Nios® II 入門編トライアル・コース
【演習マニュアル】

Ver.18.1

2019 年 7 月 Rev.4

Nios® II 入門編トライアル・コース 【演習マニュアル】

目次

Nios® II 入門編トライアル・コース 【演習マニュアル】

1. はじめに

この資料は、 Nios® II のシンプルなデザインを作成する演習を通じて、開発ボード上で簡単な動作確認が行えるまでの手順を示した資料です。

また、一般的な組み込みシステムと、 Nios® II システムを比較しながら、デザイン開発フローとの関連性を持たせることで、組み込み系エンジニアとロジック設計エンジニアの双方が互いに理解できることを目的としています。

今回の演習では、次のようなシステム構成を題材にしています。

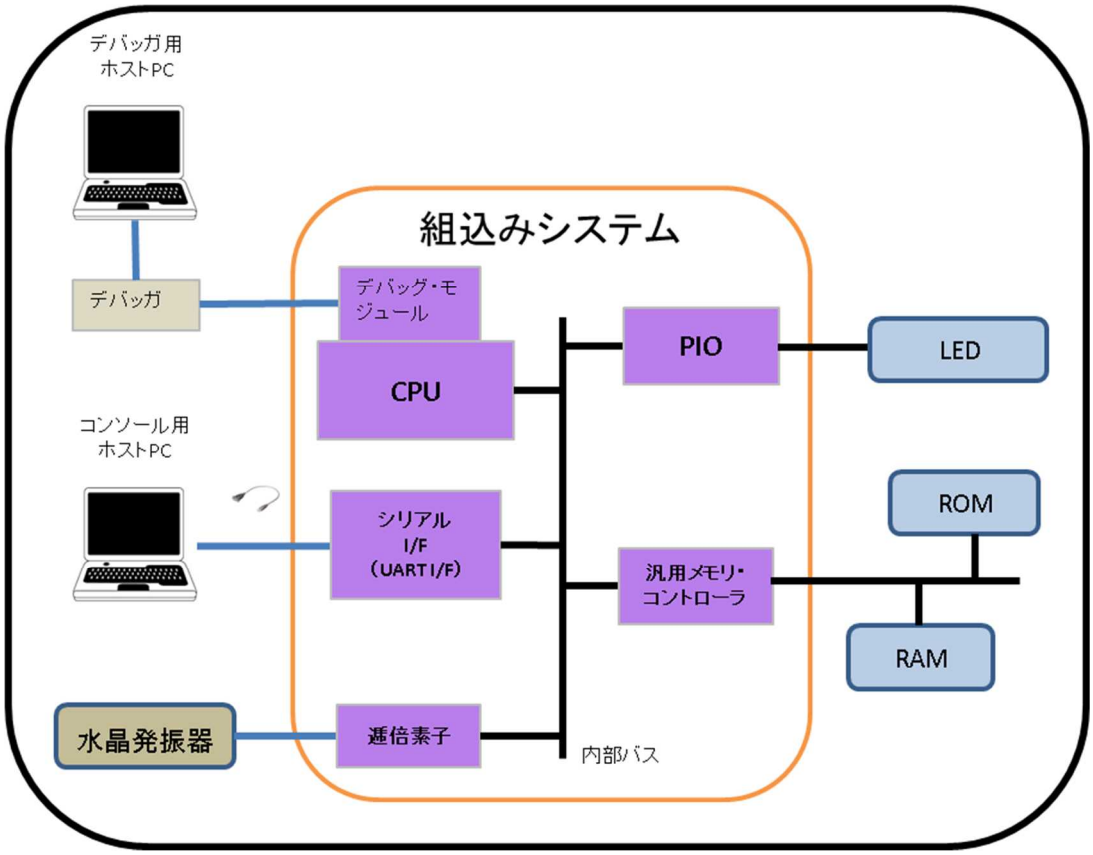


図 1 演習題材のシステム構成（一般的な組み込みシステムから見た場合）

一般的な組み込みシステムのソフトウェア開発では、ターゲットの CPU ボードに対して、コンソール用ホスト・マシンの実行ファイルを RAM 上にダウンロードしてソフトウェアを実行します。ダウンロードや簡単なデバッグ・コマンドは、シリアル・ポートやイーサーネットを経由して行われます。また、デバッグ用ホスト・マシンからデバッガを介して、CPU デバッグ・モジュール経由での高度なデバッグを行うこともできます。

図 1 では、逓倍した入力クロックを CPU の動作クロックやシステム・クロックに適用し、 ROM をソフトウェア・プログラムの格納場所に使用します。そしてワーク・メモリには RAM を使用します。

今回の演習では、 Parallel I/O (PIO) を経由して、外部の LED を点滅させるソフトウェア・プログラムを作成しますが、これらの周辺ペリフェラルは CPU 専用内部バスで接続されます。

ROM や RAM と接続するメモリ・インタフェースは、汎用メモリ I/F ブリッジを経由して、 CPU 内部バスと接続されます。

前ページのシステム構成をインテル® FPGA の開発ツールで実現した場合、図 2 のようなブロック構成になります。

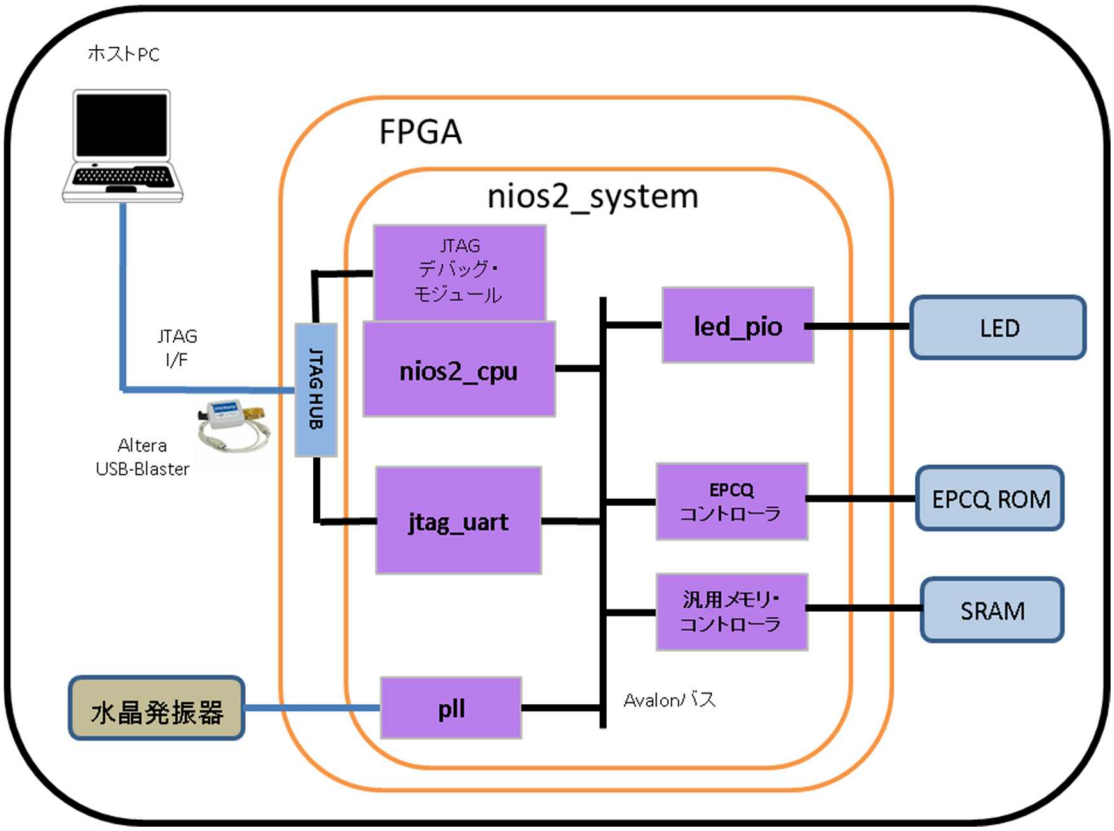


図 2 演習題材のシステム構成（Nios® II システムから見た場合）

この場合、 JTAG_UART というインテル® FPGA 特有のシリアル・ポートを利用することで、ホストとのシリアル通信をインテル® FPGA ダウンロード・ケーブル（USB-Blaster™ 等）を使用して、 JTAG ポート経由で行うことができます。これによりソフトウェア・プログラムは、 JTAG 経由で RAM 上にダウンロード後、実行されます。JTAG デバッグ・モジュール経由でのデバッグについても、ダウンロード・ケーブルを介して行われます。

入力クロックは、図 2 のように、 FPGA 内蔵の PLL で逓倍することもできます。 EPCQ デバイス（インテル® FPGA シリアル・コンフィグレーション・デバイス）をプログラムの格納場所に使用して、ワーク・メモリには SRAM を使用します。

今回の演習では PIO を経由して、外部の LED を点滅させるソフトウェア・プログラムを作成しますが、これらの周辺ペリフェラルは Nios® II プロセッサ用内部バスである Platform Designer インタコネクトで接続されます。

EPCQ デバイスや SRAM と接続するメモリ・インタフェースは、それぞれコントローラを経由して、Platform Designer インタコネクトと接続されます。

図 2 の組み込みシステムは、システム名 nios2_system というブロックとして FPGA 内に実装されます。

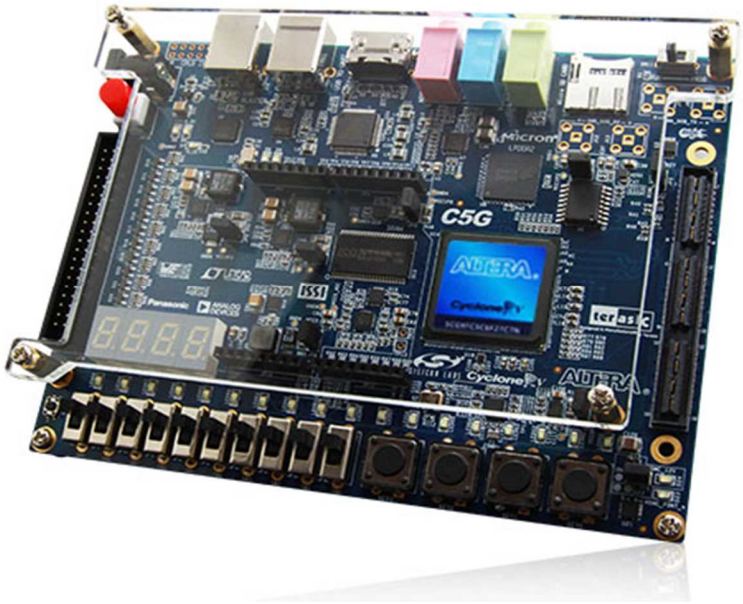
2. 演習で使用する評価ボード

今回の演習では、 Terasic 社の Cyclone® V GX スタータ開発キットを使用します。

このボードの構成は、 4 個のプッシュ・ボタンと 18 個の汎用 LED、 10 個のスライド・スイッチ、4 個の 7 セグ LED を実装しており、シンプルかつ高機能な動作が確認できます。 USB-Blaster™ の本体部分が基板に実装（Embedded USB-Blaster）されておりますので、付属の USB ケーブルのみでホスト PC と、基板上 FPGA や Nios® II プロセッサ間の通信が可能です。また、EPCQ デバイスと揮発性 RAM として LPDDR2-SDRAM, SRAM を実装しております。クロック・システムは本演習では、 50MHz の水晶発振器を利用し、 SRAM や EPCQ デバイスに対しては、 FPGA 内蔵の PLL で逓倍したクロックを利用しています。その他、詳細については、以下の説明ページを参照ください。

Terasic 社の Cyclone V GX スタータ開発キット

https://www.intel.co.jp/content/www/jp/ja/programmable/products/boards_and_kits/dev-kits/altera/kit-terasic-cyclone-v-gx-starter.html



3. 開発ツールとデザイン開発フロー

開発フローと開発ツールとの関係を示します。

3-1. 組み込みシステム・ハードウェアおよび FPGA デザインの開発フロー

① デザイン・エントリ

- ハードウェア記述言語（Verilog HDL / VHDL）プロジェクトに登録します。回路図や EDIF ネットリストでもプロジェクトに登録できます
- 組み込みシステム部分のハードウェア記述言語は、 Platform Designer が生成します
- タイミング制約は、 SDC ファイルとして生成しプロジェクトに登録します

② 論理合成

- ハードウェア記述言語から論理回路を合成します
- 3rd パーティーの論理合成ツールや Quartus® Prime（Synthesis）を実行します
- 結果は、専用のネットリストに反映されます

③ 配置配線

- 合成した論理回路に対して、対象の FPGA の配線経路の情報を加味します
- Quartus® Prime（Fitter）を実行します
- 結果は、専用のネットリストに反映されます

④ タイミング解析

- TimeQuest Timing Analyzer を使用して、登録した SDC ファイルの内容に基づきタイミング解析を行います

⑤ アセンブル

- Quartus® Prime（Assembler）を実行します
- 配置配線結果をビットストリーム（プログラミング・データ）に展開します
- このビットストリームが、ターゲットのデバイスに実装されます
- インテル® FPGA のビットストリームには、 sof ファイルと、 pof ファイルとが存在します
- ターゲットのデバイスが揮発性素子（FPGA）の場合、 sof が適用されます
- ターゲットのデバイスが不揮発性素子（ROM や CPLD）の場合、 pof が適用されます

