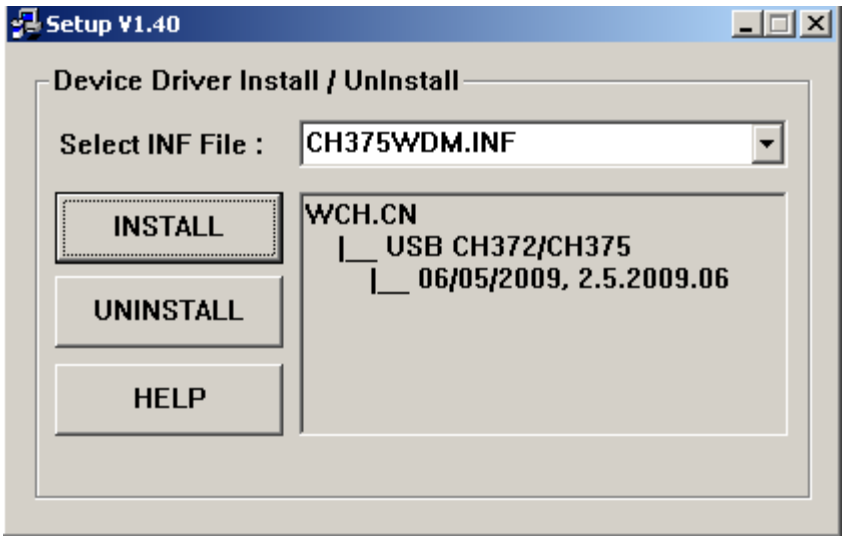
```
        if(usb.receive_ok_flag){
            printf("%s¥n",usb.receive_buffer);
            usb.receive_ok_flag = 0;
        }

        send_string_to_usb(tmp,sizeof(tmp));
        usleep(100000);
    }
    return 0;
}
```

４．ホスト側プログラム

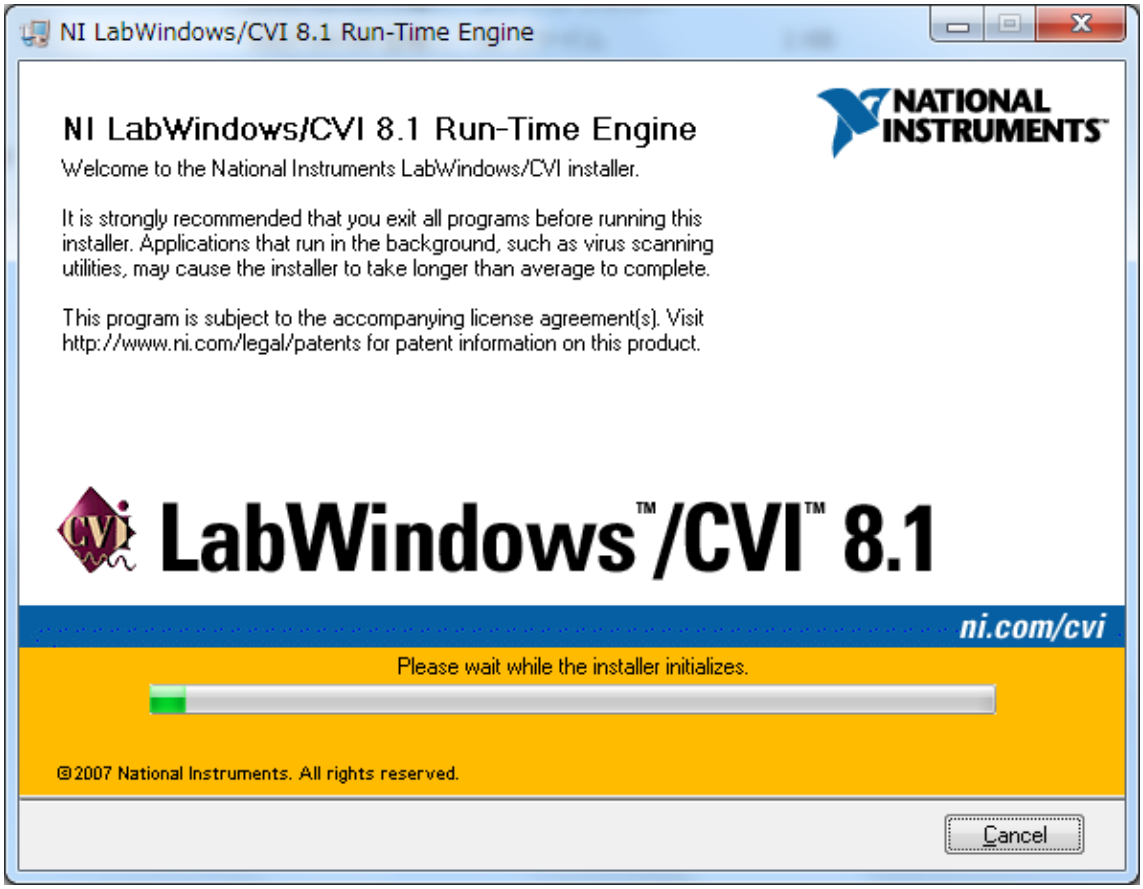
デバッグしたい場合、使われるパソコンに CH376 ドライバをインストール必要です。

下記 URL からダウンロードしインストールしてください。（CH376 が CH372、CH375 のドライバと同じです。）

http://www.dragonwake.com/download/FPGA/EP4CE15/CH376-Driver.zip



インストール完了後、またホスト側のプログラムも必要です。メーカがホスト側プログラムに必要のヘッダファイルとライブラリを提供しています、上記圧縮ファイルを解凍後、「LIB」にあります。このライブラリを利用して NI 社の「Labwindows/CVI 8.1」を使ってホスト側プログラムを作成できます、勿論、皆様は Microsoft 社の VC を使っても良いです。



弊社からホストプログラムを作成しましたので、ダウンロードして参照してください。

http://www.dragonwake.com/download/FPGA/EP4CE15/CH376-Driver.zip

7.13 USB ホスト

7.13.1 概要

USB デバイス節に基づき、CH375 チップを利用して USB メモリをどのように読み込む・書き込むかを説明します。

USB メモリは PC ハードデスク、SD カードと同じのようなファイルシステム（FAT、FAT32）があります、もし、USB メモリで開発ボードが PC とデータ交換を実現したい場合、開発ボード側にも FAT 仕様に準拠しファイルの形で USB メモリのデータを読み込み・書込みます。CH376 チップに FAT ファイルシステムがありますので、この部分作業もすでに実現しました、我々はファイルシステムのソース部分を作成必要でありません。CH376 が USB メモリから読み込み・書込み方法が二つあります、セクターモードとバイトモード。

セクターモード：

セクタ単位で USB メモリのファイルを読み取りと書き込みます（各セクタは通常 512 バイトです）、スピードが若干速いですが、余分のファイルデータバッファは必要です、そして、余分なファイルデータのバッファはセクタ長さ 512 の整数倍であります、それは、頻繁に

アクセス、RAM が大きい、データが多い SCM システムに適しています。セクタ R/W のサブプログラムはセクタ読み込み：CH376SecRead とセクタ書込み：CH376SecWrite という二つ関数があります。

バイトモード：

バイト単位で USB メモリのファイルを読み取りと書き込みます、1 バイトから 65535 バイトでも OK です、スピードが若干遅いですが、余分のファイルデータバッファは必要ではありません、簡単に使えます。それは、RAM が小さい（何バイトから何十 K でよい）、データが少ないあるい少しずつ、あまり頻繁にアクセスされない SCM システムに適しています。但し、Flash は消去の回数が限られますので、頻繁に USB メモリに少しずつデータを書き込むと、USB メモリの寿命が短くなる場合があります。

7.13.2 ソフトウエア開発

ハードウエアは前節に実現されますので、USB メモリ R/W プログラムだけを実現すれば良いです。

１．ヘッダファイル修正

①hal.h ファイル追加

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック

hal.h ファイル内容：

```
/*
 *
=====================================================================
=
 *
 *      Filename:  hal.h
 *
 *    Description:  CH376基本ドライバ
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
 */
```

```
#ifndef __usb_h__
#define __usb_h__

/*-------------------------------------------------------------------------
---
 * Include

*-------------------------------------------------------------------------
*/
#include "system.h"

/*-------------------------------------------------------------------------
---
 * Define

*-------------------------------------------------------------------------
*/

//common
#define USB_HOST       0X06
#define USB_DEVICE     0x02
#define USB_DISABLE    0X00

#define RESET_ALL      0X05
#define CHECK_EXIST    0X06
#define SET_USB_ID     0X12
#define SET_USB_MODE   0X15
#define GET_STATUS     0X22
#define UNLOCK_USB     0X23
#define RD_USB_DATA    0X28
#define WR_USB_DATA5   0X2A
#define WR_USB_DATA7   0X2B
#define GET_IC_VER     0X01
#define ENTER_SLEEP    0X03
#define CHK_SUSPEND    0X0B
```

```
#define RD_USB_DATA0    0X27

#define RET_SUCCESS     0X51
#define RET_ABORT       0X5B

#define INT_EP2_OUT     0x02
#define INT_EP2_IN      0x0a

//host
#define DISK_READ       0X54
#define DISK_RD_GO      0X55

#define DISK_READY      0X59
#define DISK_INIT       0X51

//status

#define USB_INT_CONNECT 0x15


/*-------------------------------------------------------------------------
---
 * Include

*-------------------------------------------------------------------------
*/
#define CMD_RET_SUCCESS     0x51          /* コマンド成功 */
#define CMD_RET_ABORT       0x5F          /* コマンド失敗 */

/*-------------------------------------------------------------------------
---
 * Define

*-------------------------------------------------------------------------
*/
//usb
```

```
#define PIO_USB_DB     *(volatile unsigned long int *)USB_DB_BASE
#define PIO_USB_WR     *(volatile unsigned long int *)USB_WR_BASE
#define PIO_USB_RD     *(volatile unsigned long int *)USB_RD_BASE
#define PIO_USB_A0     *(volatile unsigned long int *)USB_A0_BASE
#define PIO_USB_NINT    *(volatile unsigned long int *)USB_NINT_BASE

#define PIO_USB_DB_DIR  *(volatile unsigned long int *)(USB_DB_BASE+4)

#define VID 0X0FFE
#define PID 0X1000

/*-------------------------------------------------------------------
---
 * Struct

*-------------------------------------------------------------------
*/
typedef struct{
   char receive_buffer[200];
   int send_ok_flag;
   int receive_ok_flag;
}USB_T;

/*-------------------------------------------------------------------
---
 *  Extern

*-------------------------------------------------------------------
*/
extern USB_T usb;
extern void xWriteCH376Cmd(unsigned char command);
extern void xWriteCH376Data(unsigned char data);
extern unsigned char xReadCH376Data(void);
extern unsigned int mInitCH376Host(void);
extern unsigned char Query376Interrupt(void);
```

```
#endif //__usb_h__
```

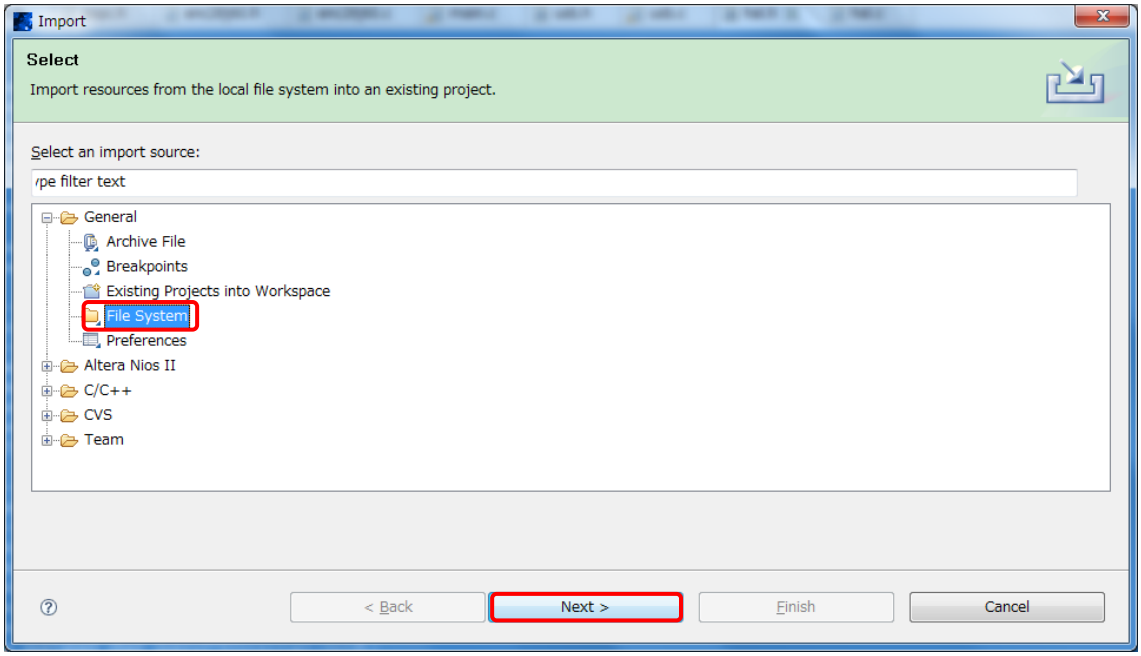
②host.h ファイル作成

このファイルはメーカから提供関数を使うため作成したヘッダファイルです。

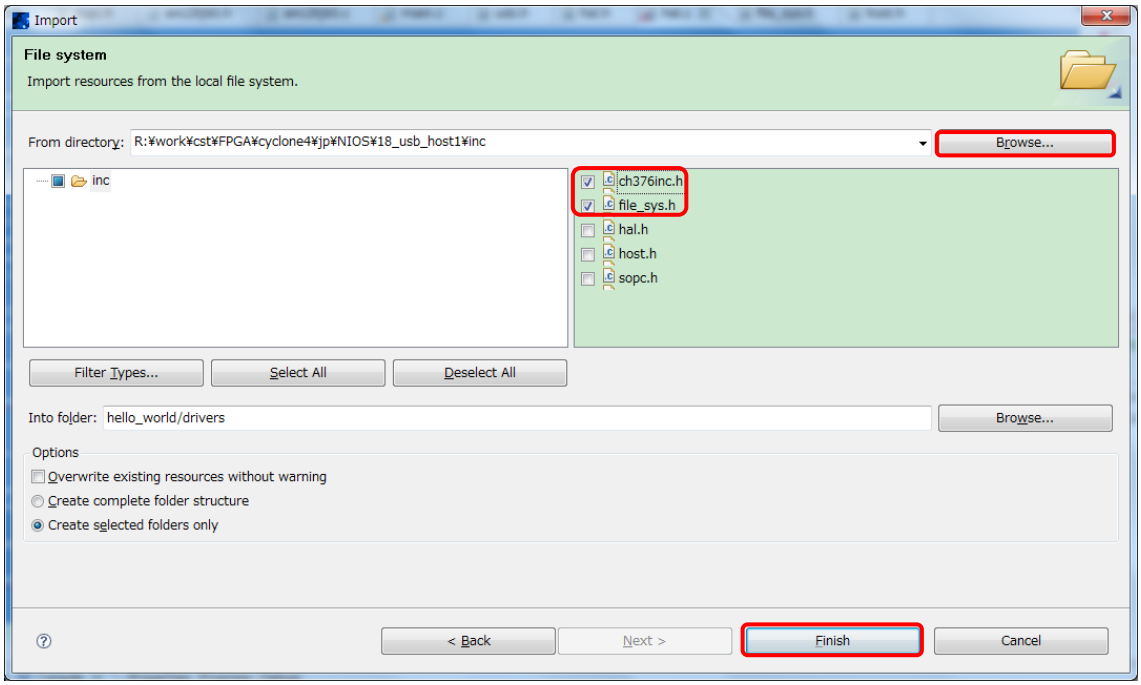
```
/*
 *
=====================================================================
=
 *
 *      Filename:  host.h
 *
 *    Description:  ヘッダファイル
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
 */

#ifndef HOST_H_
#define HOST_H_

extern void host(void);

#endif /*HOST_H_*/
```

③メーカ提供の file_sys.h と ch376.h ファイルをインポート

プロジェクトの inc フォルダを右クリック、メニュー「Import」をクリック、下記画面が出ます、「File System」を選択、「Next」ボタンをクリック



「Browse」をクリック、file_sys.h と ch376inc.h ファイル所属のパスを選択、file_sys.h と ch376inc.h をチェックしてから「Finish」ボタンを押す



2．ドライバーファイル作成

①プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「hal.c」を作成します。

```
/*
 *
```

```
/*=====================================================================
=
 *
 *      Filename:  hal.c
 *
 *    Description:  CH376基本ドライバ
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
 */

/*---------------------------------------------------------------------
---
 * Include

*---------------------------------------------------------------------
*/
#include "../inc/hal.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include <unistd.h>
#include <stdio.h>
#include "../inc/ch376inc.h"
#include "system.h"

/*---------------------------------------------------------------------
---
 * Function

*---------------------------------------------------------------------
```

```
*/
void xWriteCH376Cmd(unsigned char command);
void xWriteCH376Data(unsigned char data);
unsigned char xReadCH376Data(void);
unsigned int mInitCH376Host(void);
void delay(void);
int set_usb_mode(unsigned char type);
unsigned char Query376Interrupt(void);
void irq_usb(void * context,unsigned int id);

/*-----------------------------------------------------------------------
---
 * Struct

*-----------------------------------------------------------------------
*/
USB_T usb;

/*
 * ===  FUNCTION
======================================================
 *        Name:  set_usb_mode
 *  Description:  動作モード設定
 *
=======================================================================
=
 */
int set_usb_mode(unsigned char type)
{

    xWriteCH376Cmd(SET_USB_MODE);
    xWriteCH376Data(type);
    xReadCH376Data();

    if((xReadCH376Data()) == CMD_RET_SUCCESS)
        return 0;
```

```
    else
        return -1;

}
/*
 * ===  FUNCTION
======================================================
 *        Name:   xWriteCH376Cmd
 *  Description:  書き込みコマンド
 *
=====================================================================
=
 */
void xWriteCH376Cmd(unsigned char command)
{

    //A0
    PIO_USB_A0=1;

    //DB DIR output
    PIO_USB_DB_DIR=0xff;

    PIO_USB_DB=command;

    PIO_USB_WR=0;
    PIO_USB_WR=1;

}
/*
 * ===  FUNCTION
======================================================
 *        Name:  delay
 *  Description:  遅延関数
 *
=====================================================================
=
```

```
 */
void delay(void)
{
    int i;
    for(i=0;i<1000;i++);
}
/*
 * ===  FUNCTION
======================================================
 *         Name:  xWriteCH376Data
 *  Description:  データを書き込む
 *
=====================================================================
=
 */
void xWriteCH376Data(unsigned char data)
{
   //A0
   PIO_USB_A0=0;

   //DB DIR output
   PIO_USB_DB_DIR=0xff;

   PIO_USB_DB=data;
   usleep(20);
   PIO_USB_WR=0;
   delay();
   usleep(20);
   PIO_USB_WR=1;
}
/*
 * ===  FUNCTION
======================================================
 *         Name:  xReadCH376Data
 *  Description:  データを読み取り
 *
```

```
=====================================================================
=
 */
unsigned char xReadCH376Data(void)
{
    unsigned char data=0;
     //A0
    PIO_USB_A0=0;

    //DB DIR output
    PIO_USB_DB_DIR=0;

    PIO_USB_RD=0;
    delay();
    data=PIO_USB_DB;
    PIO_USB_RD=1;

    return data;
}
/*
 * ===  FUNCTION
======================================================
 *        Name:  mInitCH376Host
 *  Description:  初期化
 *
=====================================================================
=
 */
unsigned int mInitCH376Host(void)
{
    PIO_USB_RD=1;
    PIO_USB_WR=1;
    PIO_USB_A0=1;
    usb.receive_ok_flag=0;
    usb.send_ok_flag=0;
    //enable the io interrupt
```

```
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(USB_NINT_BASE,0);
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(USB_NINT_BASE,0);

    set_usb_mode(USB_HOST);

    return USB_INT_SUCCESS;
}
/*
 * ===  FUNCTION
======================================================
 *        Name:  Query376Interrupt
 *  Description:  割り込み信号を読み込み
 *
=====================================================================
=
 */
unsigned char Query376Interrupt( void )
{
    return(PIO_USB_NINT?TRUE:FALSE);
}
```

②メーカ提供の file_sys.c と host.c ファイルをインポート

プロジェクトの drivers フォルダを右クリック、メニュー「Import」をクリック、ヘッダファイルと同じ様に c ファイルもインポートします。

host.c ファイル説明：host プログラムはバイトの読み込み・書込み、ファイルの列挙機能を含みます、USB メモリの中「/C51/CH376HFT.C」というファイルの最初 200 バイトを表示させます。ファイル「CH376HFT.C」が見つからない場合、C51 というフォルダの中の全て先頭が CH376 になるファイルを列挙します。C51 というフォルダーも見つからない場合、USB メモリの直下の全てファイルを表示させます。

３．メイン関数修正

```
/*
 *
=====================================================================
=
```

```
 *
 *       Filename:  main.c
 *
 *    Description:  USBメモリ実例
 *
 *        Version:  1.0.1
 *        Created:  2012.1.31
 *       Revision:  none
 *       Compiler:  Nios II 11.1 IDE
 *           URL:  http://www.csun.co.jp
 *
 *
================================================================================
 */

#include <stdio.h>
#include "../inc/hal.h"
#include <unistd.h>
#include "../inc/sopc.h"
#include "../inc/file_sys.h"
#include "../inc/host.h"

int main()
{
   unsigned char tmp[] = "Hello USB!¥n";
   int i;


   printf("Hello from Nios II!¥n");

   host();

   while(1){
      for(i=0;i<4;i++){
         LED->DATA = 1<<i;
         usleep(100000);
```

```
        }

    }

    return 0;
}
```

コンパイル前に、もし前節のプロジェクトをそのまま利用したら、一度プロジェクトから usb.c を外してください。なぜなら、hal.c ファイルにも下記二つ関数を実現します。

```
void delay(void);
int set_usb_mode(unsigned char type);
```

同じ名前の関数を重複実現できませんので、本節使わないファイル「usb.c」を外す必要です。(一時 usb.c.bak という名前に変更しても OK)

最後、コンパイルして開発ボードにダウンロードし、実行しましょう。(実行前、USB メモリに「/C51/CH376HFT.C」というファイルをコピーします、勿論、そのままコピーしなくても良いですが、さまざまの状況を試してプログラムが正しいかどうかも検証できますね。)

7.14 LCD(一)

本節から LCD の原理及び画像表示方法を説明します。

7.14.1 概要

本節からドットマトリクス LCD に関わる内容を説明します、主にハードウエアとドライバ作成を紹介します。

一般に LCD モジュールには LCM (ガラス)、バックライト、駆動 PCB ボードがあります。但し、この三つ部分は一つのみ（LCD：ガラス）が必要です、ここ疑問があるでしょう、駆動 PCB ボードがないと使えるか？

実は、ドットマトリクス LCD モジュールは駆動コントローラの統合方式により、

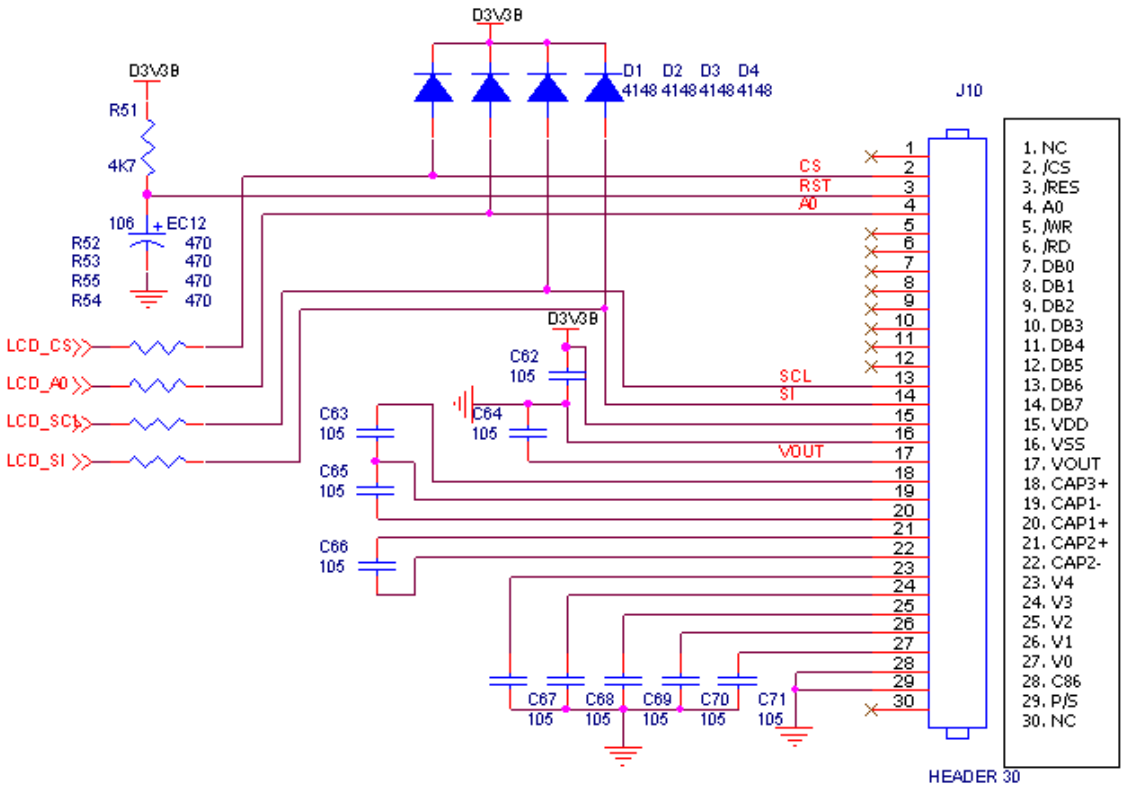
COB と COG を二つ分けられます、COG は駆動コントロールチップが LCM ガラスに統合されます、LCM ガラスに統合できないコンデンサと抵抗を開発ボードに追加すれば、LCD を使えます。もう一つ統合方式:COB については駆動チップを LCD モジュールの後ろ PCB ボードに溶接必要です。ここまで解明しましたね、本マニュアルで使われる開発ボードには 128*64 の COG 液晶を使用されます、この液晶は駆動コントロール IC を LCM に統合されますので、PCB ボードもう必要がなくなります。

この LCD の一部仕様は下記です、駆動チップは ST7565P です、三つ接続インタフェースをサポートします。他の二つ方式と比べて、シリアルタイミング方式は接続簡単、便利に使えると言う特徴がありますので、我々はシリアルタイミング方式を使います。

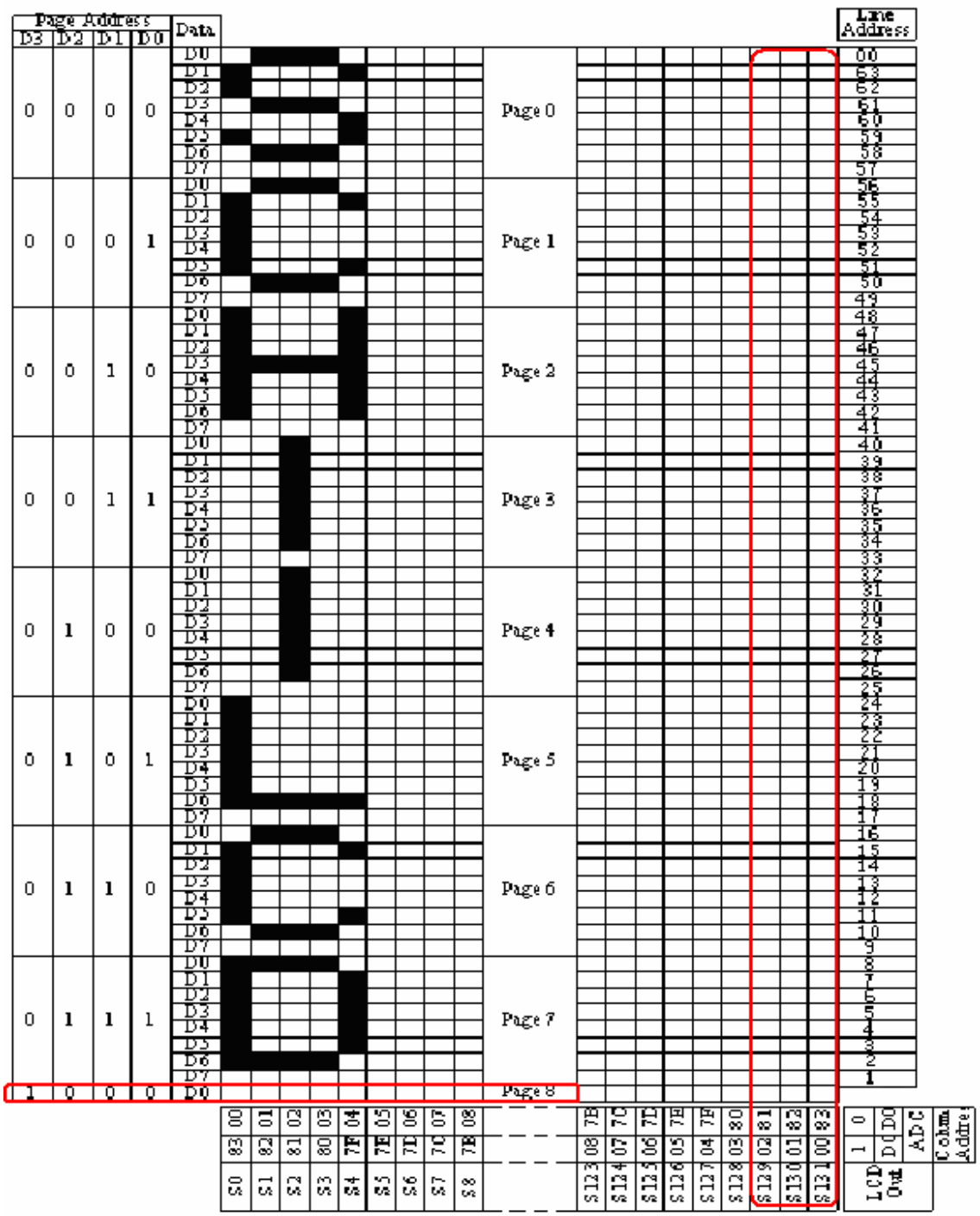
Item	Contents	Unit
LCD Mounting mode	COG, LCD, FPC	
LCD Display mode	Reflective, Transflective and positive	
LCD Display type	STN: Yellow Green mode, Gray mode, Blue mode	
	FSTN	
Viewing direction	6 O'clock or 12 O'clock	
LCD Module size	70.0(W)×50.0(H)×8.0(D,MAX)	mm
LCD Viewing area	54.0(W)×31.0(H)	mm
LCD Display format	128×64 dot matrix	
Dot size	0.34(W)×0.37(H)	mm
Dot pitch	0.38(W)×0.41(H)	mm
LCD Duty	1/65	
LCD Bias	1/9	
LCD Controller/driver LSI	ST7565P (COG)	
LCM Operation temperature (N*)	0~+50	℃
LCM Storage temperature (N*)	-10~+60	℃
LCM Operation temperature (E*)	-20~+70	℃
LCM Storage temperature (E*)	-30~+80	℃
Back light	Edge light LED: Green, White, Blue, Amber	
	EL: White, Yellow green, Blue	
Input data	8080 MPU Interface	
	6800 Series MPU Interface	
	Series data input	
	Parallel data input	
Power supply	2.8-5.5V single power input.	V
	Built- in DC/DC converter for LCD driving.	
	High-accuracy voltage adjustment circuit(thermal gradient –0.05%/℃)	
LCD Expected life	50,000	Hours

7.14.2 LCD 原理紹介

下記回路図により、LCD は 4 ピンのみで FPGA と接続必要です。



LCD に表示するため、駆動コントロール IC のグラフメモリが LCD のドットとの対応関係をよく理解必要です。下図により、LCD のグラフメモリに 8（page）*8+1 行、即ち 65 行、s0-s131、即ち 132 列があります、液晶に 64*128 ドットしかありませんので、グラフメモリの中の一部データは表示できません。実際の操作により、最後の一行（page8 の D0）と最後の三列（ADC が正常時、s129、s130、s131；ADC が逆方向の場合、s0、s1、s2）が表示できません。それ以外、グラフメモリ上のデータは LCD とドット対ドットで一致します。（下図の赤□に囲まれる内容）



液晶上の各ドットはコントロールチップのグラフメモリ RAM のビットと一致します、64*128 個ドットはそれぞれグラフメモリ RAM 上の 8 個 Page とも一致します、Page ごとには 128 バイトのスペースと一致します、下記テーブルで示します。

			列 / 行	LCD地平座標（左から右）								
				0	1	2	3	……	……	125	126	127
LCD縦座標（上から下）	Page0	8bit	0	bit0	bit0	bit0	bit0	……	……	bit0	bit0	bit0
			1	bit1	bit1	bit1	bit1	……	……	bit1	bit1	bit1
			2	bit2	bit2	bit2	bit2	……	……	bit2	bit2	bit2
			……	……	……	……	……	……	……	……	……	……
			6	bit6	bit6	bit6	bit6	……	……	bit6	bit6	bit6
			7	bit7	bit7	bit7	bit7	……	……	bit7	bit7	bit7
	Page1	8bit	8	bit0	bit0	bit0	bit0	……	……	bit0	bit0	bit0
			9	bit1	bit1	bit1	bit1	……	……	bit1	bit1	bit1
			……	……	……	……	……	……	……	……	……	……
			15	bit7	bit7	bit7	bit7	……	……	bit7	bit7	bit7
	……		……	……	……	……	……	……	……	……	……	……
	……		……	……	……	……	……	……	……	……	……	……
	Page7	8bit	56	bit0	bit0	bit0	bit0	……	……	bit0	bit0	bit0
			……	……	……	……	……	……	……	……	……	……
			59	……	……	……	……	……	……	……	……	……
			60	……	……	……	……	……	……	……	……	……
			61	……	……	……	……	……	……	……	……	……
			62	bit6	bit6	bit6	bit6	……	……	bit6	bit6	bit6
			63	bit7	bit7	bit7	bit7	……	……	bit7	bit7	bit7