⚠ *ソフトウェア・ツールチェーンのアセンブリ言語とは関連がありません*

⑥ プログラミング

- ターゲットのデバイスに対して、ビットストリームを書き込みます
- Quartus® Prime Programmer を使用します

⑦ 実機動作

3-1-1. 組み込みシステム・ソフトウェアの開発フロー

① ソフトウェア・プログラムの作成

- ソフトウェア開発環境やテキスト・エディタを使用します
- 今回の演習では、 Nios® II SBT のエディタを使用して、プログラミング言語を記述します

② C/C++ コンパイル

- C/C++ コ ンパイラが、プログラミング言語をアセンブリ言語に変換します
- 今回の演習は、 Nios® II SBT に組み込まれている GNU ベースの C/C++ クロス・コンパイラを使用します

③ アセンブル

- アセンブラが、アセンブリ言語をオブジェクト・ファイルに変換します

  ⚠ *FPGA 開発フローの「アセンブル」とは異なります*

④ リンク

- リンカが、ライブラリ情報をオブジェクト・ファイルにリンクします
- 様々な中間ファイルを経て、最終的に実行ファイル（ELF ファイル）を生成します
- Nios® II SBT では、 ELF ファイルは、拡張子 .elf で表記されます

⑤ ダウンロード・ファイル（ROM 化ファイル）の生成

- バイナリ・ユーティリティが、 ELF ファイルをモトローラ S レコード・フォーマットに変換します
- Nios® II SBT では、 S レコード・ファイルを、拡張子 .flash で表記します

  ⚠ *上記のフローの ① ～ ④ は、 makefile の記述内容に従い、一括処理されます*

  ⚠ *この一括処理は、 Nios® II SBT で「ビルド」を行うことにより実現されます*

⑥ ダウンロード、および実機動作

- Nios® II SBT が、 JTAG ポートや JTAG_UART ペリフェラルを経由して ELF ファイルを RAM 上に格納した後、プログラムを実行します

4. ハードウェア・プロジェクトの作成

Quartus® Prime を用いてハードウェア・プロジェクトを作成します。 Nios® II の開発を行う際は、必ず、ハードウェア・プロジェクトを作成する必要があります。

今回の演習では、 nios2_basic_lab という名称のハードウェア・プロジェクトを作成します。また、開発環境は、Quartus® Prime v18.1 + Nios® II SBT v18.1 を使用します。

⚠ *演習に使用するファイルはすでに解凍されておりすぐに演習が開始できる環境になっています。講師の指示に従い演習を進めてください。もしファイルが未解凍である場合は、講師の指示に従いファイルの解凍を行ってください。*

4-1. Quartus® Prime の起動

デスクトップのショートカット、もしくはスタートメニューより Quartus® Prime を起動します。

4-2. プロジェクトを作成

1. Quartus® Prime のメニュー・バーから **File ⇒ New Project Wizard** を選択します。 New Project Wizard が開きます。

   Introduction ページが表示された場合は **[Next]** ボタンをクリックします。

2. プロジェクト・フォルダを指定します。最上段のボックスに以下のパスを入力します。

   C:¥Lab¥nios2_lab¥nios2_basic_prj

   ⚠ *Quartus® Prime プロジェクトのパス名には、スペースや日本語文字は入れないでください。 Nios® II SBT をインストールする際のインストール・パスについても同様です。*

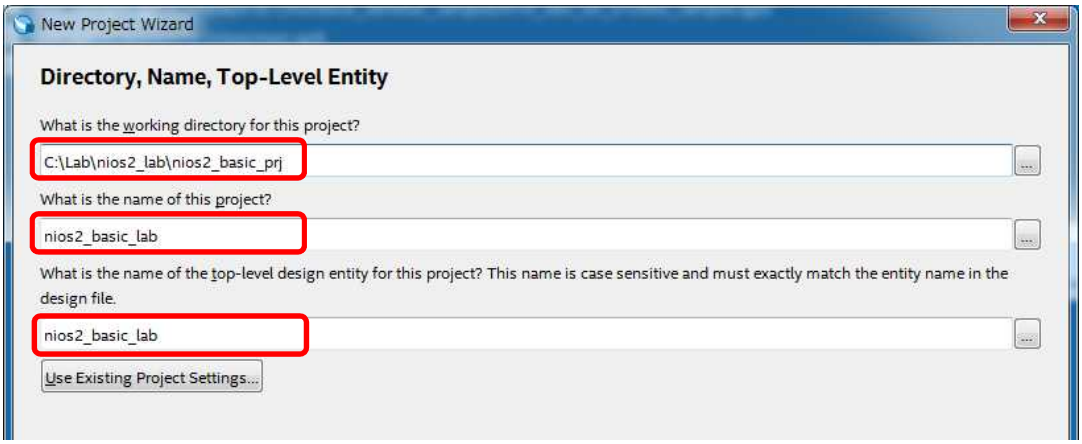
3. プロジェクト名を指定します。二段目のボックスには以下のプロジェクト名を入力します。

   nios2_basic_lab

4. トップ階層を指定します。三段目のボックスに以下のプロジェクト名が反映されていることを確認します。

   nios2_basic_lab

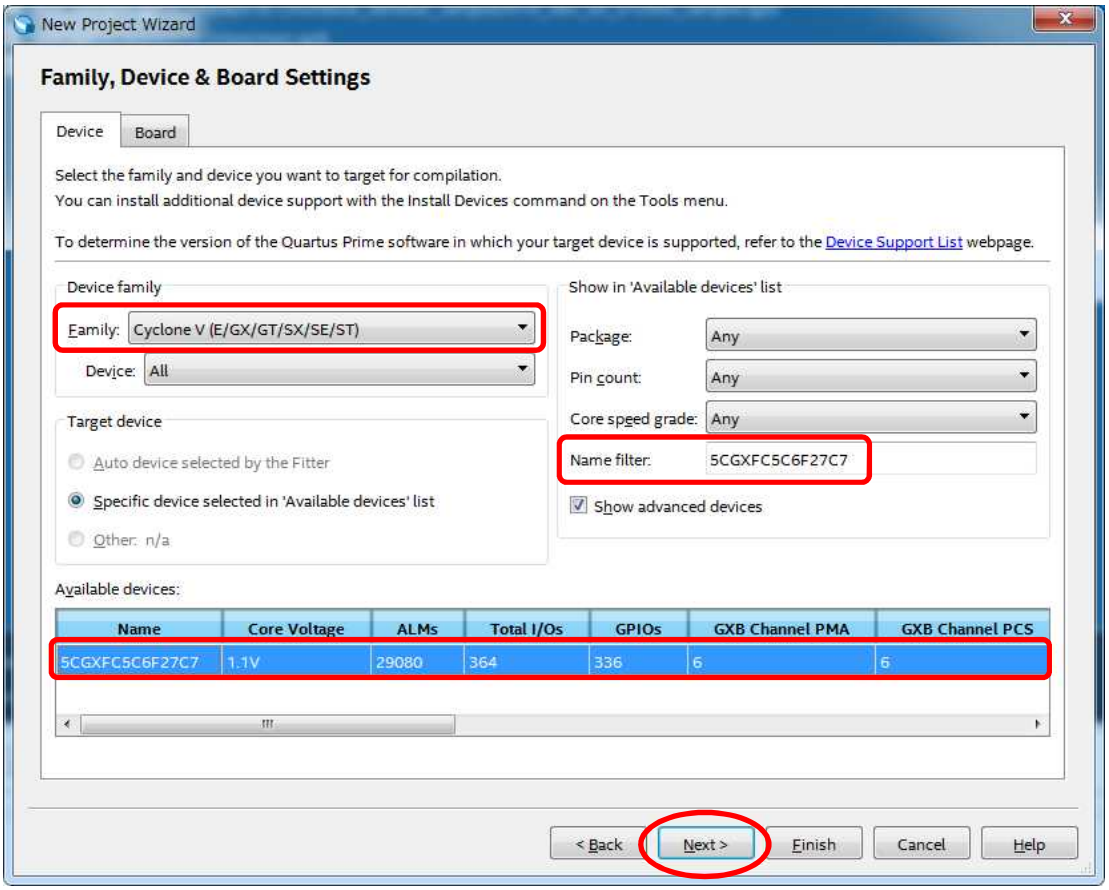
5. 右下の **[Next]** ボタンをクリックします。



6. Project Type が開くので、 **Empty Project** が選択されていることを確認して **[Next]** ボタンをクリックします。
7. Add Files が開くので、何もせず **[Next]** ボタンをクリックします。
8. Family & Device Setting が開くので、デバイス・ファミリを選択します。

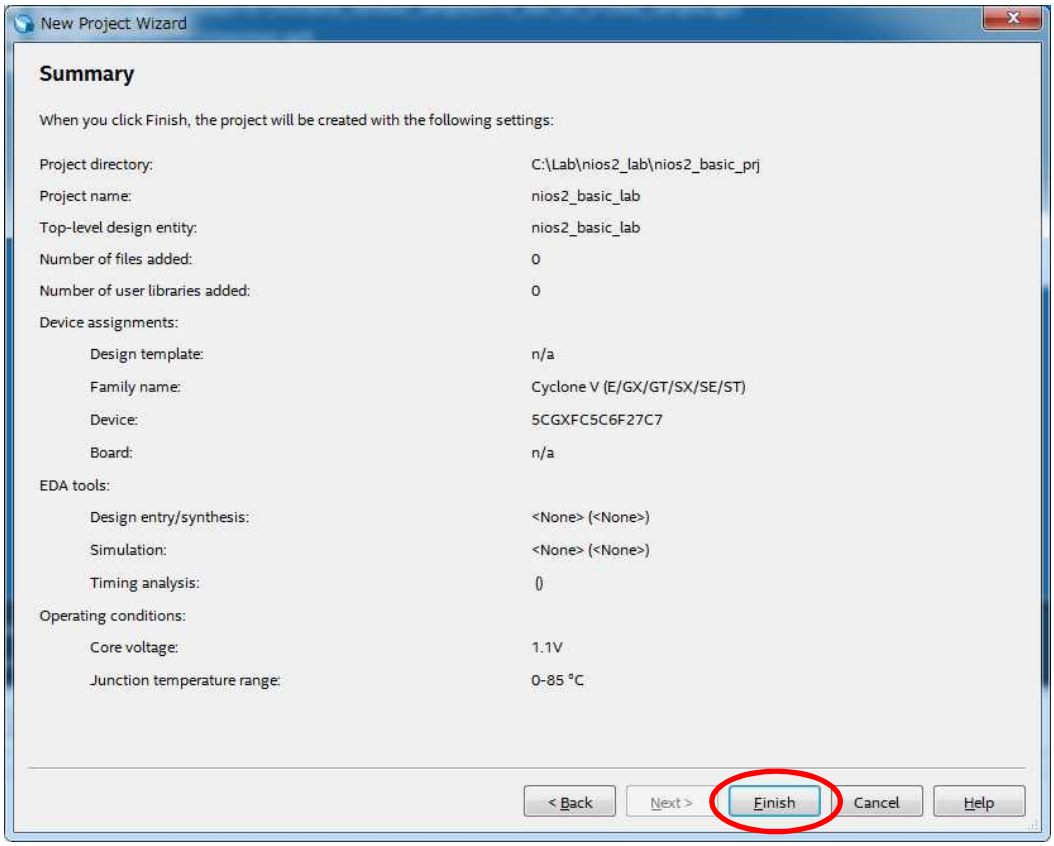
   Cyclone V (E/GX/GT/SX/SE/ST)
9. 下記の使用するデバイスの型番を Name Filter に入力します。

   5CGXFC5C6F27C7
10. Available devices から該当の型番を選択して、 **[Next]** ボタンをクリックします。

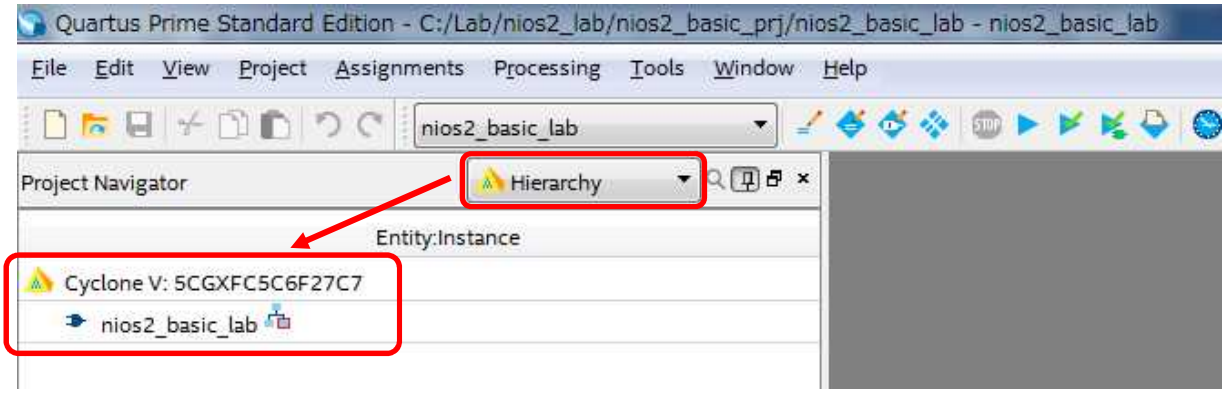


11. EDA Tools Settings が開くので、何もせず **[Next]** ボタンをクリックします。

12. Summary が開くので、以下のようになっていることを確認して **[Finish]** ボタンをクリックします。



13. Project Navigator ウインドウの Hierarchy タブに nios2_basic_lab が設定されていることを確認してください。

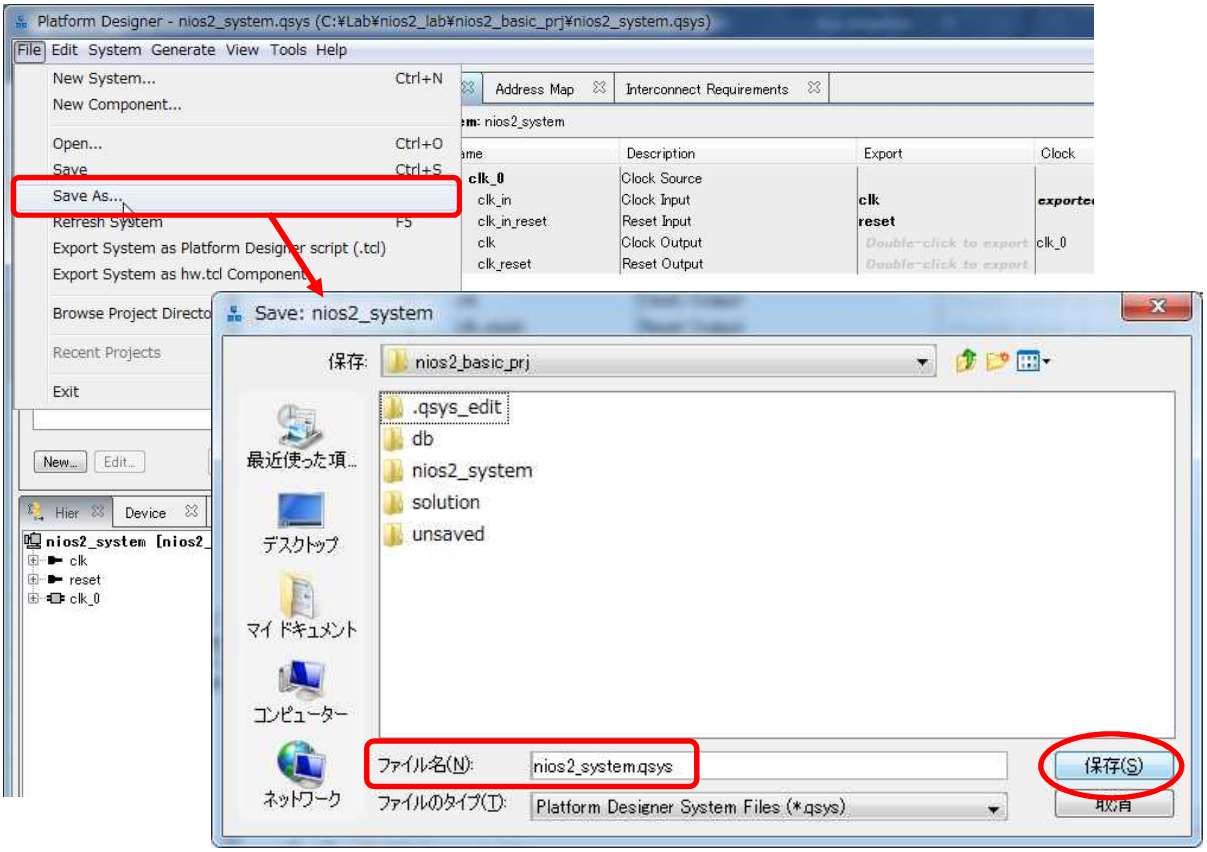


5. Platform Designer システム・モジュールの作成

Nios® II プロセッサ・コアを含む Platform Designer システム・モジュールを作成します。

5-1. Platform Designer の起動

1. Quartus® Prime の **Tools メニュー** ⇒ **Platform Designer** を選択します。
2. Platform Designer が起動したら、**File メニュー** ⇒ **Save as...** を選択して先にファイルにセーブします。
   今回は **nios2_system.qsys** というファイル名で保存します。



5-2. 使用するコンポーネント

Platform Designer システム・モジュールに、 Nios® II プロセッサ・コア、周辺 I/O、ペリフェラルを追加します。

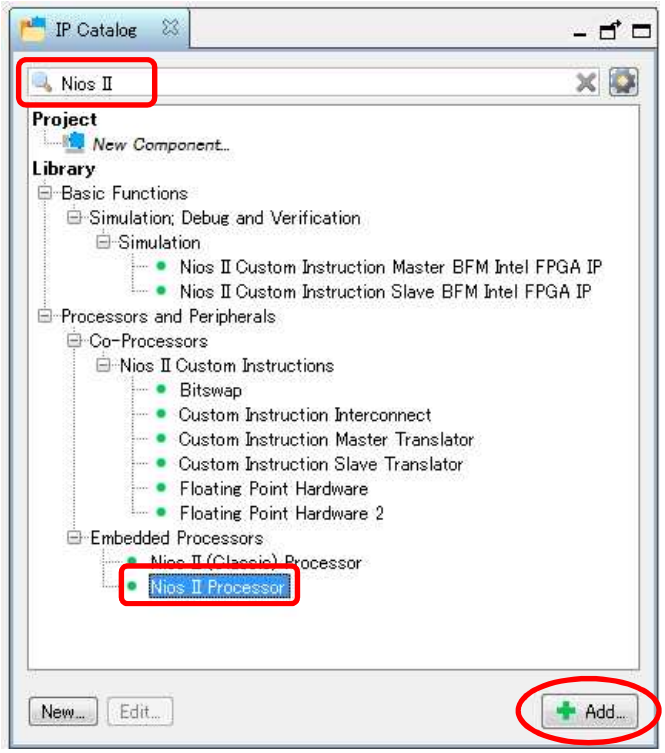
追加した Nios® II プロセッサ・コアやペリフェラル等を接続するシステムバス（Avalon バス）は、 Platform Designer により自動的に最適化され生成されます。今回の演習で追加するコンポーネントは以下になります。

- Nios® II プロセッサ・コア
- JTAG UART シリアル・インタフェース
- オンチップ・メモリ・インタフェース
- 外部 SRAM インタフェース
- トライステート・コンポーネント
- 外部 EPCQ インタフェース
- PIO ペリフェラル
- PLL

5-3. Nios® II プロセッサ・コアの追加

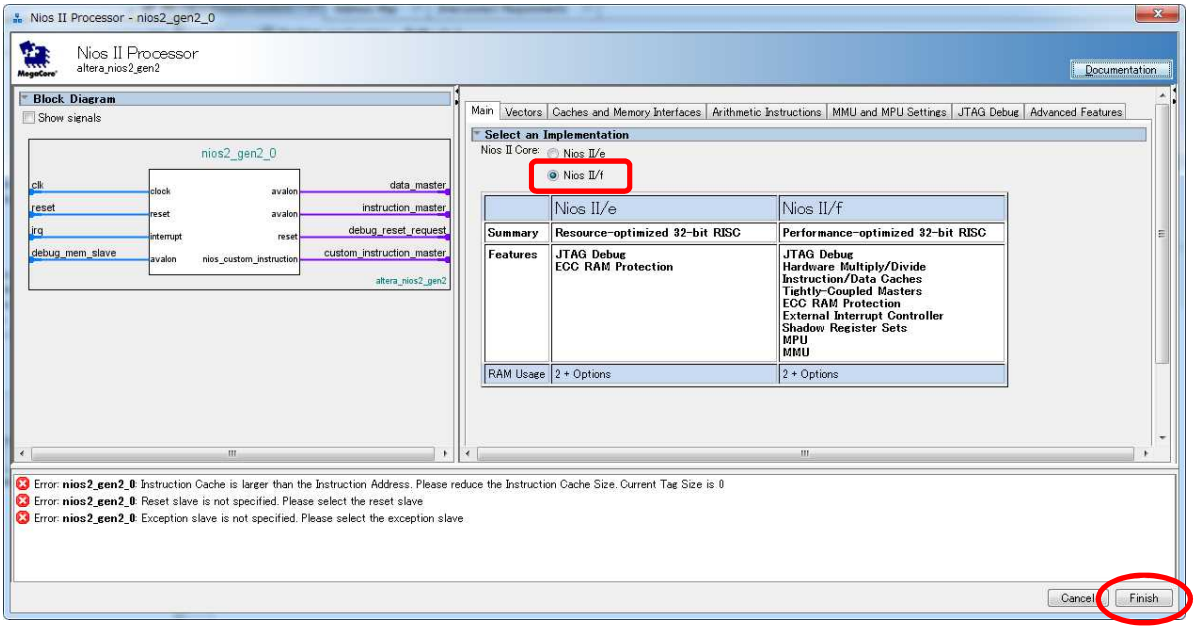
1. IP Catalog ウインドウの **Library** の項目から、 **Processors and Peripherals** ⇒ **Embedded Processors** 下の **Nios II Processor** を選択し、 **[Add]** ボタンをクリックします。

⚠ *IP Catalog ウインドウの検索入力（虫メガネのボックス部分）に該当するワード（ここでは、 Nios II）を入力すると簡単に該当のコンポーネントが見つかります。*



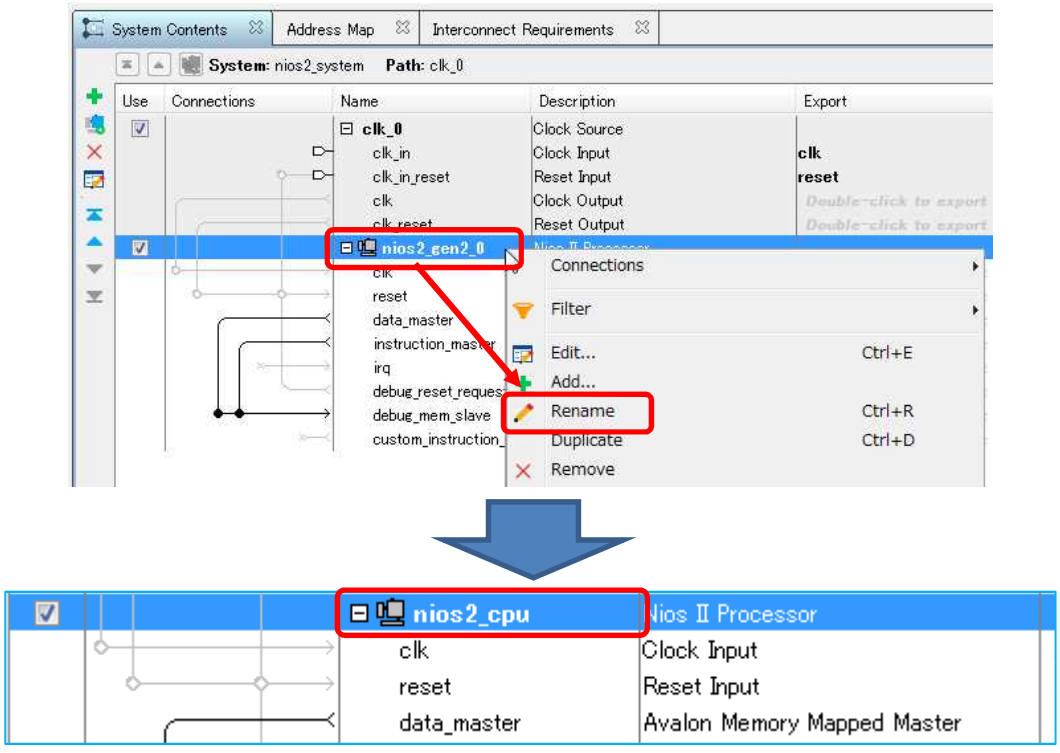
2. Nios® II 設定ウインドウが表示されますので、 **Nios II/f** が選択されていることを確認して **[Finish]** ボタンをクリックします。

⚠ *この時点で、 Message ウインドウにエラーが出ますが気にせず先に進んでください。*



3. **System Contents** タブの **Name** 項目の **nios2_gen2_0** をハイライト後、マウスで右クリックして **Rename** を選択します。