LCD 上のあるドットを点灯にする時、この点と一致している RAM のあるビットを 1 に設定します。そうすると、この点の所属の行アドレス、列アドレスが必要です。上図により、液晶の行アドレスは Page の情報です、Page ごとに 8 行があります、同じ様に、列アドレスはこの点の地平座標（LCD 上左から右）です。Page の 1 バイトと一致しているのは 1 列（8 行、即ち 8 個ドット）、全部は 128 列があります、このロジックにより、プログラムの中に LCD の表示をコントロールできます。

7.14.3 ハードウエア開発

これから NIOS II で LCD をどのように駆動するかを説明します。まず、IP マクロで 4 つ PIO モジュールを追加必要です、この 4 つ PIO モジュールは LCD の 4 つピンを対応します。（PIO モジュールは全て 1bit の出力ものです。）

※PIO モジュール作成方法は1．PIO モジュールを IP コアに追加を参照してください。

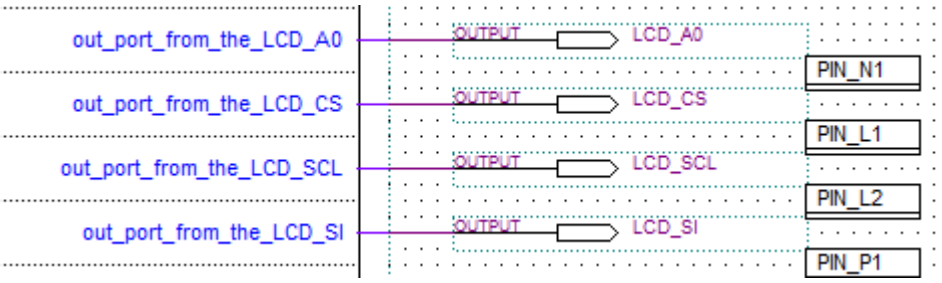
作成後、ベースアドレス、IRQ の自動割り当てを忘れないでください。

保存、「Generate」ボタンをクリックしてコンパイルします。

コンパイル完了後、Quartus II メイン画面に戻します、KERNEL を更新し、その後、ピン

を追加して TCL ファイルによりピンの名前を修正して TCL ファイルを実行します。

実行後の様子：



最後、すべてファイルを保存してからコンパイルします。

7.14.4 ソフトウエア開発

NIOS II IDE11.1 を起動させ、起動後、ソフトマクロが変わりますので、まず、プロジェクトをコンパイルしてください。（ショットキー：Ctrl+b）

暫く待って、コンパイル完了後、LCD に関する内容（ベースアドレスと IRQ）を system.h ファイルに追加されます。

```
/*
 * LCD_SI configuration
 *
 */

#define LCD_SI_NAME "/dev/LCD_SI"
#define LCD_SI_TYPE "altera_avalon_pio"
#define LCD_SI_BASE 0x000019b0
……
/*
 * LCD_A0 configuration
 *
 */

#define LCD_A0_NAME "/dev/LCD_A0"
#define LCD_A0_TYPE "altera_avalon_pio"
#define LCD_A0_BASE 0x000019c0
……
/*
 * LCD_SCL configuration
 *
 */
```

```
#define LCD_SCL_NAME "/dev/LCD_SCL"
#define LCD_SCL_TYPE "altera_avalon_pio"
#define LCD_SCL_BASE 0x000019d0
……
/*
 * LCD_CS configuration
 *
 */

#define LCD_CS_NAME "/dev/LCD_CS"
#define LCD_CS_TYPE "altera_avalon_pio"
#define LCD_CS_BASE 0x000019e0
.....
```

次はプログラムを修正していきましょう。

１．ヘッダファイル修正

①sopc.h ファイルは下記のソースを追加します。

```
#define _LCD
……
#ifdef _LCD
#define LCD_CS          ((PIO_STR *) LCD_CS_BASE)
#define LCD_SCL         ((PIO_STR *) LCD_SCL_BASE)
#define LCD_A0          ((PIO_STR *) LCD_A0_BASE)
#define LCD_SI          ((PIO_STR *) LCD_SI_BASE)
#endif /* _LCD */
```

②lcd.h 新規作成

プロジェクトの inc フォルダを右クリック、メニュー「New」→「Header File」をクリック

lcd.h ファイル内容

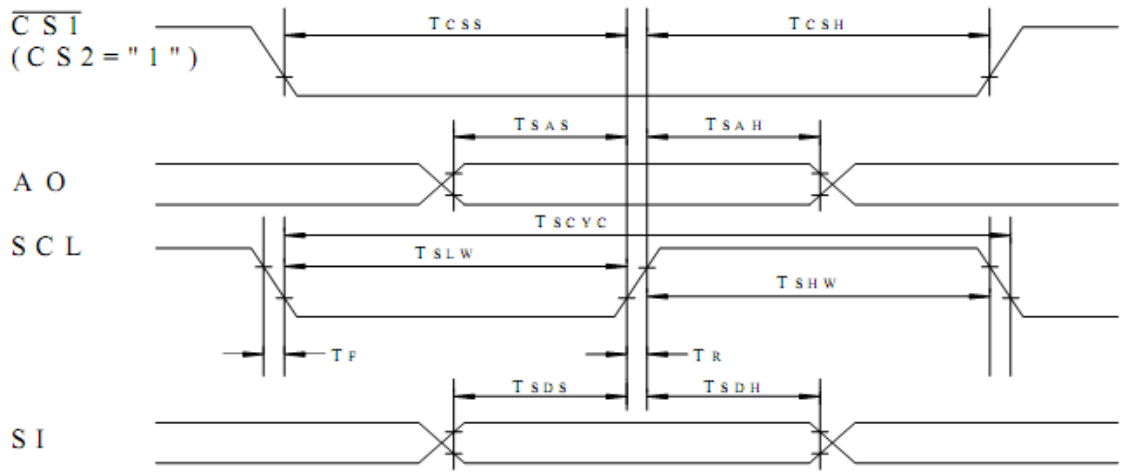
```
/*
 *
================================================================
===============
 *
 *      Filename:  lcd.h
 *
```

```
 *    Description:  LCDヘッダファイル
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
===============
 */

#ifndef _lcd_h_
#define _lcd_h_

extern void initialize_lcd(void);
extern void draw_screen(unsigned char *p);
extern void clear(void);
//extern void write_data(unsigned char dat);
//extern void data_send(unsigned char dat);
//extern void write_command(unsigned char com);

extern unsigned char buf[];
#endif //_lcd_h_
```

２．ドライバーファイル作成

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をクリック、ドライバファイル「lcd.c」を作成します。

LCD のシリアルタイミング図によりドライバーを作成します。



lcd.c ファイル内容：

```
/*
 *
=====================================================================
===============
 *
 *      Filename:  lcd.c
 *
 *    Description:  LCDドライバファイル
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
===============
 */

/*---------------------------------------------------------------------
---------
 *  Include
```

```
*----------------------------------------------------------------------
---------*/
#include "system.h"
#include <unistd.h>
#include "../inc/sopc.h"


/*----------------------------------------------------------------------
---------
 *  Variable

*----------------------------------------------------------------------
---------*/
unsigned char buf[]={

/*--  画像ファイルを読み込み：C:¥無題.bmp  --*/
/*--  幅x高さ=128x64  --*/
0x00,0x00,0x00,0xC0,0x40,0x20,0x20,0x20,0x10,0x10,0x10,0x08,0x08,0x08,
0x08,0x18,
0xE8,0x24,0x22,0x42,0x44,0x84,0x88,0x08,0x08,0x08,0x90,0x50,0x30,0xF0,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x01,0x06,0x38,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x07,0xB8,0xC0,0x00,0x00,0x00,0x01,0x01,0xFF,0x00,0x2A,0x00,0xFF,
```

```
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x78,0x84,0x02,0x03,0x0C,0x08,0x04,0x1C,0x64,0x82,
0x02,0x01,
0x03,0x1D,0x61,0x02,0x84,0x78,0x20,0x40,0x40,0x7F,0x44,0x22,0x11,0x0F,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x02,0x02,0x02,0x02,0x04,0x04,0x05,
0x04,0x02,
0x02,0x02,0x02,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

```
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,
0xFF,0xFF,0x0F,0x0F,0x0F,0x0F,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0x1F,0x0F,
0x0F,0xCF,
0xEF,0xFF,0xFF,0xFF,0xC7,0xC7,0xC7,0xC7,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x07,
0x07,0x07,
0x07,0x07,0x07,0x07,0x07,0x07,0x07,0x07,0x07,0x07,0x07,0xE7,0xE7,0xE7,
0xE7,0xE7,
0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0x07,
0xE7,0xE7,
0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0xE7,0x07,0x07,
0x07,0x07,
0x07,0x07,0x07,0x87,0xE7,0xE7,0xE7,0xE7,0xE7,0x07,0x07,0x07,0x07,0x07,
0xFF,0xFF,
0xFF,0xFF,0x00,0x00,0x00,0x00,0x1F,0x8F,0x03,0x01,0x00,0x30,0xFC,0xFE,
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0x01,0x01,0x01,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x01,
0x01,0x01,
0xF3,0xF3,0xF1,0xF1,0xF0,0xE0,0x01,0x01,0x03,0xFF,0xFF,0xFF,0x3F,0x0F,
0x03,0x03,
0xC1,0xF1,0xF8,0xF8,0xF8,0xF8,0xF9,0x01,0x01,0x01,0xFF,0xFF,0xFF,0x00,
0xF0,0xF8,
0xFC,0xFC,0xFC,0xBC,0xBC,0x3C,0x3C,0x3C,0x7C,0x7C,0x7C,0x03,0x03,0x03,
0x03,0x03,
0x03,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x03,0x03,0x03,0x03,0x03,0x03,0x00,
```

```
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xF3,0xF3,0xF3,0xE3,0xE3,0xE3,0xC3,0x83,0x00,0x00,
0x00,0x0F,
0x0F,0x0F,0x0F,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,
0xFF,0xFF,
0xFF,0xFF,0x80,0x80,0x80,0x80,0xFF,0xFF,0xFF,0xFE,0xF8,0xF0,0xE0,0x81,
0x87,0x8F,
0x9F,0xFF,0xFF,0xFF,0x80,0x80,0x80,0x80,0xFF,0xFF,0xFF,0xFF,0x80,0x80,
0x80,0x80,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x80,0x80,0x80,0xFF,0xFF,0xFF,0xFE,0xF0,
0xE0,0xC0,
0x83,0x87,0x8F,0x8F,0xCF,0xEF,0xE7,0x00,0x00,0x00,0xFF,0xFF,0xFF,0x00,
0xF3,0xE7,
0xEF,0xCF,0xCF,0xCF,0xDF,0xFF,0xFF,0xFF,0xFF,0xFE,0x7C,0x00,0x00,0x00,
0x00,0x00,
0x00,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0xF0,
0xF3,0xE3,
0xE1,0xC1,0xC1,0xC1,0xC1,0xE3,0xFF,0xFF,0xFF,0xFF,0xFF,0x7F,0x00,0x00,
0x00,0xC0,
0xC0,0xC0,0xC0,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xC0,0xC0,0xC0,0xC0,0x00,
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xE3,0xE3,
0xE3,0xE3,0xE3,0xE3,0xE3,0xE1,0xF0,0xF0,0xF8,0xFE,0xFF,0xFF,0xFF,0xE0,
0xE1,0xE3,
0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE1,0xE1,0xE0,0xE0,0xE0,0xE0,
0xE0,0xE0,
0xE0,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE0,0xE0,0xE0,0xE0,0xE0,0xE0,0xE1,
0xE3,0xE3,
0xE3,0xE7,0xE7,0xE7,0xE7,0xE7,0xE3,0xE3,0xE3,0xE1,0xE0,0xE0,0xE0,0xE0,
0xE0,0xE3,
0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE3,0xE0,
```

```
0xFF,0xFF,
};
//-----------------Function Prototype--------------------//
void initialize_lcd(void);
void draw_screen(unsigned char *p);
void clear(void);
void write_data(unsigned char dat);
void data_send(unsigned char dat);
void write_command(unsigned char com);
void set_x(unsigned char x);
void set_y(unsigned char y);

/*
 * ===  FUNCTION
======================================================================
 *        Name:  data_send
 *  Description:  1バイトデータを送信
 *
======================================================================
===============
 */
void data_send(unsigned char dat)
{
    unsigned char i;

    LCD_CS->DATA=0;
    LCD_SCL->DATA=0;

    for(i=0;i<8;i++){
        if(dat&0x80)LCD_SI->DATA=1;
        else LCD_SI->DATA=0;

        dat<<=1;

        LCD_SCL->DATA=0;
```

```
        LCD_SCL->DATA=1;
    }

    LCD_CS->DATA=1;
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  write_command
 *  Description:  A0に低電位を設定、コマンドを書き込む
 *
======================================================================
===============
 */
void write_command(unsigned char com)
{
    LCD_A0->DATA=0;

    data_send(com);
}



/*
 * ===  FUNCTION
======================================================================
 *         Name:  write_data
 *  Description:  A0に高電位を設定、データを書き込む
 *
======================================================================
===============
 */
void write_data(unsigned char dat)
{
    LCD_A0->DATA = 1;
```

```
    data_send(dat);
}

/* * === FUNCTION ======================================
 * Name: set_x
 * Description: 地平座標アドレスを設定
 * ===================================================== */
void set_x(unsigned char x)
{
    write_command(x>>4|0x10);
    write_command(x&0xf);
}

/* * === FUNCTION ======================================
 * Name: set_y
 * Description: Pageアドレスを設定、すべては８ページである
 * ===================================================== */
void set_y(unsigned char y)
{
    write_command(y|0xb0);
}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  clear
 *  Description:
 *
======================================================================
===============
 */
void clear(void)
{
    int seg;
    int page;
```

```
    for(page=0xb0;page<0xb8;page++) {
        write_command(page);
        write_command(0x10);
        write_command(0x00);

        for(seg=0;seg<128;seg++){
            write_data(0);
        }
    }

}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  draw_screen
 *  Description:  128*64の画像を表示
 *
======================================================================
===============
 */
void draw_screen(unsigned char *p)
{
    int seg;
    int page;

    for(page=0xb0;page<0xb8;page++) {
        write_command(page);
        write_command(0x10);
        write_command(0x00);

        for(seg=0;seg<128;seg++){
            write_data(*p++);
        }
    }
```

```
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  initialize_lcd
 *  Description:  LCD初期化、初期化関数はメーカから提供されます、データシートを参照
 *
======================================================================
===============
 */
void initialize_lcd(void)
{
    write_command(0xaf); //ON DISPLAY
    write_command(0x40); //STAR DISPLAY
    write_command(0xa0); //ADC NORMAL
    write_command(0xa6); //Display Normal
    write_command(0xa4); //CLEAR
    write_command(0xa2); //1/9BIAS
    write_command(0xc8); //COMMON OUTPUT DIRECTION
    write_command(0x2f); //POWER CONTROL
    write_command(0x24); //RESISTER RATIO
    write_command(0x81); //VOLUM MODE SET
    write_command(0x24); //RESISTER RATIO
}
```

３．メイン関数修正

```
/*
 *
======================================================================
===============
 *
 *      Filename:  main.c
 *
 *    Description:  LCD実例
 *
```

```
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
================================================================================
 */

/*-----------------------------------------------------------------------------
 *  Include
 *-----------------------------------------------------------------------------*/
#include "../inc/lcd.h"

/*
 * ===  FUNCTION
================================================================================
 *       Name:  main
 *  Description:
 *
================================================================================
 */
int main(void)
{
    initialize_lcd();
    clear();
    draw_screen(buf);

}
```

ここまですべてソースを作成しました、コンパイル後、プログラムを開発ボードにダウンロードして効果が見られます。

7.15 LCD(二)

本節から LCD に英語、日本語の表示方法を説明します。

7.15.1 概要

前節に基づき、英語及び日本語の表示方法を紹介します。

LCD 上に文字の表示については、英語とも日本語とも、フォントライブラリが必要です。

ある LCD モジュールでフォントライブラリを既にチップに組み込まれます、このようなフォントライブラリは利点、欠点が両方もあります。利点は操作簡単です、欠点は拡張性が低いです、表示効果があまりよくないです。

開発ボードに付けられる LCD はフォントライブラリを搭載していないです。ここ良く使われる表示方法を説明します、外部フォントライブラリを利用して綺麗な文字を表示できるようにしましょう。

7.15.2 英語フォントライブラリ

まず、英語フォントライブラリの仕組みを紹介します。英語フォントライブラリで ASCII コードを使われます、コードが 0～127 までです、アドレッシング方法は下記通りです。

英語ドットデータが英語フォントライブラリでのオフセット=英語の ASCII コード×一つ英語文字所要バイト数

我々が使っているフォントライブラリは tahoma フォントです、高さが 16bit、幅が変動されます。下記の tahoma_font_offset[]の中、コメントがオフセットと一対一です、例えば、「！」のオフセットが 7、幅が 11-7=4bit です、即ち、文字幅＝次の文字のオフセット-文字自身のオフセット

```
#ifndef TAHOMA_H_
#define TAHOMA_H_
//文字の相対オフセット
unsigned int tahoma_font_offset[]={
//   ! "  #  $  %  &  '  (
0 ,3 ,7 ,11 ,19 ,25 ,36 ,43 ,45 ,49 ,
// ) *  +   ,  -  .  /  0 1  2
53 ,59 ,67 ,71 ,75 ,79 ,83 ,89 ,95 ,101,
// 3 4  5  6  7  8  9  :  ;  <
107,113,119,125,131,137,143,147,152,159,
// = >  ?  @  A  B  C  D  E  F
168,175,180,190,197,203,210,217,223,229,
// G H  I  J  K  L  M  N  O  P
236,243,247,252,258,263,271,278,286,292,
```