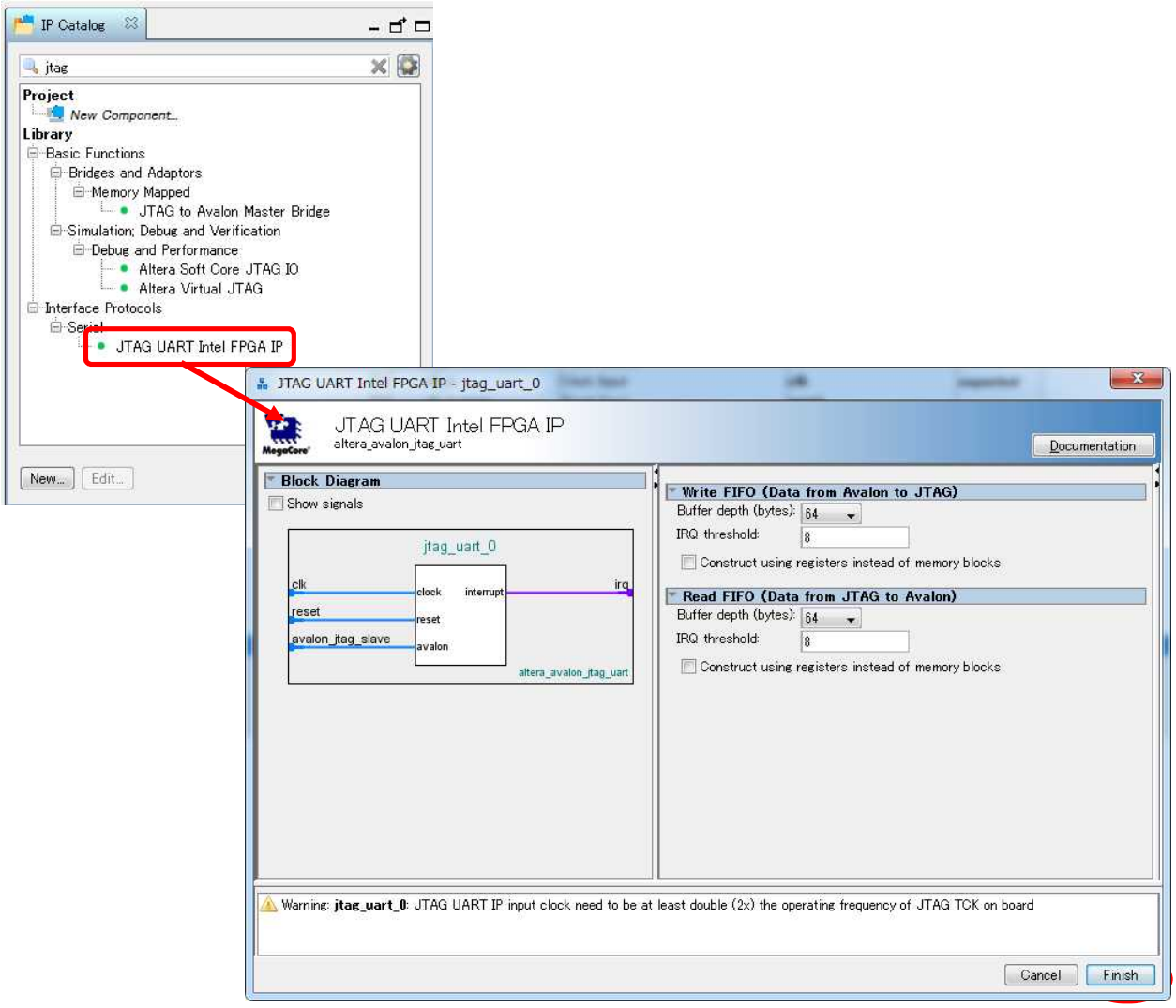
4. **nios2_gen2_0** から **nios2_cpu** に名前を変更します。



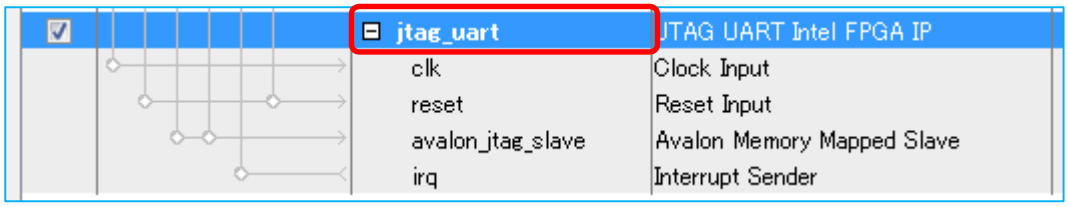
5-4. JTAG UART シリアル・インタフェースの追加

Nios® II とシリアル接続を行うためのインタフェースとして、 JTAG UART を追加します。

1. **IP Catalog** ウインドウの **Library** の項目から、 **Interface Protocols** ⇒ **Serial** より **JTAG UART Intel FPGA IP** を選択し、 **[Add]** ボタンをクリックします。
2. JTAG UART ウインドウが表示されるので、デフォルトのまま **[Finish]** ボタンをクリックし、 Platform Designer のメイン・ウインドウに戻ります。

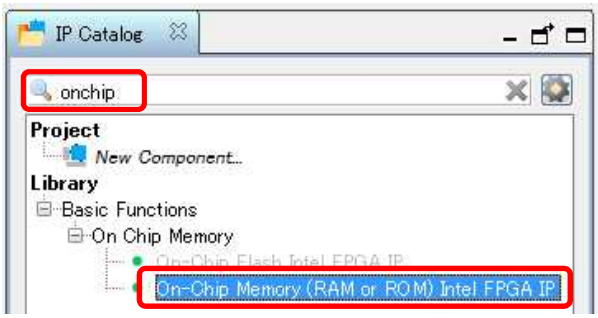


3. **Name** 項目の **jtag_uart_0** をハイライト後、マウスで右クリックして **Rename** を選択します。
4. **jtag_uart_0** から **jtag_uart** に名前を変更します。



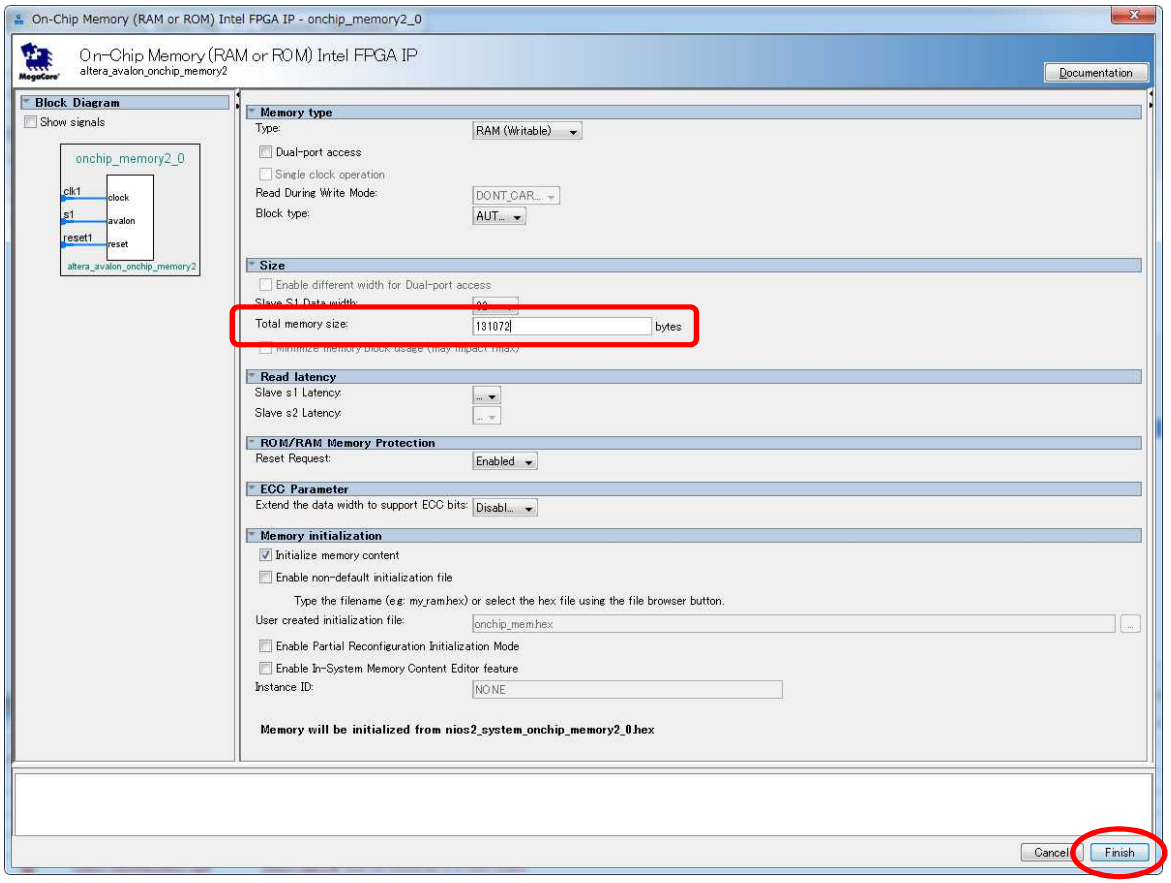
5-5. オンチップ・メモリ・インタフェースの追加

1. **IP Catalog** ウインドウの **Library** の項目から、**Basic Functions** の **On Chip Memory** より **On-Chip Memory (RAM or ROM)** を選択し、**[Add]** ボタンをクリックします。



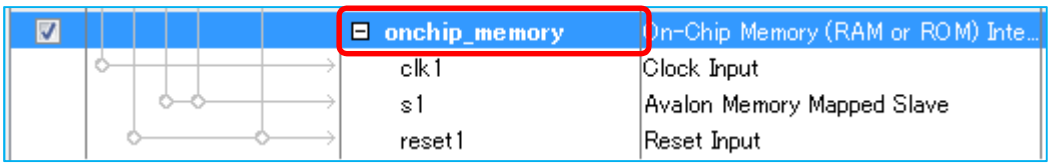
2. On -Chip Memory (RAM or ROM) ウインドウが表示されるので、**Total Memory Size** に **131072** と入力し、**[Finish]** ボタンをクリックします。

⚠ *128K と入力し、Tab キーを押すと自動的に 131072 に変換されます。*



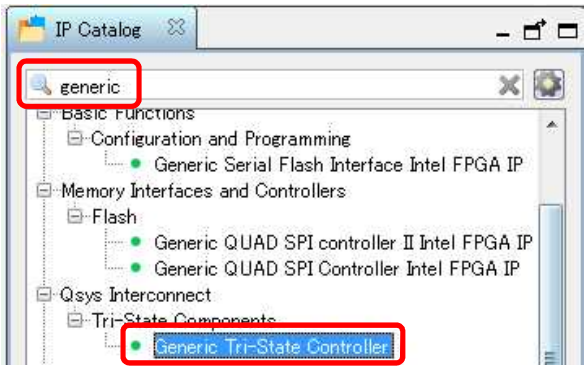
3. **Name** 項目の **onchip_memory2_0** をハイライト後、マウスで右クリックして **Rename** を選択します。

4. **onchip_ memory2_0** から **onchip_memory** に名前を変更します。



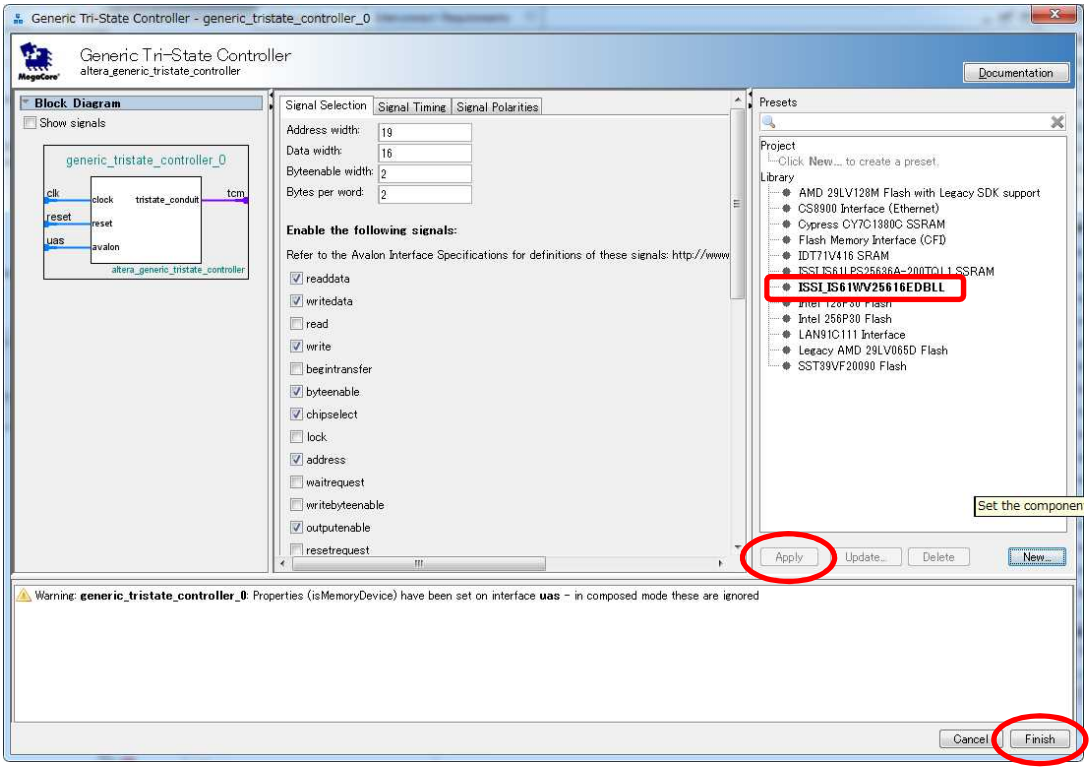
5-6. 外部 SRAM インタフェースの追加

1. **IP Catalog** ウインドウの **Library** の項目から、**Qsys Interconnect** の **Tri-State Components** の **Generic Tri-State Controller** を選択し、 **[Add]** ボタンをクリックします。



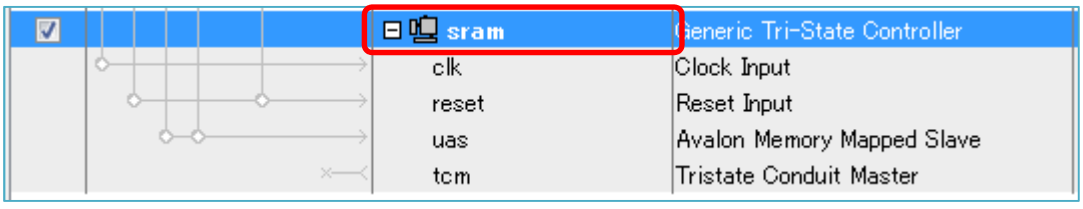
2. Presets 一覧より **ISSI_IS61WV25616EDBLL** を選択し、 **[Apply]** ボタンをクリックします。次に **[Finish]** ボタンをクリックし画面を閉じます。

⚠ *下図では、ISSI_IS61WV25616EDBLL は、Library フォルダ内に一覧されていますが、Project フォルダ内に表示されている場合もあります。*



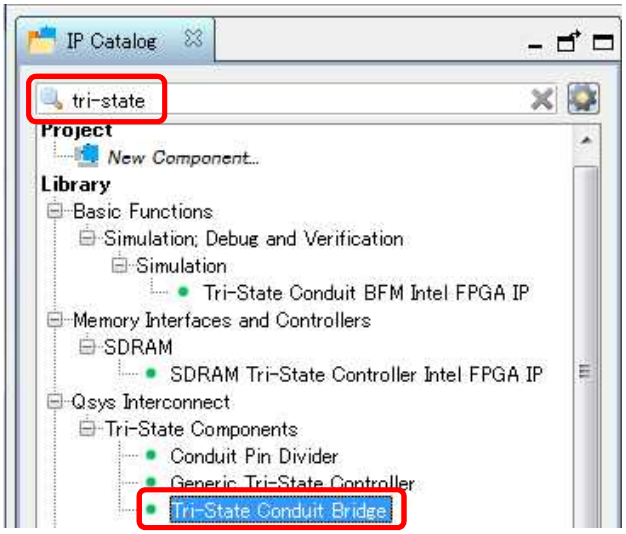
3. **Name** 項目の **generic_tristate_controller_0** をハイライト後、マウスで右クリックして **Rename** を選択します。

4. **generic_tristate_controller_0** から **sram** に名前を変更します。

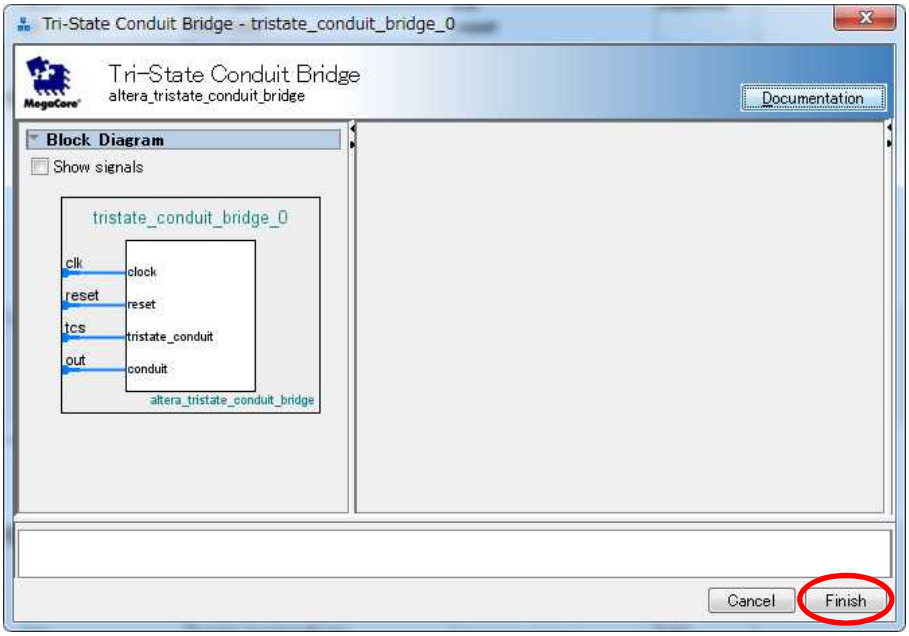


5-7. トライステート・コンポーネント

1. **IP Catalog** ウインドウの **Library** の項目から、**Qsys Interconnect** の **Tri-State Components** の **Tri-State Conduit Bridge** を選択し、**[Add]** ボタンをクリックします。

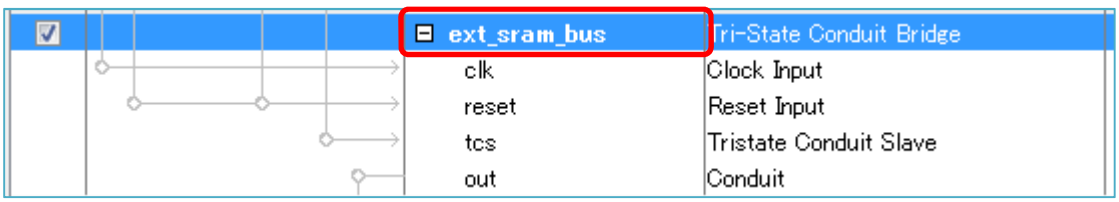


2. 設定はデフォルトのまま、**[Finish]** ボタンをクリックして画面を閉じます。



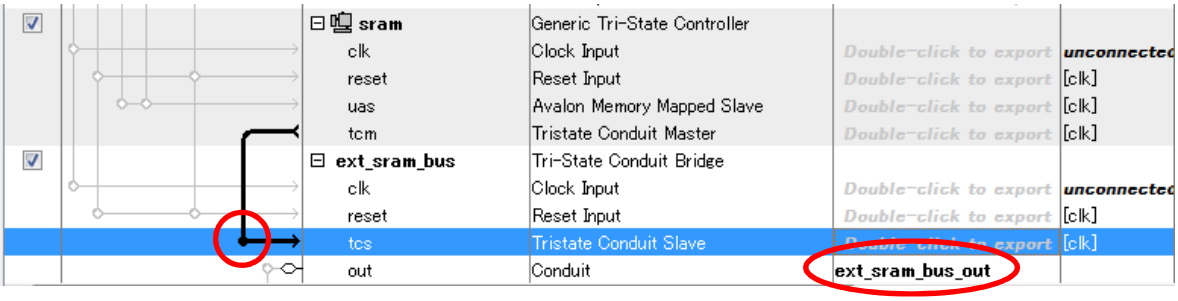
3. **Name** 項目の **tristate_conduit_bridge_0** をハイライト後、マウスで右クリックして **Rename** を選択します。

4. **tristate_conduit_bridge_0** から **ext_sram_bus** に名前を変更します。



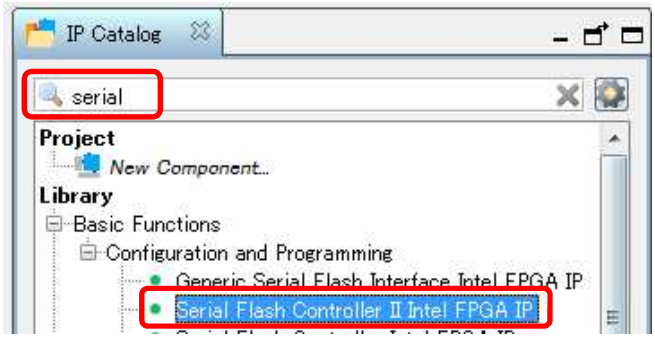
5. **Connections** 項目の **sram** の **tcm** ポートと、**ext_sram_bus** の **tcs** ポートの交点の白丸をクリックして黒丸に変更します。この操作により、**sram** と **ext_sram_bus** の 2 つのペリフェラルが接続されます。

6. **ext_sram_bus** の **out** の **Export** 項目をダブル・クリックすると自動的に **ext_sram_bus_out** という文字列が挿入されます。この操作で、**sram** のバス出力が Platform Designer システムの外部へエクスポートされるようになります。

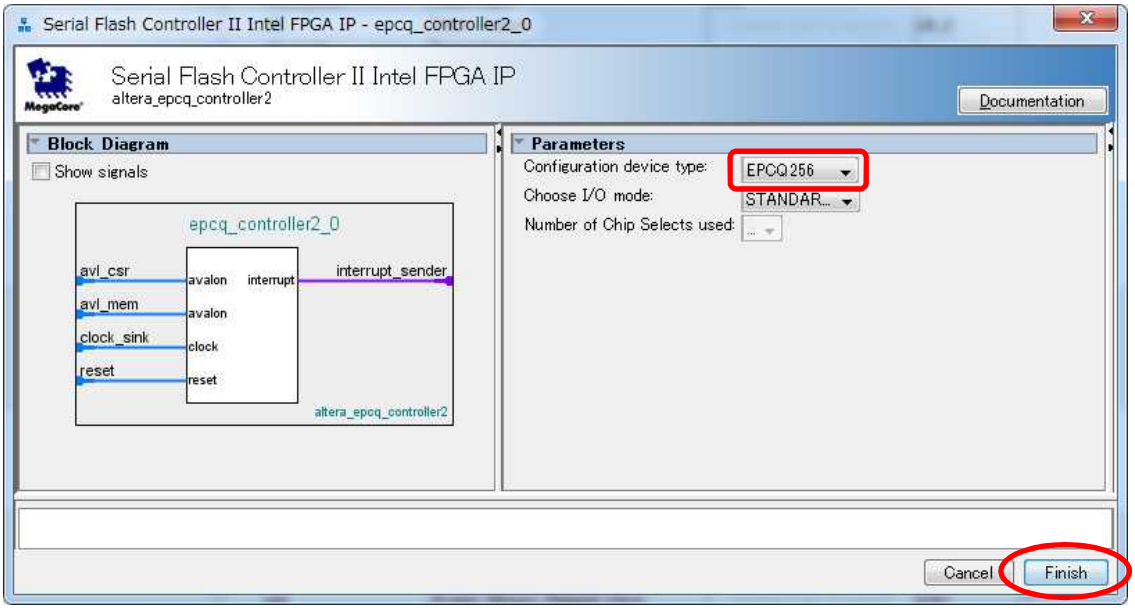


5-8. 外部 EPCQ インタフェースの追加

1. **IP Catalog** ウインドウの **Library** の項目から、**Basic Function** の **Configuration and Programming** の **Serial Flash Controller II Intel FPGA IP** を選択し、**[Add]** ボタンをクリックします。



2. Serial Flash Controller II Intel FPGA IP ウインドウが起動するので、**Configuration device type** に **EPCQ256** を選択し、**[Finish]** ボタンをクリックします。



3. **Name** 項目 **epcq_controller_2_0** をハイライト後、マウスで右クリックして **Rename** を選択します。

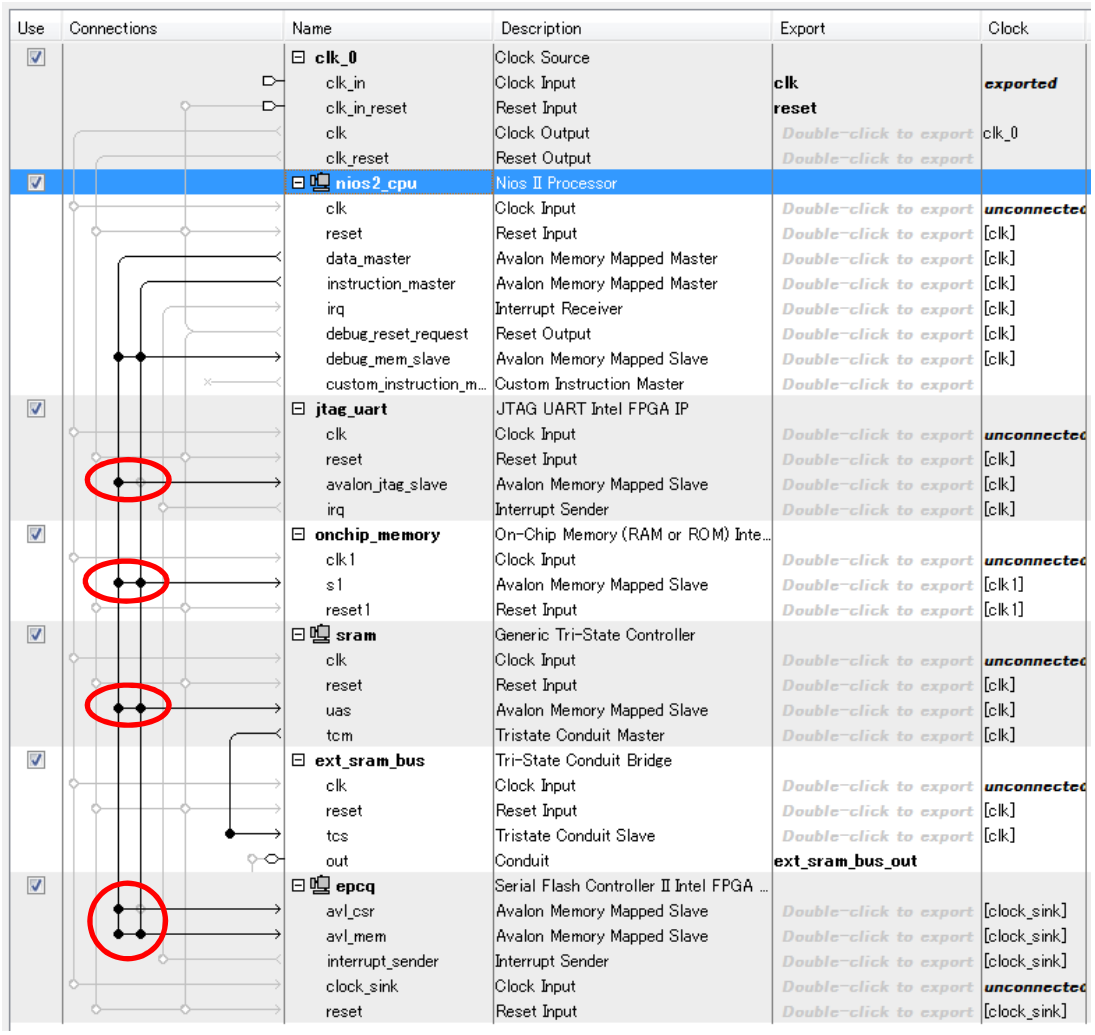
4. **epcq_controller_2_0** から **epcq** に名前を変更します。