```
// Q R  S  T  U  V  W  X  Y  Z
300,307,313,319,326,332,342,348,354,360,
// [ ¥  ]  ^  _  `  a  b  c  d
364,368,372,380,387,392,398,404,409,415,
// e f  g  h  i  j  k  l  m  n
421,425,431,437,439,442,447,449,457,463,
// o p  q  r  s  t  u  v  w  x
469,475,481,485,490,494,500,506,514,520,
// y z  {  |  }  ~
526,531,536,540,545,553 };

const char
font_tahoma_8[]={ 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0B,0xF0,0x00,0x00,0x00,0x00,
0x00,0x38,
0x00,0x00,0x00,0x38,0x00,0x00,0x02,0x00,0x0E,0x40,0x03,0xC0,0x0E,0x70,0x03,0xC0,
0x02,0x70,0x00,0x40,0x00,0x00,0x08,0xC0,0x09,0x20,0x3F,0xF8,0x09,0x20,0x06,0x20,
0x00,0x00,0x00,0x60,0x00,0x90,0x00,0x90,0x0C,0x60,0x03,0x00,0x00,0xC0,0x06,0x30,
0x09,0x00,0x09,0x00,0x06,0x00,0x00,0x00,0x07,0x60,0x08,0x90,0x08,0x90,0x09,0x60,
0x06,0x00,0x05,0x80,0x08,0x00,0x00,0x38,0x00,0x00,0x07,0xC0,0x18,0x30,0x20,0x08,
0x00,0x00,0x20,0x08,0x18,0x30,0x07,0xC0,0x00,0x00,0x02,0x80,0x01,0x00,0x07,0xC0,
0x01,0x00,0x02,0x80,0x00,0x00,0x01,0x00,0x01,0x00,0x01,0x00,0x0F,0xE0,0x01,0x00,
0x01,0x00,0x01,0x00,0x00,0x00,0x20,0x00,0x1C,0x00,0x00,0x00,0x00,0x00,0x01,0x00,
0x01,0x00,0x01,0x00,0x00,0x00,0x08,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x38,0x00,
0x07,0xC0,0x00,0x38,0x00,0x00,0x07,0xE0,0x08,0x10,0x08,0x10,0x08,0x10,0x07,0xE0,
0x00,0x00,0x00,0x00,0x08,0x20,0x0F,0xF0,0x08,0x00,0x00,0x00,0x00,0x00,0x0C,0x20,
0x0A,0x10,0x09,0x10,0x08,0x90,0x08,0x60,0x00,0x00,0x04,0x20,0x08,0x10,0x08,0x90,
0x08,0x90,0x07,0x60,0x00,0x00,0x01,0x80,0x01,0x40,0x01,0x20,0x0F,0xF0,0x01,0x00,
0x00,0x00,0x04,0xF0,0x08,0x90,0x08,0x90,0x08,0x90,0x07,0x10,0x00,0x00,0x07,0xC0,
0x08,0xA0,0x08,0x90,0x08,0x90,0x07,0x00,0x00,0x00,0x00,0x00,0x10,0x0C,0x10,0x03,0x10,
0x00,0xD0,0x00,0x30,0x00,0x00,0x07,0x60,0x08,0x90,0x08,0x90,0x08,0x90,0x07,0x60,
0x00,0x00,0x00,0xE0,0x09,0x10,0x09,0x10,0x05,0x10,0x03,0xE0,0x00,0x00,0x00,0x00,
0x0C,0xC0,0x00,0x00,0x00,0x00,0x20,0x00,0x1C,0xC0,0x00,0x00,0x00,0x00,0x00,0x00,
0x01,0x00,0x02,0x80,0x02,0x80,0x04,0x40,0x04,0x40,0x08,0x20,0x00,0x00,0x02,0x80,
0x02,0x80,0x02,0x80,0x02,0x80,0x02,0x80,0x02,0x80,0x00,0x00,0x00,0x00,
0x08,0x20,0x04,0x40,0x04,0x40,0x02,0x80,0x02,0x80,0x01,0x00,0x00,0x00,0x00,0x10,
0x0B,0x10,0x00,0x90,0x00,0x60,0x00,0x00,0x07,0xC0,0x08,0x20,0x13,0x90,0x14,0x50,
```

```
0x14,0x50,0x17,0xD0,0x04,0x10,0x04,0x20,0x03,0xC0,0x00,0x00,0x0E,0x00,0x03,0xC0,
0x02,0x30,0x02,0x30,0x03,0xC0,0x0E,0x00,0x00,0x00,0x0F,0xF0,0x08,0x90,0x08,0x90,
0x08,0x90,0x07,0x60,0x00,0x00,0x03,0xC0,0x04,0x20,0x08,0x10,0x08,0x10,0x08,0x10,
0x08,0x10,0x00,0x00,0x0F,0xF0,0x08,0x10,0x08,0x10,0x08,0x10,0x04,0x20,0x03,0xC0,
0x00,0x00,0x0F,0xF0,0x08,0x90,0x08,0x90,0x08,0x90,0x08,0x10,0x00,0x00,0x0F,0xF0,
0x00,0x90,0x00,0x90,0x00,0x90,0x00,0x90,0x00,0x00,0x03,0xC0,0x04,0x20,0x08,0x10,
0x09,0x10,0x09,0x10,0x0F,0x10,0x00,0x00,0x0F,0xF0,0x00,0x80,0x00,0x80,0x00,0x80,
0x00,0x80,0x0F,0xF0,0x00,0x00,0x08,0x10,0x0F,0xF0,0x08,0x10,0x00,0x00,0x08,0x00,
0x08,0x10,0x08,0x10,0x07,0xF0,0x00,0x00,0x0F,0xF0,0x01,0x80,0x02,0x40,0x04,0x20,
0x08,0x10,0x00,0x00,0x0F,0xF0,0x08,0x00,0x08,0x00,0x08,0x00,0x00,0x00,0x0F,0xF0,
0x00,0x30,0x00,0xC0,0x03,0x00,0x00,0xC0,0x00,0x30,0x0F,0xF0,0x00,0x00,0x0F,0xF0,
0x00,0x30,0x00,0xC0,0x03,0x00,0x0C,0x00,0x0F,0xF0,0x00,0x00,0x03,0xC0,0x04,0x20,
0x08,0x10,0x08,0x10,0x08,0x10,0x04,0x20,0x03,0xC0,0x00,0x00,0x0F,0xF0,0x01,0x10,
0x01,0x10,0x01,0x10,0x00,0xE0,0x00,0x00,0x03,0xC0,0x04,0x20,0x08,0x10,0x08,0x10,
0x18,0x10,0x24,0x20,0x23,0xC0,0x00,0x00,0x0F,0xF0,0x01,0x10,0x01,0x10,0x03,0x10,
0x04,0xE0,0x08,0x00,0x00,0x00,0x08,0x60,0x08,0x90,0x08,0x90,0x08,0x90,0x07,0x10,
0x00,0x00,0x00,0x10,0x00,0x10,0x0F,0xF0,0x00,0x10,0x00,0x10,0x00,0x00,0x07,0xF0,
0x08,0x00,0x08,0x00,0x08,0x00,0x08,0x00,0x07,0xF0,0x00,0x00,0x00,0x70,0x03,0x80,
0x0C,0x00,0x03,0x80,0x00,0x70,0x00,0x00,0x00,0x70,0x03,0x80,0x0C,0x00,0x03,0x80,
0x00,0x70,0x03,0x80,0x0C,0x00,0x03,0x80,0x00,0x70,0x00,0x00,0x0C,0x30,0x02,0x40,
0x01,0x80,0x02,0x40,0x0C,0x30,0x00,0x00,0x00,0x30,0x00,0xC0,0x0F,0x00,0x00,0xC0,
0x00,0x30,0x00,0x00,0x0C,0x10,0x0A,0x10,0x09,0x90,0x08,0x50,0x08,0x30,0x00,0x00,
0x3F,0xF8,0x20,0x08,0x20,0x08,0x00,0x00,0x00,0x38,0x07,0xC0,0x38,0x00,0x00,0x00,
0x20,0x08,0x20,0x08,0x3F,0xF8,0x00,0x00,0x00,0x80,0x00,0x40,0x00,0x20,0x00,0x10,
0x00,0x20,0x00,0x40,0x00,0x80,0x00,0x00,0x20,0x00,0x20,0x00,0x20,0x00,0x20,0x00,
0x20,0x00,0x20,0x00,0x00,0x00,0x00,0x00,0x08,0x00,0x10,0x00,0x00,0x00,0x00,0x00,
0x06,0x00,0x09,0x40,0x09,0x40,0x09,0x40,0x0F,0x80,0x00,0x00,0x0F,0xF8,0x08,0x40,
0x08,0x40,0x08,0x40,0x07,0x80,0x00,0x00,0x07,0x80,0x08,0x40,0x08,0x40,0x08,0x40,
0x00,0x00,0x07,0x80,0x08,0x40,0x08,0x40,0x08,0x40,0x0F,0xF8,0x00,0x00,0x07,0x80,
0x09,0x40,0x09,0x40,0x09,0x40,0x05,0x80,0x00,0x00,0x0F,0xF0,0x00,0x48,0x00,0x48,
0x00,0x00,0x07,0x80,0x28,0x40,0x28,0x40,0x28,0x40,0x1F,0xC0,0x00,0x00,0x0F,0xF8,
0x00,0x40,0x00,0x40,0x00,0x40,0x0F,0x80,0x00,0x00,0x0F,0xD0,0x00,0x00,0x20,0x40,
0x1F,0xD0,0x00,0x00,0x0F,0xF8,0x01,0x00,0x02,0x80,0x04,0x40,0x08,0x00,0x0F,0xF8,
0x00,0x00,0x0F,0xC0,0x00,0x40,0x00,0x40,0x0F,0x80,0x00,0x40,0x00,0x40,0x0F,0x80,
0x00,0x00,0x0F,0xC0,0x00,0x40,0x00,0x40,0x00,0x40,0x0F,0x80,0x00,0x00,0x07,0x80,
0x08,0x40,0x08,0x40,0x08,0x40,0x07,0x80,0x00,0x00,0x3F,0xC0,0x08,0x40,0x08,0x40,
```

```
0x08,0x40,0x07,0x80,0x00,0x00,0x07,0x80,0x08,0x40,0x08,0x40,0x08,0x40,0x3F,0xC0,
0x00,0x00,0x0F,0xC0,0x00,0x80,0x00,0x40,0x00,0x00,0x09,0x80,0x09,0x40,0x0A,0x40,
0x06,0x40,0x00,0x00,0x07,0xF0,0x08,0x40,0x08,0x40,0x00,0x00,0x07,0xC0,0x08,0x00,
0x08,0x00,0x08,0x00,0x0F,0xC0,0x00,0x00,0x00,0xC0,0x03,0x00,0x0C,0x00,0x03,0x00,
0x00,0xC0,0x00,0x00,0x03,0xC0,0x0C,0x00,0x03,0x00,0x00,0xC0,0x03,0x00,0x0C,0x00,
0x03,0xC0,0x00,0x00,0x08,0x40,0x04,0x80,0x03,0x00,0x04,0x80,0x08,0x40,0x00,0x00,
0x00,0xC0,0x33,0x00,0x0C,0x00,0x03,0x00,0x00,0xC0,0x00,0x00,0x0C,0x40,0x0A,0x40,
0x09,0x40,0x08,0xC0,0x00,0x00,0x01,0x00,0x01,0x00,0x1E,0xF0,0x20,0x08,0x00,0x00,
0x00,0x00,0x3F,0xF8,0x00,0x00,0x00,0x00,0x20,0x08,0x1E,0xF0,0x01,0x00,0x01,0x00,
0x00,0x00,0x03,0x00,0x00,0x80,0x00,0x80,0x01,0x00,0x02,0x00,0x02,0x00,0x01,0x80,
0x00,0x00};

#endif /*TAHOMA_H_*/
```

7.15.3 日本語フォントライブラリ

キャラクタ LCD に表示する時、LED のドットマトリクスから文字を描画します、即ちビットマップフォントです。ビットマップフォントのファイル構造としては、BDF（Bitmap Distribution Format)、PCF（Portable Compiled Format)、そして FONTX 形式の 3 つが代表的なビットマップフォント形式のようである。フリーのビットマップフォントとして公開されているフォン トの大体が、これらの形式で配布されていた。

ここでは FONTX 形式のものを使います。

まず、FONTX フォントファイル仕様を簡単に紹介しましょう。

ファイルには３つのセクションがある。

■フォントヘッダ

■コードテーブル（ CodeType によるオプション - CodeType はフォントヘッダで指定 ）

■フォントパターン

１．フォントヘッダ

●00 - 05 ： 6byte ： Identifer - "FONTX2" という文字

●06 - 13 ： 8byte ： フォント名

●14 - 14 ： 1byte ： X size - 幅

●15 - 15 ： 1byte ： Y size - 高さ

●16 - 16 ： 1byte ： CodeType - 0 で ASCII, 1 で ２バイト文字

２．コードテーブル

○CodeType = 1 のときはフォントヘッダに続き

○コードテーブルがあるコードテーブルで指定されたコードの領域がフォントパターンに隙間無く並べられる

●17 - 17 ： 1byte ： エントリ数
●18 - 19 ： 2byte ： １番目の領域開始コード
●20 - 21 ： 2byte ： １番目の領域終了コード
以下 エントリの数だけテーブルが続く
●18 + (N-1)*2 - 19 + (N-1)*2 ： 2byte ： Ｎ番目の領域開始コード
●20 + (N-1)*2 - 21 + (N-1)*2 ： 2byte ： Ｎ番目の領域終了コード

３．フォントパターン
○フォントヘッダで指定された X size 幅のビット列が Y size 個並んで１フォント
○フォントを表現するビットの並びが延々と最後まで続く
●0 bit - X-1 bit ： １番目のフォントの１ライン目
Y size だけ繰り返し
●(N-1) * X bit - N * X-1 bit ： １番目のフォントのＮライン目
それぞれのラインは、ライン毎にバイトアラインされる
●フォントの数だけ繰り返し

FONTX2 なフォントのリンク集

- 東雲フォント
- Font Silo - 幾つものフォントが公開されている
- 恵理沙フォント
- ぱうフォント

FONTX2 形式フォント作成ツール：
http://www.hmsoft.co.jp/lepton/software/dosv/fontx.htm
FONTX2 形式フォントから C 言語用のヘッダファイルに変換ツール：
http://www.programmersheaven.com/file/c/utils/BIN2C.ZIP
※弊社サンプルから既に FONTX2 の 16 フォントを作成しました、そのまま利用してください。(FONTX2 作成手順は省略します。)

7.15.4 ソフトウエア開発

上記フォント仕様を理解のうえ、プログラムを作成しましょう。
FONTX フォントドライバーはインタネットから流用するものがあります。
※「組み込みで FONTX2 形式を使用するドライバを作成する」を参照ください。
１．FONTX ドライバファイルをインポート
■C 言語用のフォントファイルをインポート
○SHGZN16X.INC：ゴシック 16 フォント
○SHMZN16X.INC：明朝 16 フォント

○SHNHN16X.INC：ASCII コード

■fontdata.h、fontdrv.h を NIOS II IDE プロジェクト「hello_world」の inc フォルダーにインポート

■fontdata.c、fontdrv.c を NIOS II IDE プロジェクト「hello_world」の driver フォルダーにインポート

fontdrv.c に下記内容を修正

```
//コメントします。
//#define FONT_BLOCK_S_KANJI(n)
       (SWITCH_ENDIAN_US((((FontX_Kanji*)font_table_kanji)->Block[n]).
Start))
//#define FONT_BLOCK_E_KANJI(n)
       (SWITCH_ENDIAN_US((((FontX_Kanji*)font_table_kanji)->Block[n]).
End))
//追加
#define FONT_BLOCK_S_KANJI(n)
((((FontX_Kanji*)font_table_kanji)->Block[n]).Start)
#define FONT_BLOCK_E_KANJI(n)
((((FontX_Kanji*)font_table_kanji)->Block[n]).End)
```

２．ヘッダファイル作成

①gui.h ファイル作成

```
/*
 *
=============================================================================
===============
 *
 *      Filename:  gui.h
 *
 *    Description:  LCD表示ヘッダファイル
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
```

```
================================================================================
===============
 */
#ifndef GUI_H_
#define GUI_H_

typedef struct{
       unsigned char x;
       unsigned char y;
       unsigned char * matrix;
       unsigned char wide;
       unsigned char reverse;
}X_16_T;

typedef struct{
    unsigned char x;
    unsigned char y;
    unsigned char * str;
    unsigned char reverse;
}CHARACTER_STRING;

typedef struct{
    unsigned char x;
    unsigned char y;
    unsigned char reverse;
}STRING_T;

extern unsigned char printf_string(STRING_T * p, char * fmt, ...);
extern unsigned char printf_jp_string(STRING_T * p, char * fmt, ...);

#define BUFFER_SIZE 200

#endif /*GUI_H_*/
```

３．GUI 操作ファイル作成

プロジェクトの drivers フォルダを右クリック、メニュー「New」→「Source File」をク

リック、ドライバファイル「gui.c」を作成します。

gui.c ファイル内容：

```
/*
 *
=====================================================================
===============
 *
 *      Filename:  gui.c
 *
 *    Description:  LCD表示
 *
 *       Version:  1.0.1
 *       Created:  2012.1.31
 *      Revision:  none
 *      Compiler:  Nios II 11.1 IDE
 *          URL:  http://www.csun.co.jp
 *
 *
=====================================================================
===============
 */
#include "../inc/gui.h"
#include "../inc/lcd.h"
#include "../inc/tahoma.h"
#include "../inc/hzk12.h"
#include "../inc/fontdrv.h"
#include "../inc/fontdata.h"
#include <string.h>
#include <stdarg.h>

/*
  漢字フォントに関する定義
 */
// フォントサイズ(X)
#define FONT_SIZE_X_KANJI
(((FontX_Kanji*)font_table_kanji)->XSize)
```

```
// フォントサイズ(Y)
#define FONT_SIZE_Y_KANJI
(((FontX_Kanji*)font_table_kanji)->YSize)
// 1フォントのサイズ(Bytes)
#define FONT_SIZE_KANJI           ((FONT_SIZE_X_KANJI >> 3) *
FONT_SIZE_Y_KANJI)


static int _gui_x_16(X_16_T *p);
static int  _print_tahoma(CHARACTER_STRING *p);
static int _tahoma_string_length(char * p);
unsigned char printf_string(STRING_T * p, char * fmt, ...);
static int h_to_v( unsigned char *dest, unsigned char *source);
int _print_simsun(CHARACTER_STRING *p);

/*
 * ===  FUNCTION
======================================================================
 *        Name:  _gui_x_16
 *  Description:
 *
======================================================================
===============
 */
int _gui_x_16(X_16_T *p)
{
    unsigned char i;

    if(p == NULL)return -1;
    if(p->x > 128)return -1;
    if(p->y > 8)return -1 ;
    if(p->matrix == NULL)return -1;

    if(p->reverse > 1)return -1;

    lcd.set_x(p->x);
```

```
    lcd.set_y(p->y);
    for(i=0;i<p->wide;i++){

lcd.write_data(p->reverse?p->matrix[2*i+1]^0xfe:p->matrix[2*i+1]);
    }

    lcd.set_x(p->x);
    lcd.set_y(p->y+1);
    for(i=0;i<p->wide;i++){
           lcd.write_data(p->reverse?~p->matrix[2*i]:p->matrix[2*i]);
    }

   return 0;

}
/*
 * ===  FUNCTION
======================================================================
 *         Name:  _print_tahoma
 *  Description:
 *
======================================================================
===============
 */
int  _print_tahoma(CHARACTER_STRING *p)
{
    unsigned char l,i=0;
    unsigned int wide;
    X_16_T tmp;

    if(p == NULL)return 0;
    if(p->x > 128)return 0;
    if(p->y > 8)return 0;
    if(p->str == NULL)return 0;
    if(p->reverse > 1)return 0;
```

```
    l=strlen((const char *)p->str);

    for(i=0;i<l;i++){
        wide=tahoma_font_offset[p->str[i]-' '+1];
        wide-=tahoma_font_offset[p->str[i]-' '];
            tmp.x=p->x;
            tmp.y=p->y;
            tmp.matrix=(unsigned char
*)font_tahoma_8+tahoma_font_offset[p->str[i]-' ']*2;
            tmp.reverse=p->reverse;
            tmp.wide=wide;

            _gui_x_16(&tmp);

            p->x+=wide;
    }

    return 0;
}
/*
 * ===  FUNCTION
======================================================================
 *         Name:  _tahoma_string_length
 *  Description:
 *
======================================================================
===============
 */
int _tahoma_string_length(char * p)
{
    int temp=0;

    while(*p !='¥0'){
        temp+=tahoma_font_offset[*p-' '+1];
        temp-=tahoma_font_offset[*p-' '];
        p++;
```

```
    }
    return temp;
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  SJISMultiCheck
 *  Description:
 *
======================================================================
===============
 */
int SJISMultiCheck(unsigned char c){
   if(((c>=0x81)&&(c<=0x9f))||((c>=0xe0)&&(c<=0xfc)))return 1;
   else return 0;
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  StringCount
 *  Description:
 *
======================================================================
===============
 */
int StringCount(const char *str){
   /*****************************
   strの文字列の文字数を返す関数(日本語対応)

   戻り値：strの文字数

   const char *str：文字数を数える文字列
   *****************************/
   int i,cnt=0;
```

```
   for(i=0;str[i]!='¥0';){//(1)
      if(SJISMultiCheck(str[i]))i+=2;
      else i++;
      cnt++;
   }
   return cnt;
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  JISlength
 *  Description:
 *
======================================================================
===============
 */
int JISlength(const char *str){
   /*****************************
   strの文字列の文字数を返す関数(日本語対応)

   戻り値：日本語の長さ（全角：２文字；半角：1文字）

   const char *str：文字数を数える文字列
   *****************************/
   int i;
   for(i=0;str[i]!='¥0';){//(1)
      if(SJISMultiCheck(str[i]))i+=2;
      else i++;
   }
   return i;
}



/*
 * ===  FUNCTION
```

```
================================================================================
 *        Name:  printf_string
 *  Description:  中国語とASCIIを混在してLCDへ出力関数
 *
================================================================================
===============
 */
unsigned char printf_string(STRING_T * p, char * fmt, ...)
{
    unsigned char buf1[BUFFER_SIZE];
    CHARACTER_STRING tmp;
    unsigned int i=0,offset,c=0;
    va_list arg_ptr;
    unsigned char buf[BUFFER_SIZE];

   if(p == NULL)return 0;
   if(p->x > 191)return 0;
   if(p->y > 6)return 0 ;

   memset(buf,'¥0',sizeof(buf));

   va_start(arg_ptr,fmt);
   vsprintf(buf,fmt,arg_ptr);
   va_end(arg_ptr);

   offset=p->x;

   while(buf[c]!='¥0'){

      if(buf[c] < 0x7f && buf[c]!='¥n'){
         i=0;
         memset(buf1,'¥0',sizeof(buf1));

         while(buf[c]<0x7f && buf[c]!='¥0' &&buf[c]!='¥n'){
            buf1[i++]=buf[c++];
         }
```

```
        tmp.x=offset;
        offset+=_tahoma_string_length(buf1);
        tmp.y=p->y;
        tmp.str=buf1;

        tmp.reverse=p->reverse;

        if(_print_tahoma(&tmp)==-1)return 0;

    }

    if(buf[c] > 0x7f){
        i=0;
        memset(buf1,'¥0',sizeof(buf1));

        while(buf[c] >0x7f){
            buf1[i++]=buf[c++];
            buf1[i++]=buf[c++];
        }

        tmp.x=offset;
        offset+=i*6;

        tmp.y=p->y;
        tmp.str=buf1;
        tmp.reverse=p->reverse;

        if(_print_simsun(&tmp)==-1)return 0;
    }
    //new line symbol

    if(buf[c] =='¥n'){
        p->y+=2;
        offset=p->x;
        c++;
```

```
        }
    }
    p->x=tmp.x;

    return 0;
}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  printf_jp_string
 *  Description:  日本語(Shift JISコード)とASCIIをLCDへ出力関数
 *
======================================================================
===============
 */
unsigned char printf_jp_string(STRING_T * p, char * fmt, ...)
{
     unsigned char buf1[BUFFER_SIZE];
     CHARACTER_STRING tmp;
     unsigned int i=0,offset,c=0;
     va_list arg_ptr;
     unsigned char buf[BUFFER_SIZE];

    if(p == NULL)return 0;
    if(p->x > 191)return 0;
    if(p->y > 6)return 0 ;

    memset(buf,'¥0',sizeof(buf));

    va_start(arg_ptr,fmt);
    vsprintf(buf,fmt,arg_ptr);
    va_end(arg_ptr);

    offset=p->x;
```

```
    while(buf[c]!='¥0'){

        if(buf[c] < 0x7f && buf[c]!='¥n'){
            i=0;
            memset(buf1,'¥0',sizeof(buf1));

            while(buf[c]<0x7f && buf[c]!='¥0' &&buf[c]!='¥n'){
                buf1[i++]=buf[c++];
            }

            tmp.x=offset;
            offset+=_tahoma_string_length(buf1);
            tmp.y=p->y;
            tmp.str=buf1;

            tmp.reverse=p->reverse;

            if(_print_tahoma(&tmp)==-1)return 0;

        }

        if(buf[c] > 0x7f){
            i=0;
            memset(buf1,'¥0',sizeof(buf1));

            while(buf[c] >0x7f){
                buf1[i++]=buf[c++];
                buf1[i++]=buf[c++];
            }

            tmp.x=offset;
            offset+=JISlength(buf1);

            tmp.y=p->y;
            tmp.str=buf1;
            tmp.reverse=p->reverse;
```

```
            if(_print_sjis(&tmp)==-1)return 0;
        }
        //new line symbol

        if(buf[c] =='¥n'){
            p->y+=2;
            offset=p->x;
            c++;
        }
    }
    p->x=tmp.x;

    return 0;
}

/*
 * ===  FUNCTION
======================================================================
 *        Name:  h_to_v
 *  Description:
 *
======================================================================
===============
 */
static int h_to_v( unsigned char *dest, unsigned char *source)
{
    int i,j;

    for(j=0;j<8;j++){

for(i=0;i<8;i++)if(source[30-i*2]&(0x80>>j))dest[j*2]|=(0x80>>i);

for(i=0;i<8;i++)if(source[14-i*2]&(0x80>>j))dest[j*2+1]|=(0x80>>i);
    }
```

```
    for(j=0;j<4;j++){

for(i=0;i<8;i++)if(source[31-i*2]&(0x80>>j))dest[j*2+16]|=(0x80>>i);

for(i=0;i<8;i++)if(source[15-i*2]&(0x80>>j))dest[j*2+17]|=(0x80>>i);
    }

    return 1;
}
/*
 * ===  FUNCTION
================================================================================
 *         Name:  _print_simsun
 *  Description:
 *
================================================================================
===============
 */
int _print_simsun(CHARACTER_STRING *p)
{
    unsigned char l,i;
    X_16_T tmp;
    unsigned int offset=0;
    unsigned char bua[32],bub[32];

    if(p == NULL)return 0;
    if(p->x > 128)return 0;
    if(p->y > 64)return 0 ;
    if(p->str == NULL)return 0;
    if(p->reverse > 1)return 0;

    l=strlen(p->str);
    l/=2;

    for(i=0;i<l;i++){
        offset=p->str[2*i];
```

```
        offset-=0xa1;
        offset*=94;
        offset+=p->str[2*i+1];
        offset-=0xa1;
        offset*=32;

        memcpy(bua,achzk12+offset,32);

        memset(bub,0,32);
        h_to_v(bub,bua);

        tmp.x=p->x;
        tmp.y=p->y;
        tmp.reverse=p->reverse;
        tmp.matrix=bub;
        tmp.wide=12;

        _gui_x_16(&tmp);

        p->x+=12;
    }
    return 0;
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  _print_sjis
 *  Description:
 *
======================================================================
===============
 */
int _print_sjis(CHARACTER_STRING *p)
{
```

```
    unsigned char l,i;
    X_16_T tmp;
    unsigned int fontaddr=0;
    unsigned char bua[32],bub[32];

    if(p == NULL)return 0;
    if(p->x > 128)return 0;
    if(p->y > 64)return 0 ;
    if(p->str == NULL)return 0;
    if(p->reverse > 1)return 0;

    l=StringCount(p->str);

    for(i=0;i<l;i++){
        fontaddr=GetFontPtr_Kanji(p->str[i]);
        memcpy(bua,fontaddr,FONT_SIZE_KANJI);
        tmp.x=p->x;
        tmp.y=p->y;
        tmp.reverse=p->reverse;
        tmp.matrix=bua;
        tmp.wide=FONT_SIZE_X_KANJI;

        _gui_x_16(&tmp);

        p->x+=12;
    }
    return 0;
}
```

４．メイン関数修正

※今回修正の関数もボード出荷時用のメイン関数、前の実例を全部動かすようにします。

```
/*
 * "Hello World" example.
 *
```

```
 * This example prints 'Hello from Nios II' to the STDOUT stream. It runs
on
 * the Nios II 'standard', 'full_featured', 'fast', and 'low_cost' example
 * designs. It runs with or without the MicroC/OS-II RTOS and requires a
STDOUT
 * device in your system's hardware.
 * The memory footprint of this hosted application is ~69 kbytes by default
 * using the standard reference design.
 *
 * For a reduced footprint version of this template, and an explanation
of how
 * to reduce the memory footprint for a given application, see the
 * "small_hello_world" template.
 *
 */

#include <stdio.h>
#include "../inc/usb.h"
#include <unistd.h>
#include "../inc/enc28j60.h"
#include "../inc/lcd.h"
#include "../inc/uart.h"
#include "../inc/ds1302.h"
#include "../inc/iic.h"
#include "../inc/gui.h"
#include "../inc/fontdata.h"
#include "../inc/key.h"

#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "alt_types.h"
#include "sys/alt_irq.h"


#define BUZZER *(volatile long int *)BUZZER_BASE
```

```
unsigned char time[7] = {0x00,0x19,0x14,0x17,0x03,0x17,0x10};//フォーマット：秒　分 時 日 月 曜日 年

alt_u8 segtab[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
//0~9
unsigned char led_buffer[8]={0};
unsigned char bittab[6]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf};
static unsigned char cnt=0;

unsigned char ti[][7]={"月","火","水","木","金","土","日"};
unsigned char key_dir[][5]={"上↑","下↓","左←","右→","確定"};

static void timer_init(void);

int main()
{
    unsigned char tmp[] = "Hello USB!¥n";
    int i;
    unsigned char buffer[100] = "Hello FPGA!¥n";
    unsigned char write_buffer[512], read_buffer[512];
    unsigned char buf[20]="¥0";
    int j=0;
    unsigned char jp_str[] = "よこそ、日昇へ！";
    unsigned char str[] = "        O(∩_∩)O~";

    STRING_T *string_t;
    unsigned char tmp_t;

    uart.init();
    enc28j60.initialize();
    initialize_usb();
    //日本語フォントドライバ初期化
    init_fontdrv();
    ds1302.set_time(time);
//    initialize_lcd();
//    clear();
```

```
//    draw_screen(pic);

    lcd.initialize();
    lcd.clear();

    key_init();

    timer_init();

    printf("Hello from Nios II!¥n");

    string_t->x = 0;
    string_t->y = 2;
    string_t->reverse = 0;

    printf_jp_string(string_t,"%s",jp_str);

    string_t->x = 0;
    string_t->y = 4;
    string_t->reverse = 0;

    printf_jp_string(string_t,"%s",str);

    unsigned short err;
    unsigned char dat;

    printf("¥nWriting data to EEPROM!¥n");
    //512byteのデータを書き込み、最初の256個数字が0から255、後ろの256位が1
    for(i=0; i<512; i++){
        if(i<256)
            dat = i;
        else
            dat = 1;

        iic.write_byte(i, dat);
        write_buffer[i] = dat;
```

```
        usleep(10000);
    }

//    printf("¥nReading data from EEPROM!¥n");

    //512byteのデータ読み込んでプリンタ

    for(i=0; i<512; i++){
        read_buffer[i] = iic.read_byte(i);

        usleep(1000);
    }

    uart.send_string(sizeof(read_buffer),read_buffer);

    err = 0;

    printf("¥nVerifing data!¥n");

    //データを比較して、何にか異なる所があれば、読み込み・書込みにはエラーがあった

    for(i=0; i<512; i++){
        if(read_buffer[i] != write_buffer[i])
        err ++;
    }

    if(err == 0)
        printf("¥nData write and read successfully!¥n");
    else
        printf("¥nData write and read failed!--%d errors¥n", err);

    /*----------------------------The End of IIC----------------------*/


    while(1){
```

```
        string_t->x = 0;
        string_t->y = 0;
        string_t->reverse = 0;

        if(tmp_t != get_key_value()){
            tmp_t = get_key_value();

            clear_page(string_t->y);
            clear_page(string_t->y + 1);

            printf_jp_string(string_t,"押下キー：%s",key_dir[tmp_t]);
        }

        BUZZER = 1;
        for(i=0;i<4;i++){
            LED->DATA = 1<<i;
            usleep(100000);
        }

        BUZZER = 0;

        ds1302.get_time(time);          //収集

        string_t->x = 0;
        string_t->y = 6;
        string_t->reverse = 1;

        printf_jp_string(string_t,"%02d-%02d-%02d %02d:%02d:%02d %s曜日
¥n",time[6],time[4],time[3],time[2],time[1],time[0],ti[time[5]-1]);


sprintf(buffer,"20%d-%d-%d %d:%d:%d¥n",time[6],time[4],time[3],time[2]
,time[1],time[0]);

        uart.send_string(sizeof(buffer),buffer);
```

```
        if(usb.receive_ok_flag){
            printf("%s¥n",usb.receive_buffer);

            usb.receive_ok_flag = 0;
        }

        send_string_to_usb(tmp,sizeof(tmp));

//        sprintf(buf,"%02u%02u%02u",time[2],time[1],time[0]);

        sprintf(buf,"%06u",j++);

        for(i=0;i<6;i++){
            led_buffer[i] = buf[i]-'0';
        }
    }

    return 0;
}
/*
 * ===  FUNCTION
======================================================================
 *         Name:  timer_handler
 *  Description:
 *
======================================================================
===============
 */
static void timer_handler(void *context, alt_u32 id)
{
    IOWR_ALTERA_AVALON_PIO_DATA(SEG_SEL_BASE, 0xff);
    IOWR_ALTERA_AVALON_PIO_DATA(SEG_SEL_BASE, bittab[cnt]);