5-9. リセット・ベクタ と 例外ベクタの設定

1. Nios® II と外部メモリを接続します。

   Connection 欄で **nios2_cpu** の **data_master** と **instruction_master** を **onchip_memory** の **s1** ポートと **sram** の **uas** ポートと、**epcq** の **avl_csr** ポートと **avl_mem** ポートに図のように接続します。

   さらに **jtag_uart** の **avalon_jtag_slave** も **nios2_cpu** の **data_master** と図のように接続します。

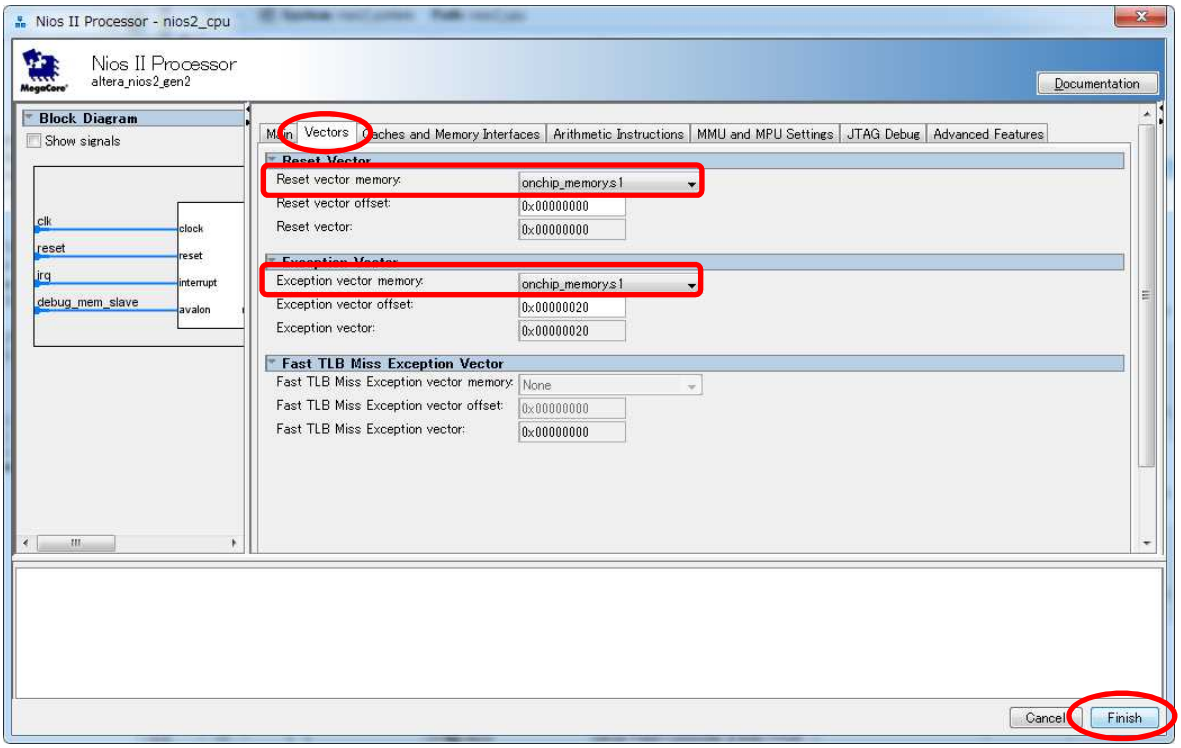


2. **nios2_cpu** をハイライト後、マウスで右クリックし **Edit** を選択します。再び、 Nios® II プロセッサ・コアの GUI 画面が起動します。

   Vectors タブの **Reset Vector** の項目のプルダウン・メニューより、 **onchip_memory.s1** を選択します。

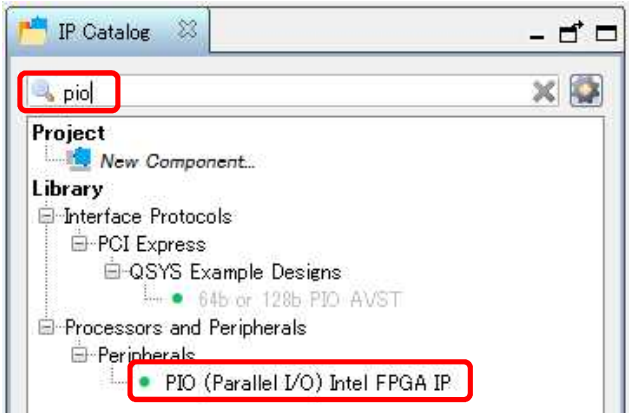
   また、同様にプルダウン・メニューより、 **Exception Vector** の項目も **onchip_memory.s1** を選択します。設定が終わったら、 **[Finish]** ボタンをクリックします。

   ⚠ *本演習では、まず Nios® II をオンチップ・メモリからブートする手法を説明します。後の章（オプション）では、 Reset Vector の設定を EPCQ に指定して、 EPCQ からのブートの方法について説明します。*

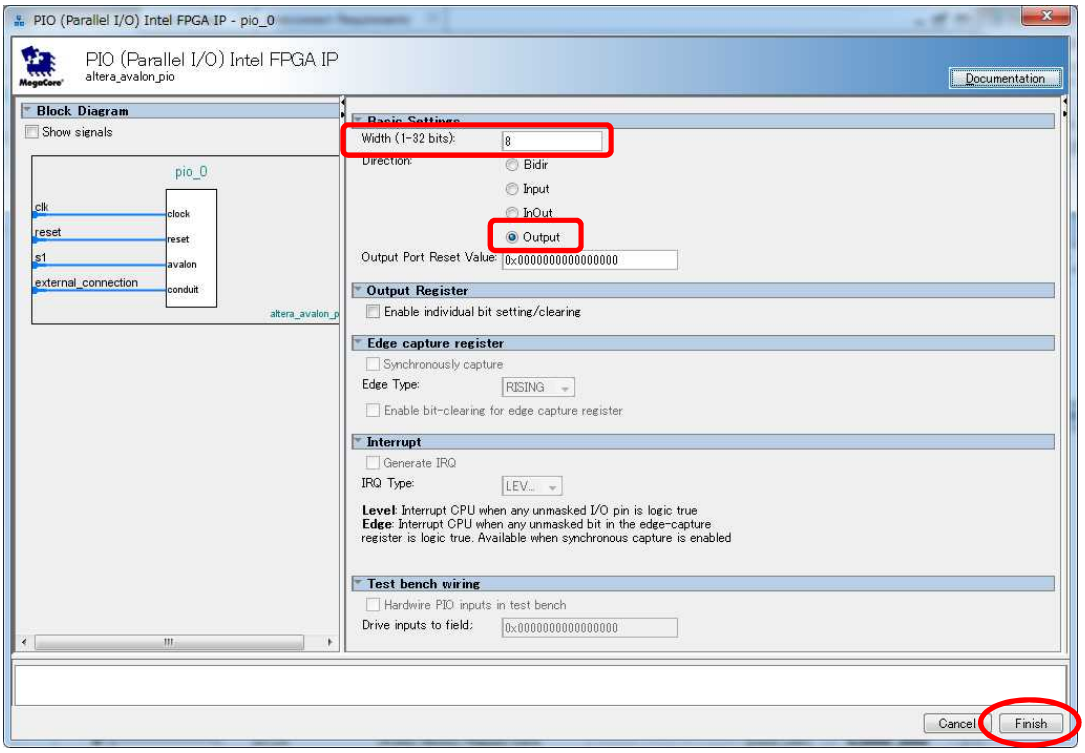


5-10. PIO ペリフェラルの追加

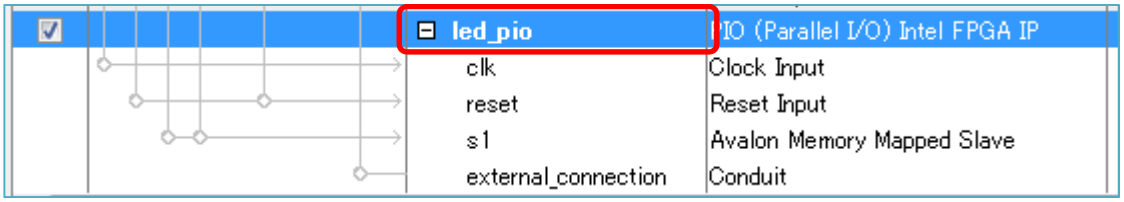
1. **IP Catalog** ウインドウの **Library** の項目から、**Processors and Peripherals ⇒ Peripherals** の項目より **PIO (Parallel I/O) Intel FPGA IP** を選択し、**[Add]** ボタンをクリックします。



2. **Basic Settings** タブ内の設定内容が以下になっていることを確認し、**[Finish]** ボタンをクリックします。
   - Width : **8 bits**
   - Direction : **Output**

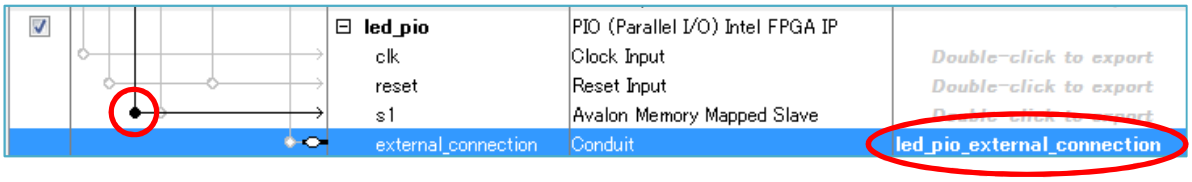


3. **Name** 項目の **pio_0** をハイライト後、マウスで右クリックして **Rename** を選択します。
4. **pio_0** から **led_pio** に名前を変更します。



5. Connections 項目の **led_pio** の **s1** を **nios2_cpu** の **data_master** に接続します。

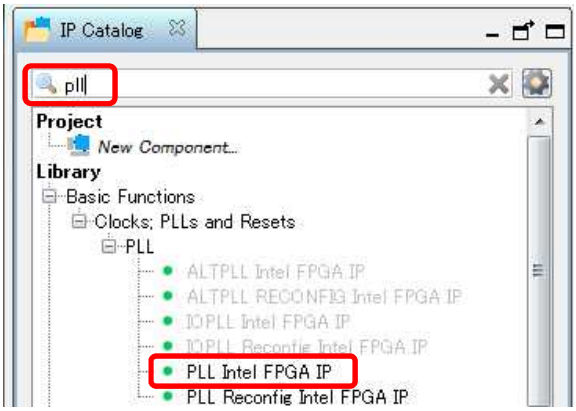
6. **external_connection** の **Export** 項目をダブル・クリックすることで、LED 出力を Export します。信号名 **led_pio_external_connection** が自動的に入力されます。この設定を行うと、Connections 項目にこの信号が Export であることが表示されます。