    IOWR_ALTERA_AVALON_PIO_DATA(SEG_DAT_BASE, segtab[led_buffer[cnt]]);

    cnt++;
```

```
    if(cnt==6)
        cnt=0;

    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0);
}

/*
 * ===  FUNCTION
======================================================================
 *         Name:  timer_init
 *  Description:
 *
======================================================================
===============
 */
static void timer_init(void)
{
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_BASE, 0);

    IOWR_ALTERA_AVALON_TIMER_PERIODL(TIMER_BASE,200000);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TIMER_BASE,200000 >> 16);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_BASE, 0x07);

    alt_irq_register(TIMER_IRQ, NULL, timer_handler);
}
```

第八章 uCLinux 移植

8.1 移植環境準備

①Linux ホスト環境作成

uCLinux をコンパイルするため、Linux ホストが必要です、Linux ホスト環境作成は弊社の「ARM シリーズボード上 Qtopia 組込開発マニュアル」の P9 を参照ください。

②NIOS II gcc cross compiler コンパイル環境構築

■ダウンロード URL :

http://www.dragonwake.com/download/FPGA/uCLinux/nios2gcc.tar.bz2

■解凍(/opt/nios2)

作業用フォルダーを作成

[root@csun Download]# mkdir nios2

[root@csun nios2]# tar -jxvf nios2gcc.tar.bz2 -C /

■cross compiler パスを設定

[root@csun nios2]# vi ~/.bash_profile

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/opt/nios2/bin

export PATH
unset USERNAME
```

パスを直ちに有効にする

[root@csun nios2]# source ~/.bash_profile

コンパイラを正常にインストールしたかどうかを確認

[root@csun nios2]# nios2-linux-uclibc-gcc –v

下記のようなメッセージが出てきれば、正常にインストールしたことを明らかにします。

```
Reading specs from /opt/nios2/lib/gcc/nios2-linux-uclibc/3.4.6/specs
Configured with: /root/buildroot/toolchain_build_nios2/gcc-3.4.6/configure
```

--prefix=/opt/nios2 --build=i386-pc-linux-gnu --host=i386-pc-linux-gnu --target=nios2-linux-uclibc --enable-languages=c --enable-shared --disable-__cxa_atexit --enable-target-optspace --with-gnu-ld --disable-nls --enable-threads --disable-multilib --enable-cxx-flags=-static

Thread model: posix

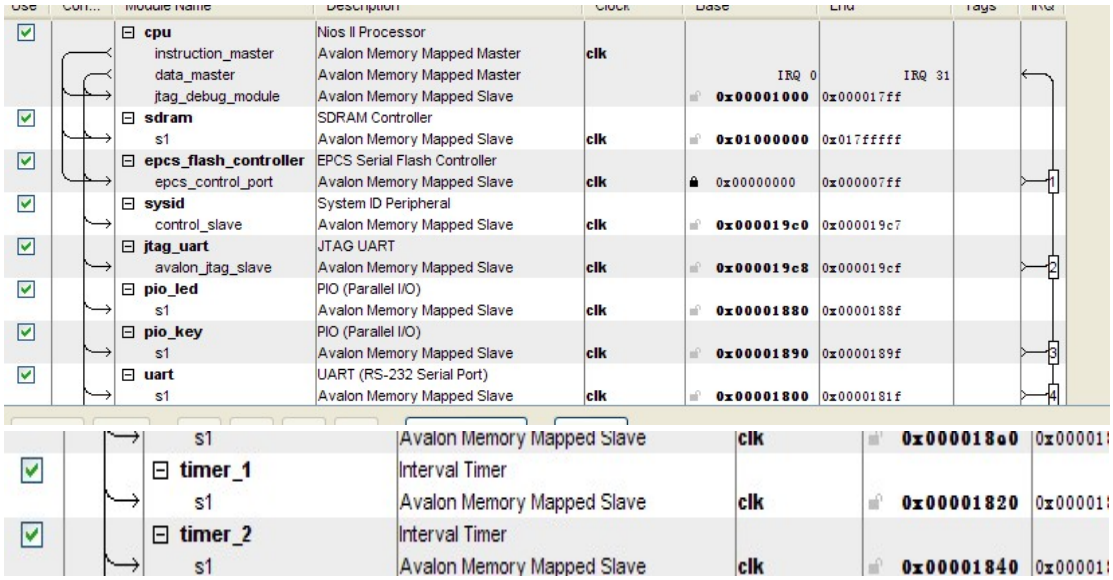
gcc version 3.4.6

8.2 ハードウエア修正

本マニュアルの前の作成した IP マクロをそのままで良いですが、下記の条件を満たす場合、uCLinux も実行できます。

①CPU、SDRAM、JTAG_UART、フル機能があるタイマー

②割り込みナンバーが 0 になりません



8.3 uCLinux コンパイル

使われる uCLinux バージョンが uClinux-dist-20070130.tar.gz です、下記 URL からダウンロードしてください。

URL1:http://www.uclinux.org/pub/uClinux/dist/uClinux-dist-20070130.tar.gz

URL2:http://www.dragonwake.com/download/FPGA/uCLinux/uClinux-dist-20070130.tar.gz

■解凍

[root@csun nios2]# tar -zxvf uClinux-dist-20070130.tar.gz

※解凍後の必要容量が 1.9G です、エラーが発生した場合、容量をチェックしてください。

■NIOS II 用パッチダウンロード

http://www.dragonwake.com/download/FPGA/uCLinux/uClinux-dist-20070130-nios2-02.diff.gz

パッチを uCLinux に適用

[root@csun nios2]# cd uClinux-dist

[root@csun uClinux-dist]# gunzip -c uClinux-dist-20070130-nios2-02.diff.gz | patch -p0

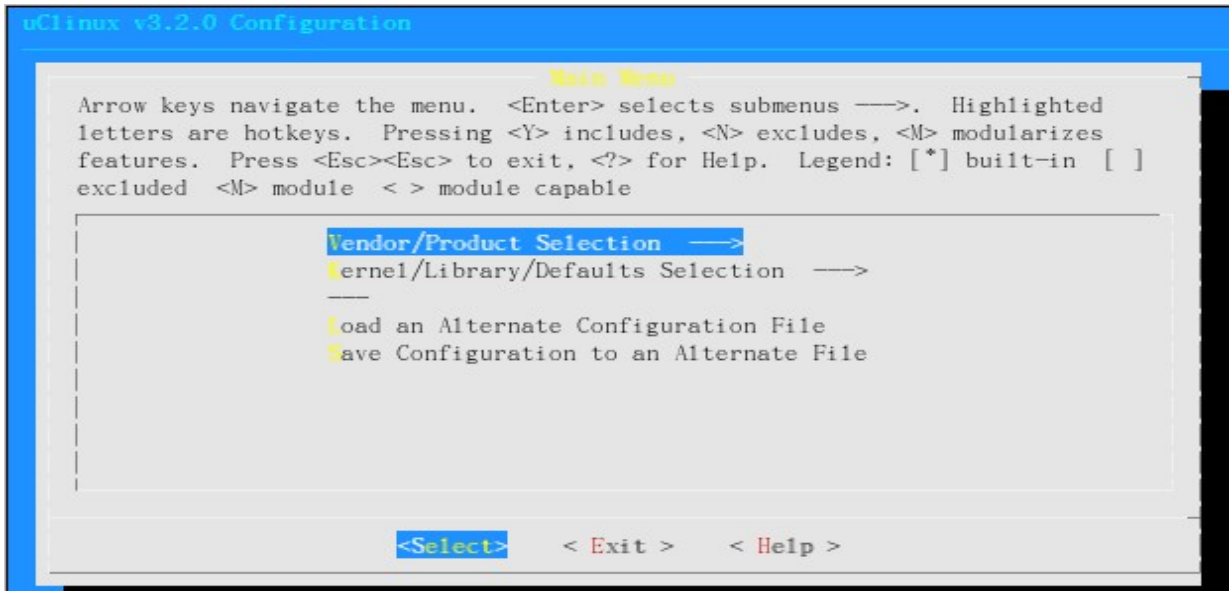
成功した場合、下記のようなメッセージが出ます。

patching file vendors/Altera/nios2nommu/config.arch
patching file vendors/Altera/nios2nommu/config.linux-2.6.x
patching file lib/libpng/Makefile
patching file linux-2.6.x/include/asm-nios2nommu/ide.h
patching file linux-2.6.x/include/linux/elf-em.h
patching file linux-2.6.x/usr/Makefile
patching file linux-2.6.x/arch/nios2nommu/kernel/vmlinux.lds.S
patching file linux-2.6.x/arch/nios2nommu/drivers/Kconfig
patching file linux-2.6.x/arch/nios2nommu/drivers/altcf.c
patching file linux-2.6.x/arch/nios2nommu/drivers/pci/Kconfig
patching file linux-2.6.x/arch/nios2nommu/drivers/pci/pci-auto.c
patching file linux-2.6.x/arch/nios2nommu/drivers/pci/pci.c
patching file linux-2.6.x/arch/nios2nommu/drivers/pci/Makefile
patching file linux-2.6.x/arch/nios2nommu/drivers/spi.c
patching file linux-2.6.x/arch/nios2nommu/drivers/Makefile
patching file linux-2.6.x/drivers/mtd/maps/altera.c
patching file linux-2.6.x/drivers/mtd/maps/Kconfig
patching file linux-2.6.x/drivers/net/Kconfig
patching file linux-2.6.x/drivers/net/Makefile
patching file linux-2.6.x/drivers/net/dm9ks.c
patching file linux-2.6.x/drivers/net/open_eth.c
patching file linux-2.6.x/drivers/net/dm9000.c
patching file linux-2.6.x/drivers/net/Space.c
patching file linux-2.6.x/drivers/net/smc91x.c
patching file linux-2.6.x/drivers/net/smc911x.c
patching file linux-2.6.x/drivers/net/mtip1000.c
patching file linux-2.6.x/drivers/usb/Kconfig
patching file linux-2.6.x/drivers/usb/host/Kconfig
patching file linux-2.6.x/drivers/usb/host/isp1362-hcd.c
patching file linux-2.6.x/drivers/usb/host/Makefile
patching file linux-2.6.x/drivers/usb/host/isp1362.h
patching file linux-2.6.x/drivers/usb/Makefile
patching file linux-2.6.x/drivers/ide/ide.c

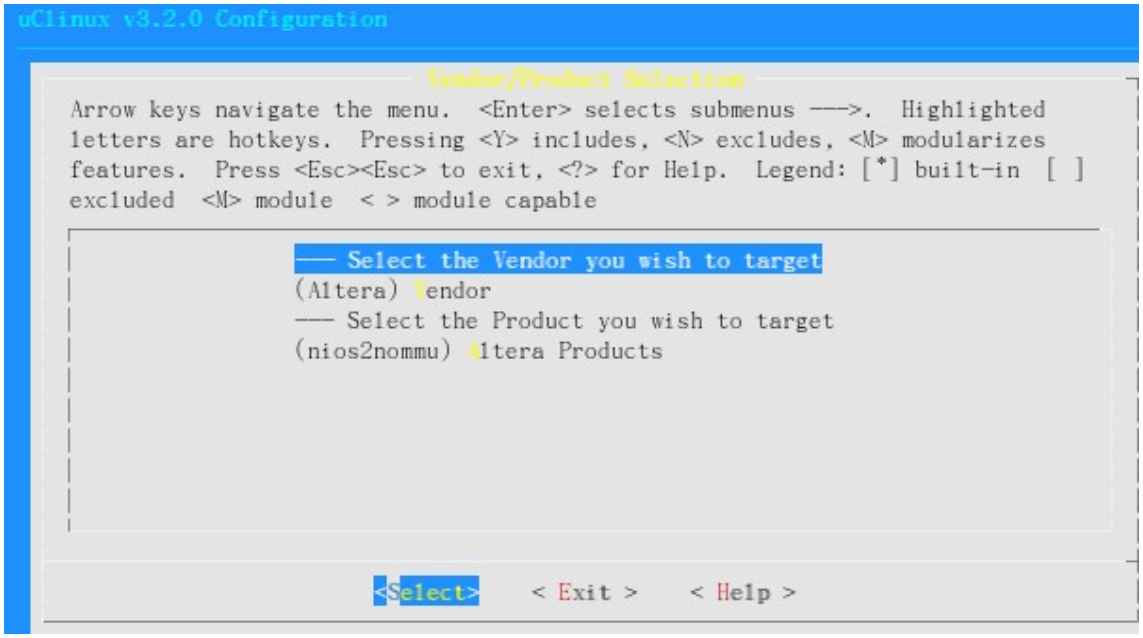
patching file user/microwin/src/fonts/X6x13.c
patching file user/microwin/src/demos/nxroach/Makefile
patching file user/microwin/src/demos/nanox/nxterm.c
patching file user/microwin/src/demos/nanox/nterm.c
patching file user/microwin/src/demos/nxkbd/keynum.c
patching file user/microwin/src/demos/nxkbd/keyctrl.c
patching file user/microwin/src/demos/nxkbd/keyshft.c
patching file user/microwin/src/Makefile.rules
patching file user/microwin/src/drivers/kbd_ttyscan.c
patching file user/microwin/src/drivers/scr_fb.c
patching file user/microwin/src/drivers/mou_ser.c
patching file user/ftpd/ftpcmd.c
patching file user/ftpd/Makefile

■uCLinux をコンフィグ

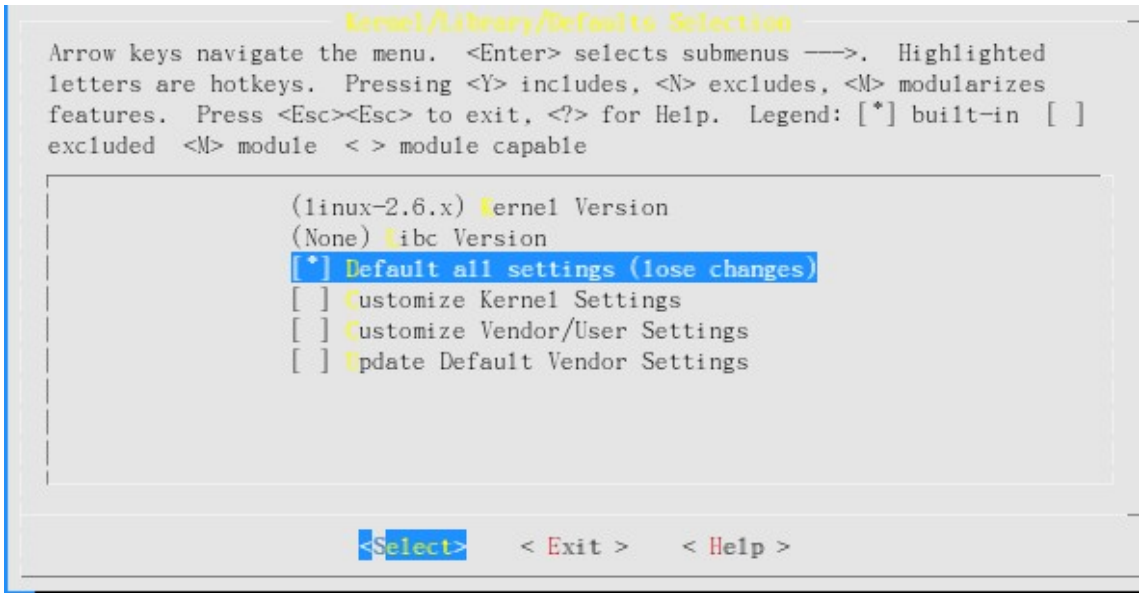
[root@csun uClinux-dist]# make menuconfig



Vendor/Product Selection を設定



Kernel/Library/Defaults Selection を設定



保存して閉じます。

■FPGA により makefile を設定

まず、Windows で Quartus II にコンパイルされた KERNEL.ptf ファイルを「/home/dwtech/Download/nios2」にコピーしてください。

次は下記コマンドを発行します。

[root@csun uClinux-dist]#make vendor_hwselect SYSPTF=/home/dwtech/Download/nios2/KERNEL.ptf

下記のような画面が出ます。

```
--- Please select which CPU you wish to build the kernel against:

(1) cpu_0 - Class: altera_nios2 Type: f Version: 7.071

Selection: 1
```

```
--- Please select a device to execute kernel from:

(1) sram_0
      Class: sram_16bit_512k
      Size: 524288 bytes

(2) sdram_0
      Class: altera_avalon_new_sdram_controller
      Size: 8388608 bytes

(3) epcs_controller
      Class: altera_avalon_epcs_flash_controller
      Size: 2048 bytes

Selection: 2
```

成功した場合、以下のようなメッセージがでます。
--- Summary using

PTF: //home/dwtech/Download/nios2/KERNEL.ptf
CPU: cpu_0
Device to upload to: cfi_flash_0
Program memory to execute from: sdram_0