5-11. PLL（フェース・ロックド・ループ）の追加

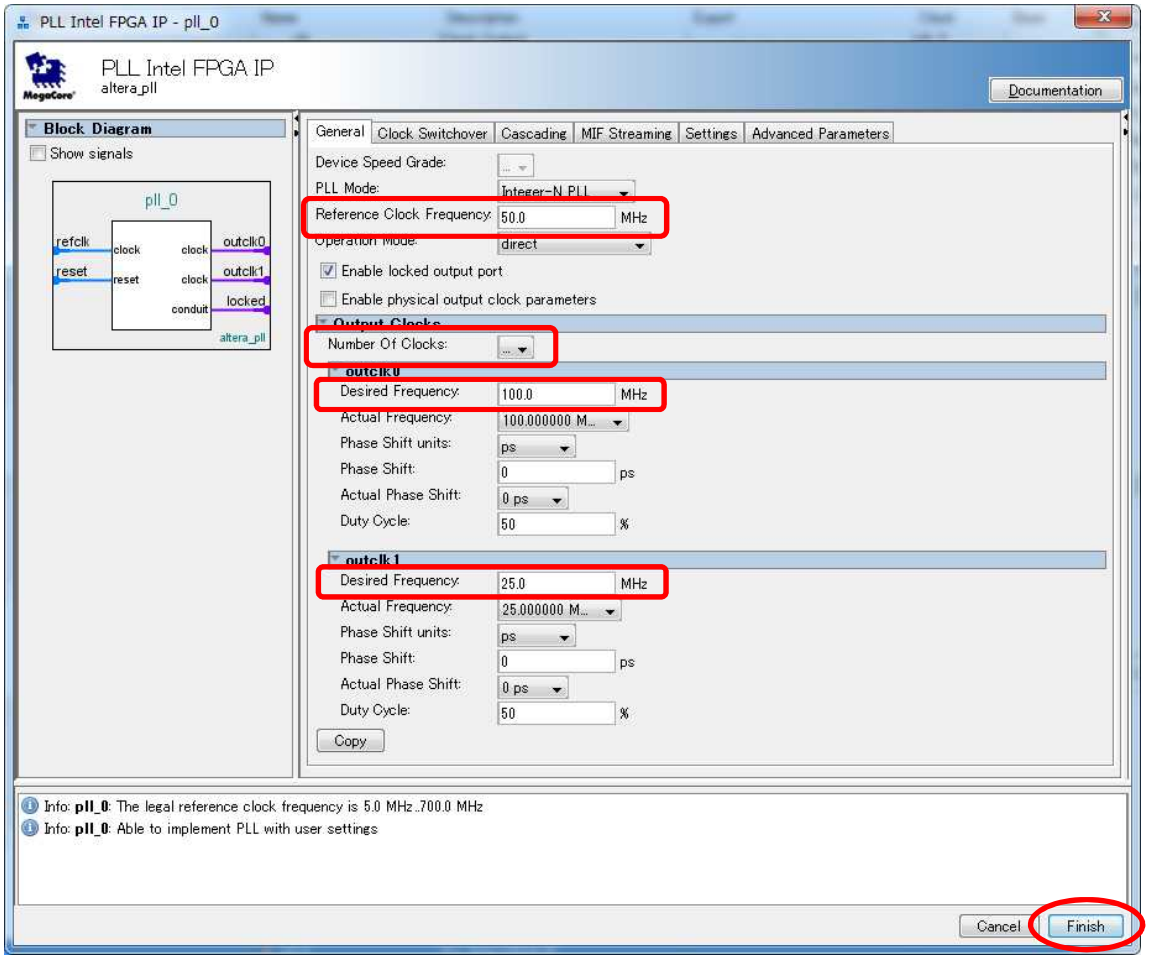
50MHz のクロックを逓倍して 100MHz のクロック 1 系統と 25MHz のクロック 1 系統を生成します。生成したクロックは、Platform Designer のシステム・クロック（プロセッサの動作クロックも含む）として使用します。

1. **IP Catalog** ウインドウの **Library** の項目から、**Basic Function** ⇒ **Clocks; PLLs and Resets** ⇒ **PLL** の項目より **PLL Intel FPGA IP** を選択し、**[Add]** ボタンをクリックします。



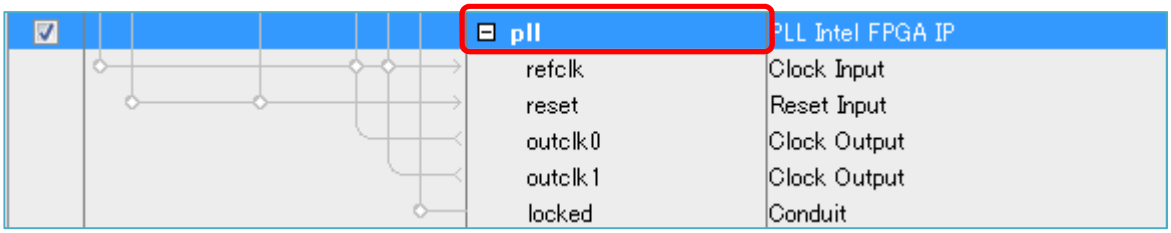
2. PLL の設定を行うための GUI 画面が起動しますので、下記のように設定し、**[Finish]** ボタンをクリックします。

   - Reference Clock Frequency ： **50MHz**
   - Number Of Clocks ： **2**
   - outclk_0.Desired Frequency ： **100MHz**
   - outclk_1.Desired Frequency ： **25MHz**



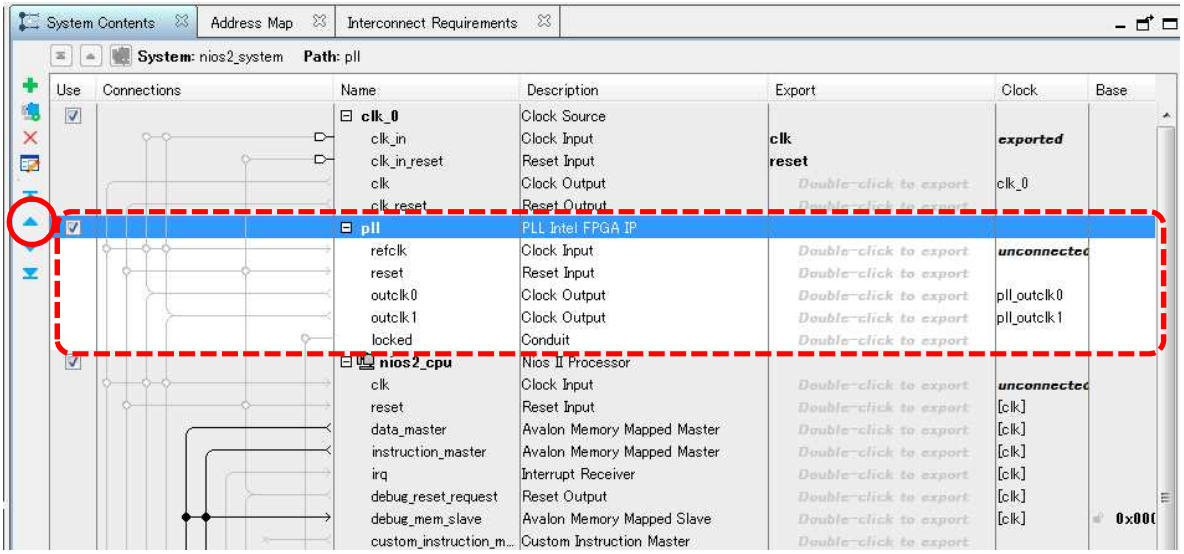
3. Name 項目の **pll_0** をハイライト後、マウスで右クリックして **Rename** を選択します。

4. **pll_0** から **pll** に名前を変更します。

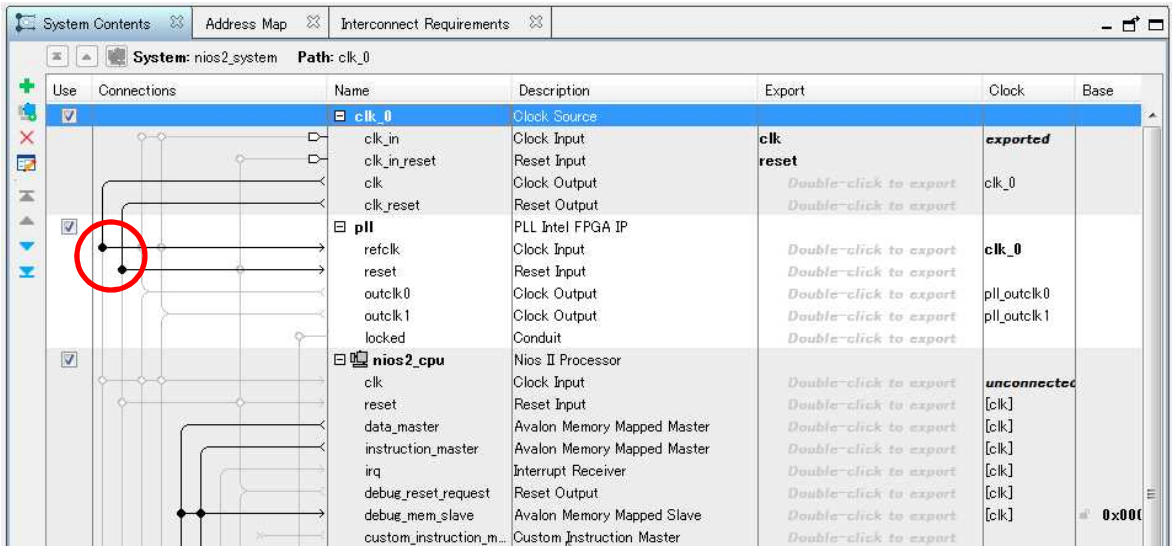


5-12. クロックとリセットの接続

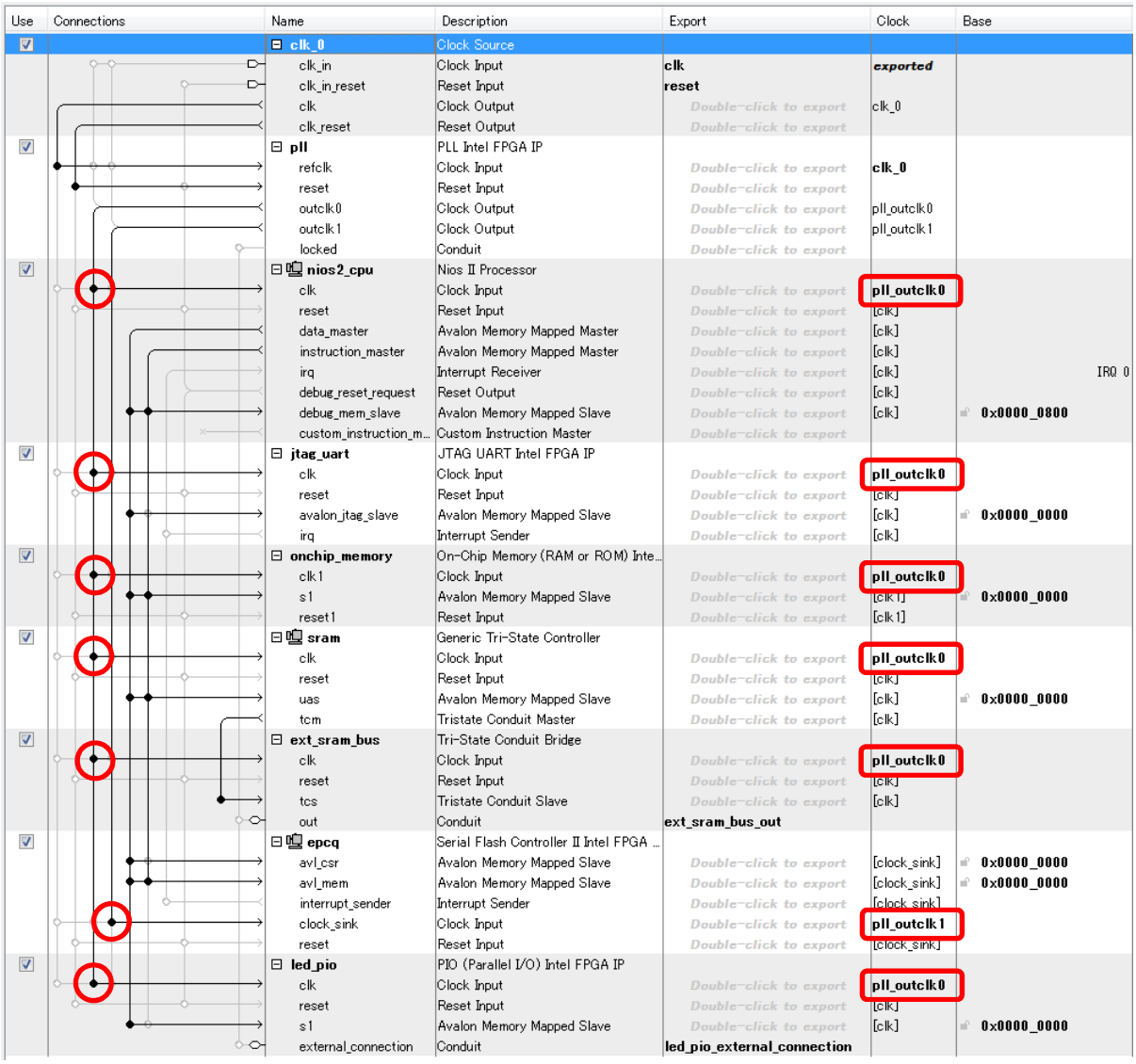
1. Platform Designer の System Contents タブに戻り、クロックとリセットの接続を行います。画面を見やすくするために **pll** を選択し、**[Move Up]** ボタン（▲）を **7** 回クリックし、**clk_0** の下に **pll** を移動します。



2. 次に **pll** の入力として、外部 **50MHz** のクロック（**clk_0.clk**）を接続します。併せて、リセット（**clk_0.clk_reset**）も接続します。図のように接続してください。