--- Settings written to /home/dwtech/Download/nios2/uClinux-dist/linux-2.6.x/arch/nios2nommu/hardware.mk

make[3]: Leaving directory `/home/dwtech/Download/nios2/uClinux-dist/linux-2.6.x'
make[2]: Leaving directory `/home/dwtech/Download/nios2/uClinux-dist/vendors/Altera/nios2nommu'
make[1]: Leaving directory `/home/dwtech/Download/nios2/uClinux-dist/vendors'
■romfs 作成
[root@csun uClinux-dist]#make romfs
※エラーメッセージが出た場合、無視してください。
■uCLinux カーネルコンパイル
※他の修正必要ファイル：（解凍後：「tomodify-files」フォルダーの 5 つファイル）
http://www.dragonwake.com/download/FPGA/uCLinux/uclinux-files.zip
修正後、コンパイルしましょう。
[root@csun uClinux-dist]#make
■uCLinux カーネルイメージ生成
[root@csun uClinux-dist]#make linux image
生成されたイメージファイル：
/home/dwtech/Download/nios2/uClinux-dist/linux-2.6.x/arch/nios2nommu/boot/zImage

8.4 uCLinux 実行

上記生成された zImage と SOPC_T.sof を「G:¥02_tools¥embedded¥fpga¥QuartusII¥11.0sp1¥nios2eds¥examples」にコピーしてください。
こちら生成されたファイル：
http://www.dragonwake.com/download/FPGA/uCLinux/uclinux-files.zip
スタート—>すべてのプログラム—>Altera—>Nios II IDE 11.1—>Nios II 11.1 Command Shell をクリックして「Nios II 11.1 Command Shell」を起動させ、次のコマンドを発行
[SOPC Builder]$nios2-configure-sof SOPC_T.sof

実行結果：

```
Searching for SOF file:

in .

  SOPC_T.sof



Info: ************************************************************************

Info: Running Quartus II Programmer

Info: Command: quartus_pgm --no_banner --mode=jtag -o p;SOPC_T.sof

Info: Using programming cable "USB-Blaster [USB-0]"

Info: Started Programmer operation at Mon Jun 09 03:01:45 2008

Info: Configuring device index 1

Info: Device 1 contains JTAG ID code 0x020B40DD

Info: Configuration succeeded -- 1 device(s) configured

Info: Successfully performed operation(s)

Info: Ended Programmer operation at Mon Jun 09 03:01:46 2008

Info: Quartus II Programmer was successful. 0 errors, 0 warnings

    Info: Allocated 54 megabytes of memory during processing

    Info: Processing ended: Mon Jun 09 03:01:46 2008

    Info: Elapsed time: 00:00:02
```

■uCLinux イメージファイルを SDRAM にダウンロード

[SOPC Builder]$nios2-download -g zImage

```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00

Pausing target processor: OK

Initializing CPU cache (if present)

OK

Downloaded 1197KB in 14.4s (83.1KB/s)

Verified OK

Starting processor at address 0x00D00000
```

■uCLinux 起動

[SOPC Builder]$nios2-terminal

下記のような起動メッセージが出てきれば、uCLinux を正常起動したことです。

```
Uncompressing Linux ... Ok, booting the kernel.

Linux version 2.6.19-uc1 (root@localhost.localdomain) (gcc version 3.4.6) #2 PR

EMPT Sun Jun 8 23:28:30 CST 2008
```

```
uClinux/Nios II
Altera Nios II support (C) 2004 Microtronix Datacom Ltd.
Built 1 zonelists.  Total pages: 2032
Kernel command line:
PID hash table entries: 32 (order: 5, 128 bytes)
Dentry cache hash table entries: 1024 (order: 0, 4096 bytes)
Inode-cache hash table entries: 1024 (order: 0, 4096 bytes)
Memory available: 5956k/8192k RAM, 0k/0k ROM (1465k kernel code, 680k data)
Mount-cache hash table entries: 512
NET: Registered protocol family 16
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 1024 (order: 0, 4096 bytes)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
io scheduler noop registered
io scheduler deadline registered (default)
Serial: JTAG UART driver $Revision: 1.3 $
ttyJ0 at MMIO 0x806810f0 (irq = 1) is a jtag_uart
TCP cubic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
Freeing unused kernel memory: 572k freed (0x97a000 - 0xa08000)
Shell invoked to run file: /etc/rc
Command: hostname uClinux
Command: mount -t proc proc /proc
Command: mount -t sysfs sysfs /sys
Command: mount -t usbfs none /proc/bus/usb
mount: Mounting none on /proc/bus/usb failed: No such file or directory
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: ifconfig lo 127.0.0.1
```

```
Command: route add -net 127.0.0.0 netmask 255.0.0.0 lo

Command: cat /etc/motd

Welcome to

      ___ _ _

     / _| ||_|

  _  _| | | | _ ___ _  _ _ _

  | | | | | | || | _¥| | | |¥ ¥/ /

  | |_| | |__| || | | | | | |_| |/   ¥

  | __¥___|_||_|_| |_|¥___|¥_/¥_/

  | |

  |_|


For further information check:

http://www.uclinux.org/


Execution Finished, Exiting


Sash command shell (version 1.1.1)

/>
```

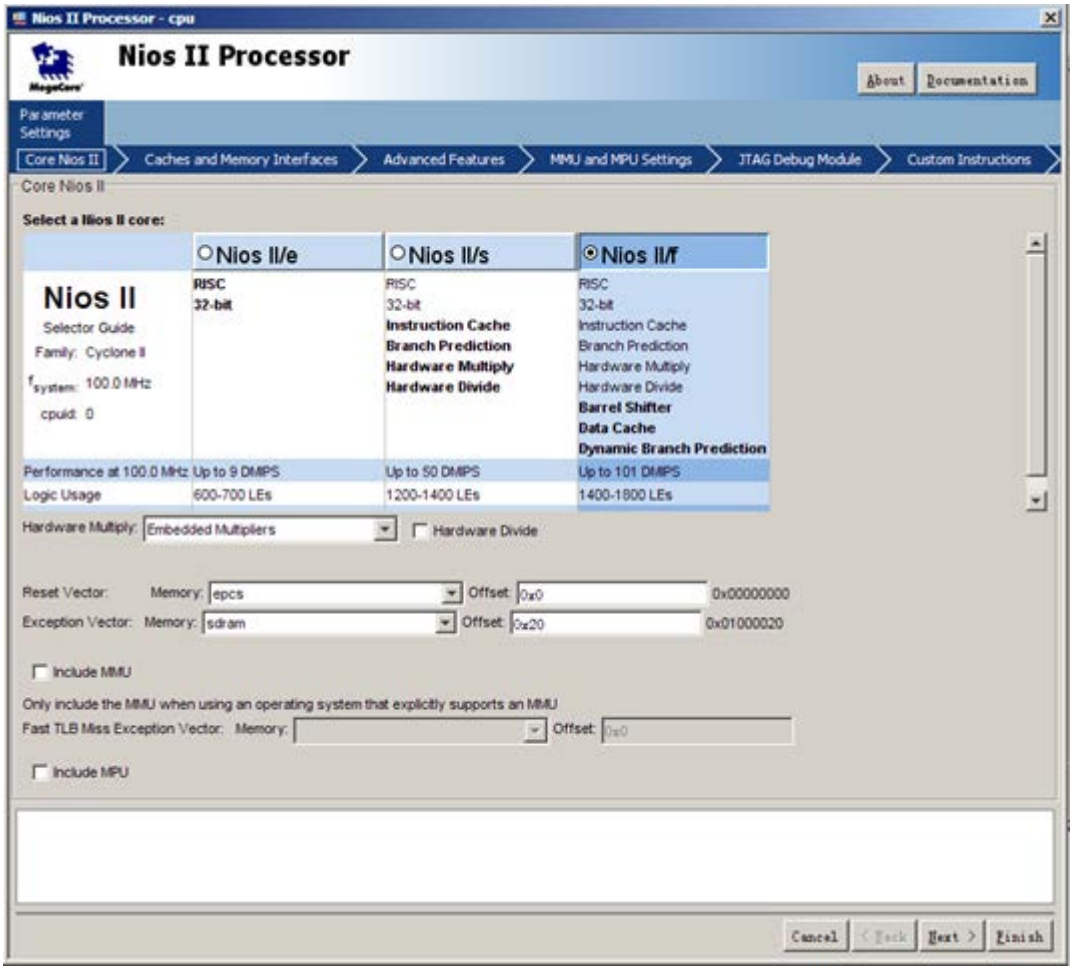
第九章 付録

付録は問題点・質問点を整理して説明します。

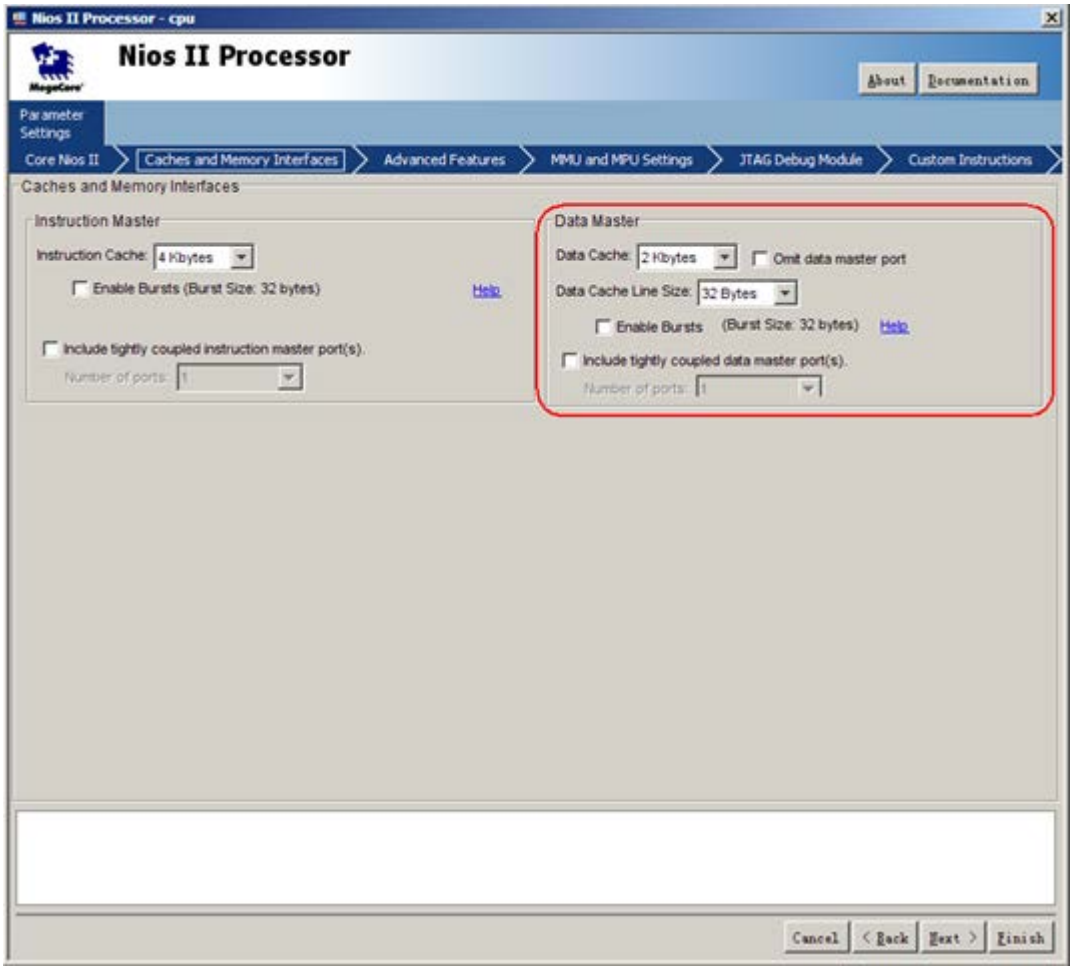
9.1 Nios II でレジスタ方式で PIO を操作できない問題解析

レジスタ直接マッピングの方法はうまく使えなく、一つ LED も点灯できないと言う現象があります、原因は下記で分析します。

CyclonII のコア基板として説明します。



上記手順には問題ありません、NIOS II/f の中、DATA CACHE（データ･キャッシュ）をチェックされます、他の 2 種類プロセッサで使えないです、NIOS II/f のみで使えます、この設定により、DATA CACHE（データ･キャッシュ）は原因を想定されます。



DATA CACHE（データ･キャッシュ）は一体どういうものですか？まず、Altera 社資料を見てましょう。(《n2cpu_nii5v1.pdf》の 2-12 ページ)

http://www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf

Cache memory resides on-chip as an integral part of the Nios II processor core. The cache memories can improve the average memory access time for Nios II processor systems that use slow off-chip memory such as SDRAM for program and data storage.

上記英文の意味は、簡単に言えば、Data Cache は PC のキャッシュと同じのものです、システム動作スピードを高めます。この Data Cache がある場合、ソースコードの実行時間は未確定になります、プログラムのリアルタイムが低くなります、例えば、プログラムから送信したデータが Data Cache に入れた後､即時実行していないと外部からプログラムが動けないように見えます。

Cache memory については詳しくは下記資料を参照

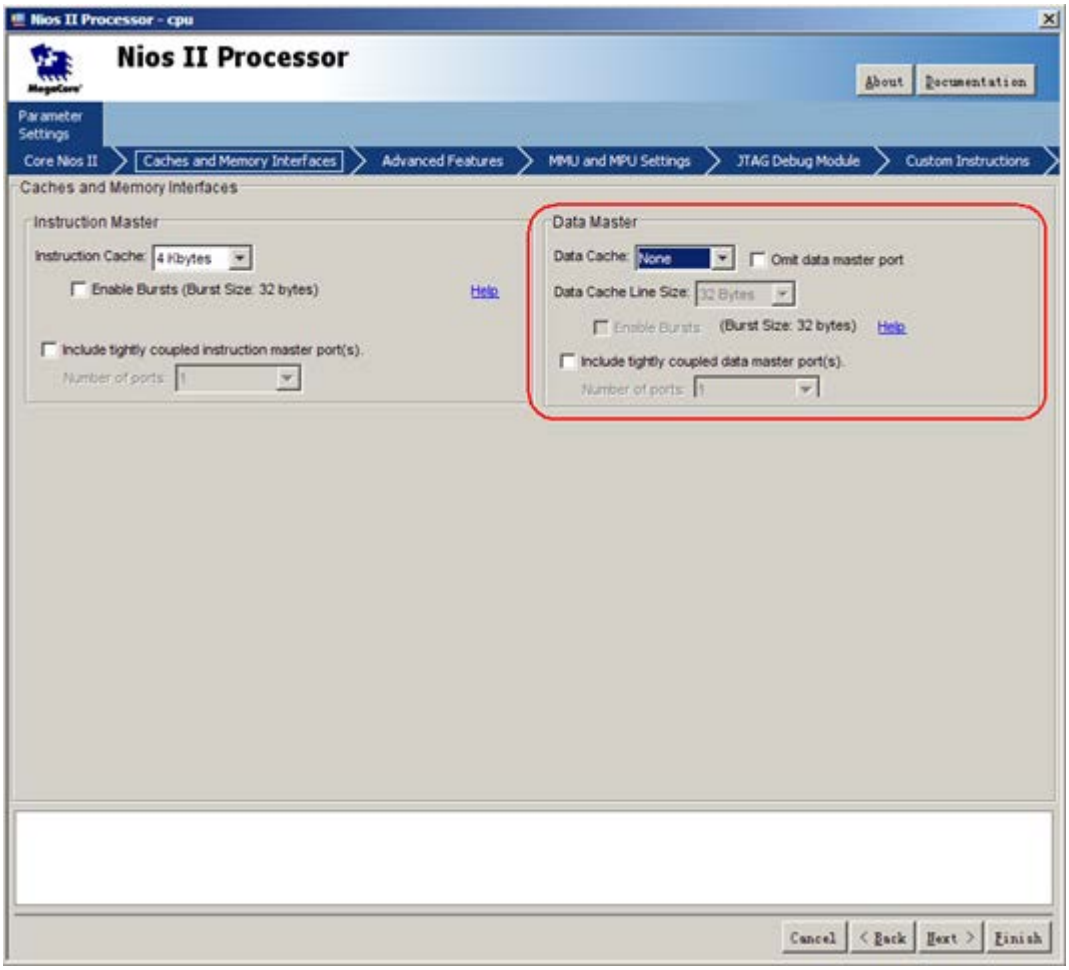
http://www.altera.com/literature/hb/nios2/n2sw_nii52007.pdf

原因を明らかにしますが、どのように解決するか？

二つ方法があります、

１．NIOS II/f を作成時に Data Cache を 0 に設定します、即ちデータキャッシュをクロズ

します。



２. Data Cache をチェックした場合、ソフトウエアに対応します。下記資料のように「31-bit Cache bypass」を使って解決します。

http://www.altera.com/literature/hb/nios2/n2sw_nii52007.pdf

説明:NIOS II でレジスタの第 31bit が Cache が有効かのコントロール位として使います、1 になる場合、Cache をクローズし使わないです、逆に 0 の場合、Cache を使います。

これにより、前の LED プログラムは下記のよう修正できます。

```
#include <unistd.h>
#include "system.h"
//ここ注意：（1<<31）を追加します。これはbypass Cacheです。
#define LED *(volatile unsigned long *)(LED_BASE | (1 << 31)
int main(void)
{
   while(1){
      LED = 1;
      usleep(100000);
```

```
        LED = 0; usleep(100000);
    }
    return 0;
}
```

9.2 レジスタの詳細分析

レジスタ問題にはよく迷ったことがあるだろう。

NIOS II 開発でこの前の方法を使ってレジスタを操作すれば、

まず、構造体を作成必要です、UART として説明します。

Offset	Register Name	R/W	Description/Register Bits													
			15:13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	rxdata	RO	Reserved					*(1)*	*(1)*	Receive Data						
1	txdata	WO	Reserved					*(1)*	*(1)*	Transmit Data						
2	status *(2)*	RW	Reserved	eop	cts	dcts	(1)	e	rrdy	trdy	tmt	toe	roe	brk	fe	pe
3	control	RW	Reserved	ieop	rts	idcts	trbk	ie	irrdy	itrdy	itmt	itoe	iroe	ibrk	ife	ipe
4	divisor *(3)*	RW	Baud Rate Divisor													
5	endof-packet *(3)*	RW	Reserved					*(1)*	*(1)*	End-of-Packet Value						

上記テーブルにより、UART は 6 個レジスタがあります（最後のレジストを一般に使わないので、構造体に追加しません）、仮にベースアドレスは 0X00 になります、これらレジスタのアドレスは 0x00,0x01,0x02,0x03,0x04,0x05 です。これらのレジスタの間に順番ああありますので、構造体を作成する時もこの順番に従わないといけないです。

前節通り作成した構造体は下記通りです。

```
/*--------------------------UART-----------------------------------------
----*/
typedef struct
{
    union{
        struct{
            volatile unsigned long int RECEIVE_DATA        :8;
            volatile unsigned long int NC                 :24;
        }BITS;
        volatile unsigned long int WORD;
    }RXDATA;

    union{
        struct{
            volatile unsigned long int TRANSMIT_DATA       :8;
```

```
        volatile unsigned long int NC                :24;
    }BITS;
    volatile unsigned long int WORD;
}TXDATA;

union{
    struct{
        volatile unsigned long int PE          :1;
        volatile unsigned long int FE          :1;
        volatile unsigned long int BRK         :1;
        volatile unsigned long int ROE         :1;
        volatile unsigned long int TOE         :1;
        volatile unsigned long int TMT         :1;
        volatile unsigned long int TRDY        :1;
        volatile unsigned long int RRDY        :1;
        volatile unsigned long int E           :1;
        volatile unsigned long int NC          :1;
        volatile unsigned long int DCTS        :1;
        volatile unsigned long int CTS         :1;
        volatile unsigned long int EOP         :1;
        volatile unsigned long int NC1         :19;
    } BITS;
    volatile unsigned long int WORD;
}STATUS;

union{
    struct{
        volatile unsigned long int IPE         :1;
        volatile unsigned long int IFE         :1;
        volatile unsigned long int IBRK        :1;
        volatile unsigned long int IROE        :1;
        volatile unsigned long int ITOE        :1;
        volatile unsigned long int ITMT        :1;
        volatile unsigned long int ITRDY       :1;
        volatile unsigned long int IRRDY       :1;
        volatile unsigned long int IE          :1;
```

```
            volatile unsigned long int TRBK        :1;
            volatile unsigned long int IDCTS       :1;
            volatile unsigned long int RTS         :1;
            volatile unsigned long int IEOP        :1;
            volatile unsigned long int NC          :19;
        }BITS;
        volatile unsigned long int WORD;
    }CONTROL;

    union{
        struct{
            volatile unsigned long int BAUD_RATE_DIVISOR    :16;
            volatile unsigned long int NC                :16;
        }BITS;
        volatile unsigned  int WORD;
    }DIVISOR;

}UART_ST;
```

構造体説明：

１．構造体は５つ共用体で構成されます、これらの共用体の順番はレジスタの順番です。

２．各共用体はさらに一つ構造体と「volatile unsigned long int」変数を構成されます、

３．共用体の中の構造体は bit 単位で構成され、bit の間にも順番があります、この順番はレジスタの仕組みにより決められます。そして、低から高までアレンジされます。NC：空くあるいは保留、操作等を行えません。

４．なぜすべて関数が unsigned long int に定義されるか？NIOS II の中、volatile unsigned long int が 32bit、volatile int と同じです、volatile long long int が 64bit です。上記レジスタテーブルに 16bit を示されますが、32bit に定義しました。これは NIOS II アドレスのアライメントのためです。システム上にデータ幅が異なるインタフェースがある場合、アドレスのアライメントを考えないといけないです。アドレスのアライメントは 2 種類を分けられます、一つは静的アドレスアライメント、もう一つは動的アドレスアライメントです。メモリ周辺デバイス等は動的アドレスアライメントを使います、レジスタは逆に静的アドレスアライメントを使います。理由は、アドレスのアライメントの特徴により決められます。静的アドレスアライメントはマスター側の転送がスレーブ側の転送と一致します、動的アドレスアライメントはマスター側の一つ転送でスレーブ側から複数の転送と対応します。

上記構造体のプログラムでレジスタが volatile unsigned long int 種類に定義される理由は

静的アドレスアライメントと繋がります。UART が 16bit となりますが、NIOS II が 32bit です。UART が NIOS II と通信の時、NIOS II の 32bit の低 16bit は有効に利用されますが、高 16bit でもアドレスを割り当てられます、即ち UART の 16bit が実際 32bit アドレスを使います。

５．共用体と中身の「volatile unsigned long int WORD」変数の作成理由

共用体の特徴はメンバーは同じメモリを使います、即ち、bit 単位で構成される構造体は WORD と同じメモリを使います。種類は両方も volatile unsigned long int になります、そうすると、構造体の各 bit メンバーは WORD の 1bit と一致します、WORD のある bit を操作したい場合、構造体の該当 bit を操作すればよいです。

例えば、

```
UART->CONTROL.BITS.IRRDY=1;//受信準備完了、割り込みが有効
```

レジスタ全体をクリアしたい場合、

```
UART->STATUS.WORD=0;//ステータスレジスタをクリア
```

9.3 NIOS II についてのよくある QA

１．Q：浮動小数点演算を NIOS 上で何ができるか？

A：NIOS 上浮動小数点演算ですることができます、完全に MCU を置き換え、クロック周波数が 100MHZ になっても良いです。一般的に、ARM7 のクロック周波数が 72MHZ になります、ARM7 より早いです。

２．Q：NIOS II で SDRAM とパラレル Flash を使わなくてもよいか？

A：プログラムは SDRAM で実行します、Flash にはソースコードを保存します。(SDRAM が電源切断後、データが無くなった、Flash が保存したらこの現象がない)

毎回電源を入れたら Flash からプログラムを読み込み SDRAM に置き、それでプログラムを実行します。FPGA 内部のメモリ（チップメモリ）は小さいです、大きなプログラムで直接実行する事が出来ないので、SDRAM を拡張します。SDRAM は不可欠ですが、パラレル Flash はシリアル Flash に置き換えても良いです、本開発ボードに EPCS1（4,16...）はシリアル Flash です、ソースコードも保存できます、そして、少ないピンだけ必要です。

３．Q：Quartus II でハードウエアを成功にコンパイルしましたが、ダウンロード時に下記エラーメッセージがあります？

Info: Started Programmer operation at Thu May 06 01:39:46 2010

Error: Application Nios2 on 127.0.0.1 is using the target device

Error: Operation failed Info: Ended Programmer operation at Thu May 06 01:39:46 2010

A：このエラーメッセージは、JTAG モードで NIOS の JTAG 機能（例：BLASTER でオンラインエミュレータ）を使うとともに、*.sof ファイル（JTAG モードでダウンロード）をダウンロードしたい場合に発生します。

簡単に言えば、JTAG が既に使われます。解決策は使われている JTAG 機能をクローズして*.sof ファイルをダウンロードします。

４．Q：プロジェクトを新規作成時に下記エラーが発生します。

The software setting(STF)file associated with this project is damaged.
This may be fixed by copying your source files into a new c/c++ application project.
For more details see the error log.

A：このエラー原因はプロジェクト保存パスの中、英語以外のパス、スペースパスがあります、NiosII IDE は英語以外の文字コードをよくサポートしていないためです。ディレクトリ名は全て英語に変更、かつスペースを削除して解決できます。

５．Q：プログラムを実行時に下記エラーが発生します。

Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Pausing target processor: not responding.
Resetting and trying again: FAILED
Leaving target processor paused

A: この問題は NIOS の IP マクロのリセット（RESET）ピンを追加しないため、あるいは追加されたピンが FPGA のピンと不一致のため起きます。

６．Q：ダウンロード時、下記エラーが発生します。

Error: Can't configure device. Expected JTAG ID code 0x020010DD for device 1,
but found JTAG ID code 0x020B40DD.

A：SOPC で選べたデバイス或いはインタフェースは開発ボードと違います。

７．Q：NIOS II にサンプルプロジェクト「hello_world」をコンパイル時、下記エラーが発生します。

gdrive/c/altera/kits/nios2/components/altera_nios2/HAL/src/alt_busy_sleep.c:68:
error: parse error before '/' token

A：これはインストールされている Quartus と NIOS のバージョンが異なる為のエラーです。

８．Q：SOPC で「Generate」をクリックすると、下記エラーが発生します。

Error: Generator program for module 'epcs_controller' did NOT run successfully.
SOPC に epcs_controller を追加すれば、上記エラーが発生します、部品を作成できません。

A: これはインストールされている Quartus と NIOS のバージョンが異なる為のエラーです。

９．Q：Quartus II に下記コンパイルエラーが発生します。

Error: Can't place pins assigned to pin location Pin_AE24 (IOC_X65_Y2_N2)

A:二つピンは両方も PIN_AE24 に割り当てられます、一個削除すれば解決できると思います。

１０．Q：NIOS II IDE に変数あるい関数の定義をどのようにトレースしますか？

A:CTRL キーを押しながら、マウスはトレースしたい関数あるいは関数の所、自動にハイライトで表示されます、クリックすれば定義が見れます。

１１．Q：Avalon Tristate Bridge を SOPC で追加する時、下記エラーが発生します。

Tri state bridge/tristate master requires a slave of type Avalon tristate. Please add a slave of type Avalon tristate.

Generate ボタンがグレーになってコンパイルできません。

A：AND ゲートと接続するトライステートブリッジデバイスが必要です、Flash を SOPC に追加すれば解決できと思います。

１２．Q：count_binary というサンプルを作成する時、下記エラーが発生します。

error: `BUTTON_PIO_IRQ' undeclared (first use in this function)

A:このエラー原因は、SOPC Builder で作成した PIO モジュールの名前はプログラムの名前と異なることです、PIO モジュールを「button_pio」に変更してください。

１３．Q：NIOS II で下記のコンパイルエラーが発生します。

region ram is full (count_binary.elf section .text). Region needs to be 24672 bytes larger. address 0x80c1f8 of count_binary.elf section .rwdata is not within region ram. Unable to reach edge_capture (at 0x00800024) from the global pointer (at 0x0081419c) because the offset (-82296) is out of the allowed range, -32678 to 32767

A：Onchip memory を使う場合、上記エラーが発生します。FPGA 内蔵メモリが小さいですから、作った NIOS II プログラムを実行する時、メモリリークが発生します。この場合、SDRAM を追加してもう一度実行してみてください。

１４．Q：NIOS II IDE にプロジェクトの System Library のオプションの意味は？
.text .rodata .rwdata が reset .exception とどういう関係がありますか？

A：.text：ソースコードエリア

.rodata：リードのみエリア、静的なグローバル変数を保存

.rwdata：書込み・読み込みエリア

.bss：初期化しない変数を保存
.text – the actual executable code
.rodata – any read only data used in the execution of the code
.rwdata – where read/write variables and pointers are stored heap – where dynamically allocated memory is located stack – where function call parameters and other temporary data is stored

１５．Q：NIOS II IDE でコンパイルできたプログラムをデバッグ時、下記エラーは発生します。

Using cable "ByteBlasterII [LPT1]", device 1, instance 0x00 Processor is already paused Downloading 00000000 (0%) Downloaded 57KB in 1.2s (47.5KB/s) Verifying 00000000 (0%) Verify failed Leaving target processor paused

A：下記いくつか原因を想定されます。
①SDRAM のタイミングが正しくないです。ある時正しくない pll clock phase shift for sdram_clk_out であれば、SDRAM が正常に動作出来ないことを招きます。
②SDRAM のピンと接続が間違います（開発ボード上に物理的に接続が間違う可能もある）
③デバッグ時、ダウンロード用のメモリは不揮発性メモリ（non-volatile memory）ではありません、あるいはメモリ容量が不足でこのエラーが発生します。
④Quartus II のデフォルトの設定で発生したエラー、デフォルトでは Quartus II にすべて使わない IO ポートをグランドに接続します、この場合、ある部品が正常に動作出来ない可能性があります、IO ポートをスリーステートに設定した方がよいです。
⑤USB-Blaster が壊れた、或いは JTAG の通信ノイズが大きすぎ、アンチジャミングのため、JTAG 側には抵抗値が小さいプルアップ抵抗が必要です。
⑥SDRAM は CPU の指令バス及びデータバスと両方接続する事を確保します。

１６．Q：SOPC で 200KB の onchip_memory を追加した場合、Quartus II でコンパイル時に下記エラーが出ます。

Error: Selected device has 105 RAM location(s) of type M4K RAM. However, the current design needs more than 105 to successfully fit

A：SOPC の onchip_memory は M4K RAM と違います、Quartus II に上記コンパイルエラーは設計上に使われている M4K が多すぎです。

１７．Q：sopc-builder の reset address 設定を詳しく説明してほしい

A:SOPC の reset address に設定される場所は最終的にプログラムダウンロードの所です、そして、プログラムは reset address から起動します。SOPC の exception address に設定される場所にはシステム異常処理コードを保存します、もし、reset address が exception

address と違う場合、プログラムは reset address から起動後、reset address に保存されている異常処理コードを exception address にコピーします。
■text address に設定される場所はプログラム実行の所です、もし、reset address が text address と違う場合、プログラムは reset address から起動後、reset address に保存されている読み取り専用のプログラムコードを text address にコピーします。
■rodata address に設定される場所には読み取り専用のデータを保存されます、もし、reset address が rodata address と違う場合、プログラムは reset address から起動後、reset address に保存されている読み取り専用のデータを rodata address にコピーします。
■rwdata address に設定される場所には読み取り・書込みできるデータを保存されます、もし、reset address が rwdata address と違う場合、プログラムは reset address から起動後、rwdata address に保存されるデータの初期化を行います。

１８．Q：NIOS II パフォーマンスをどのように向上しますか？
A：下記何点からパフォーマンスを向上可能です。
①fast CPU を使います。
②システムのクロック周波数を引き上げます。
③プログラムを先に SRAM、次に SDRAM、最後 Flash で実行します。
④チップ内の RAM をデータバッファとして使い、次は SRAM、最後 SDRAM を使います。
⑤IO データ転送は出来るだけ DMA を利用します。
⑥並行処理できるデータについては複数 CPU で共同処理
⑦典型的なアルゴリズムはユーザコマンドにします、ユーザーコマンドは 256 個までできます。
⑧HDL モジュールを周辺デバイスとして使ってください。
⑨C2H を使います。

１９．Q：下記のような問題がよくでます。
Verify failed between address 0xxxxxx and 0xxxxxx Leaving target processor paused
A：
①address の後ろの二つアドレスにより一体何のデバイスにエラーが発生したかを明確します。一般に、エラーが発生したのはメモリ関連ものです。明確方法は SOPC 中のアドレスあるいは system.h 中のアドレスにより、該当のデバイスを見つけます。
②ハードウエアの溶接が正しいかをチェックします、多くの問題はハードウエアの溶接から置きました。特に手作りボードに対して、アドレスデータバスに何にか溶接問題があれば、「verify failed」というエラーが発生します。
③SOPC の component が正しいかをチェックします、特にカスタマイズのインタフェースに対してチェック必要です。

④QuartusII の設計を確認：
■ピンロックが正しいかをチェックします、必ず 1 対 1 になるべきです。
■アドレスアライメント問題について、8、16、32bit の外部メモリデバイス等の最下位アドレスは 0、1、2 になるはずです、即ち、16bit の外部メモリデバイスを使う場合、最下位は ADD[1]になります、ADD「0」を使わないです、他の b it の場合も同じルールでチェックしてください。
■データバスは必ず双方向の IO です。
■SDRAM の場合、位相ロックループ PLL の計算と設定必要です。
⑤NIOS II IDE にオプション設定が正しいかをチェックします。

２０．Q：SDRAM と FLASH が両方もある SOPC の中、もし、最終的に FLASH にプログラムをダウンロードしたら、特に NIOS II IDE の Flash Programmer 機能を使ってダウンロードする時、下記エラーが発生します。

Programming flash with the project "$SOPC_KIT_NIOS2/bin/nios2-flash-programmer" --base=0x03000000 --cable='ByteBlasterII [LPT1]' --sidp=0x04001038 --id=417604522 --timestamp=1257256685 "ext_flash.flash" Empty flash content cannot be programmed or verified

A：予想の原因は RESET VECTOR が SDRAM にポイントされます、Flash にポイントされるべきです。

２１．Q：ダウンロード時、下記エラーが発生します。

Using cable "USB-Blaster [USB-0]", device 1, instance 0x00 Resetting and pausing target processor: OK Reading System ID at address 0x00201040: verified No CFI table found at address 0x00000000 Leaving target processor paused

A：これは FLASH 設定の原因です、注意必要の所：①データバス：双方向　②16bit と 8bit のモジュールを明確　③SOPC Builder の中、Flash 型番により多々しくパラメータを設定、パラレル Flash の場合、トライステート･ブリッジを設定必要です。