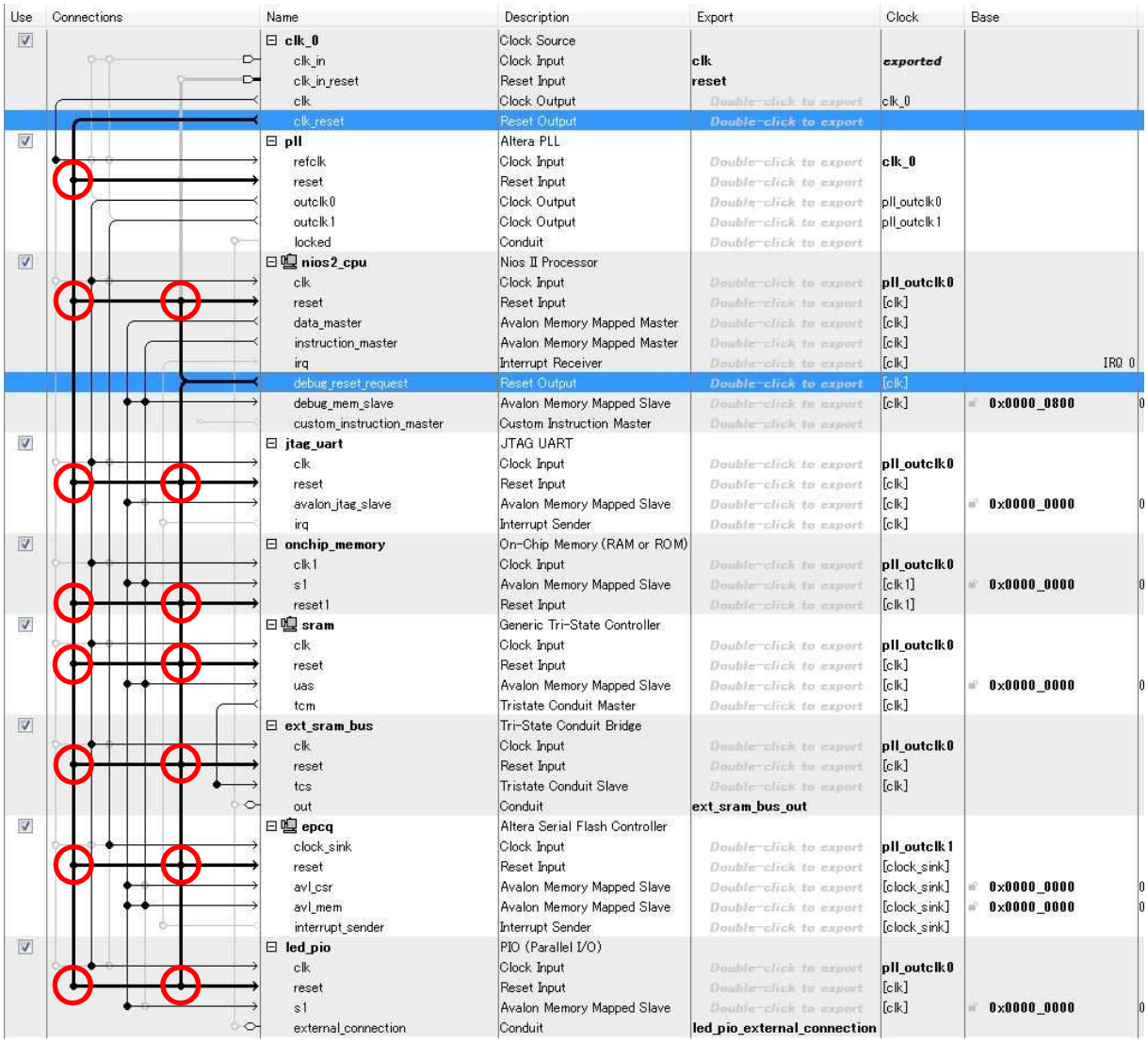


3. Clock の項目（点線部分）で、**clk_0** と **pll** を除いた全てのペリフェラルに対して、プルダウン・メニューより **Unconnected** から **pll_outclk0/1** に変更します。 **Epcq** のみ **pll_outclk1** を接続します。その他は、**pll_outclk0** を接続します。これは、 **Connections** の項目から接続で行うことも可能です。

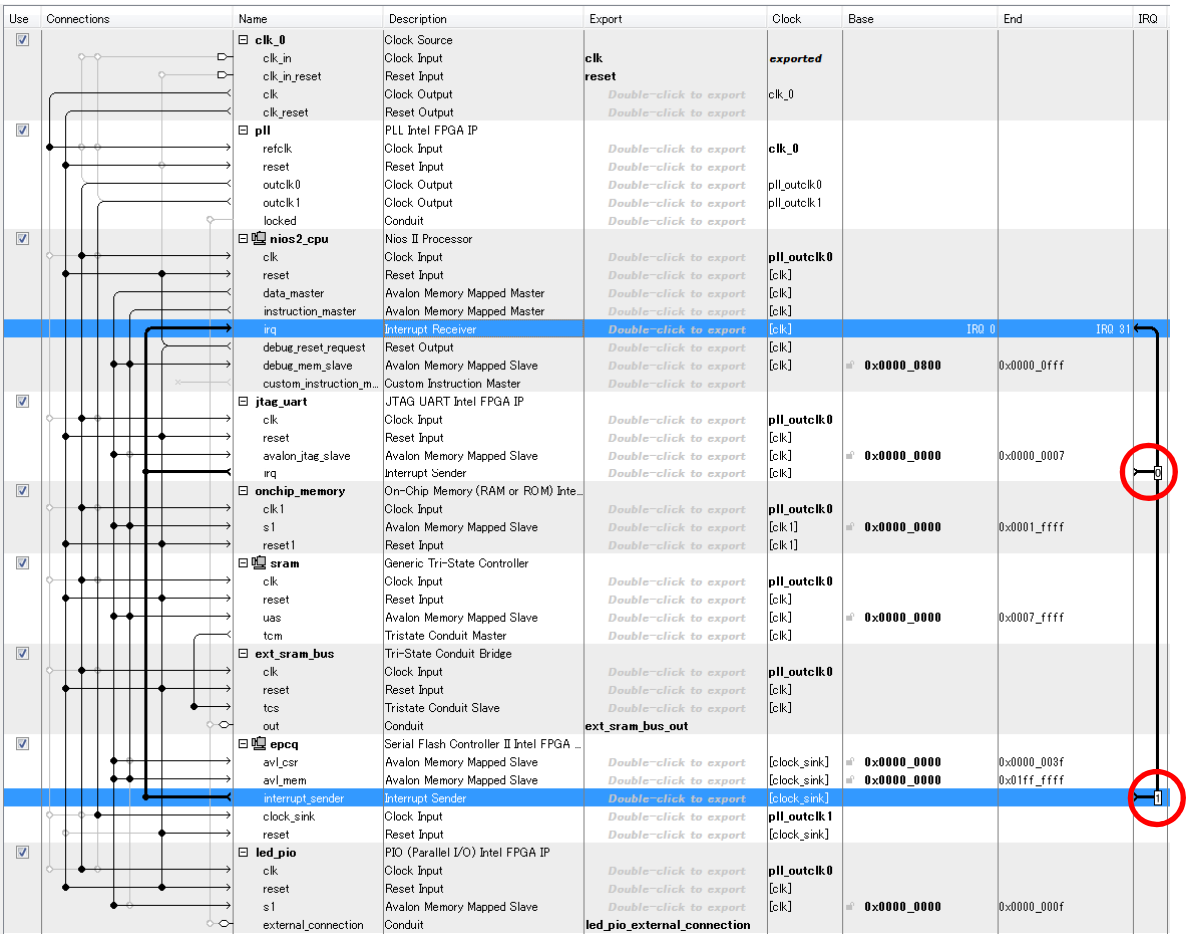


4. 次に **Connections** 項目にてリセット信号を接続します。 **clk_0** の **clk_reset** を各ペリフェラルのリセット信号に接続します。

5. 同様に **nios2_cpu** の **debug_reset_request** も **clk_0**、 **pll** 以外の各ペリフェラルのリセット信号に接続します。これは、デバッグ用途で使用されるリセット信号となります。



6. 次に **jtag_uart**、**epcq** から Nios® II コアへの割り込み接続を設定します。**IRQ** 列にある **jtag_uart** の白丸部分をクリックします。次に **epcq** の **IRQ** 列にある白丸をクリックします。**jtag_uart** の割り込み番号が **0** に、**epcq** の割り込み番号が **1** に設定されました。0 が優先順位の高い設定となります。

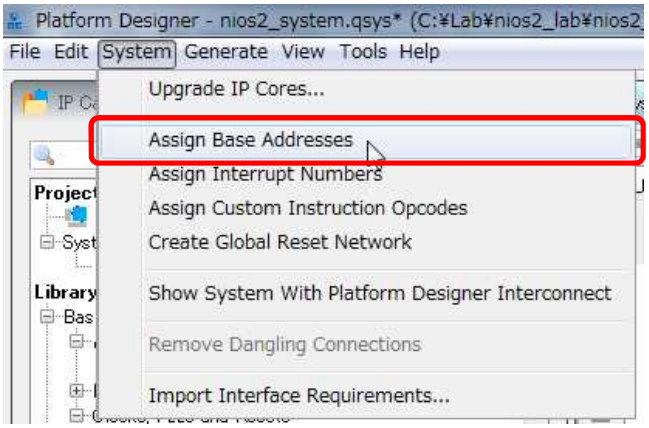


5-13. ベース・アドレスの設定

1. ペリフェラルの追加が行われると、 Platform Designer は自動的に各ペリフェラルに対して、ベース・アドレスを設定します。 **System** メニュー ⇒ **Assign Base Address** を選択すると、ベース・アドレスが自動的に変更されます。

   ⚠ *Base 項目のベース・アドレスをダブル・クリックすることで、直接ベース・アドレスを入力することもできます。*

   ⚠ *Address Map タブをクリックすると、設定されたベース・アドレスの一覧を確認できます。*



System Contents | Address Map | Interconnect Requirements

System: nios2_system **Path:** led_pio

	nios2_cpu.data_master	nios2_cpu.instruction_master
epcq_avl_csr	0x0414_1000 - 0x0414_103f	
epcq_avl_mem	0x0200_0000 - 0x03ff_ffff	0x0200_0000 - 0x03ff_ffff
jtag_uart.avalon_jtag_slave	0x0414_1050 - 0x0414_1057	
led_pio.s1	0x0414_1040 - 0x0414_104f	
nios2_cpu.debug_mem_slave	0x0414_0800 - 0x0414_0fff	0x0414_0800 - 0x0414_0fff
onchip_memory.s1	0x0412_0000 - 0x0413_ffff	0x0412_0000 - 0x0413_ffff
sram.uas	0x0408_0000 - 0x040f_ffff	0x0408_0000 - 0x040f_ffff

⚠ *表示されるアドレス値は、上図と一致するとは限りません。異なっていても問題はないので、そのまま進めてください。*

2. 最後に Messages タブの内容を確認します。下図のようにいくつかのワーニングやメッセージが表示されていますが、問題ないのでそのまま進めます。

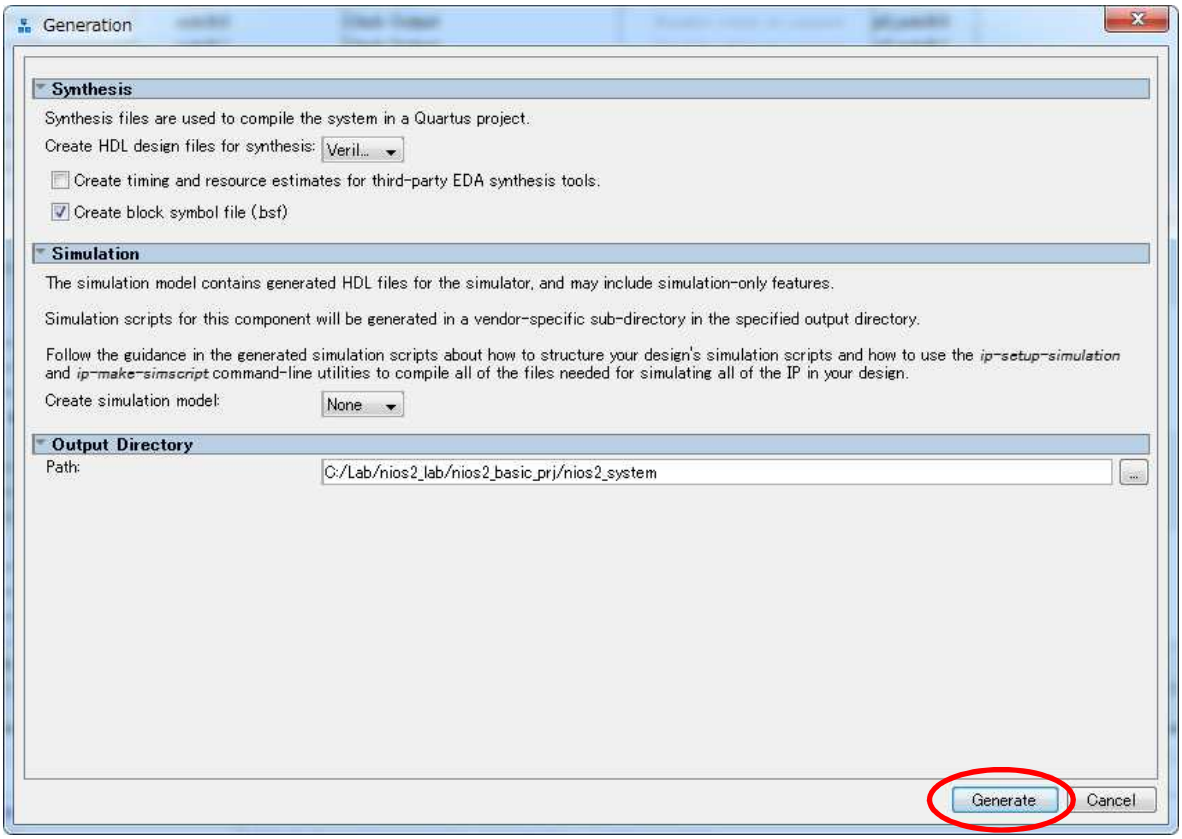
Messages

Type	Path	Message
⚠	2 Warnings	
⚠	**nios2_system.sram**	Properties (isMemoryDevice) have been set on interface **uas** - in composed mode these are ignored
⚠	**nios2_system.pll**	**pll.locked** must be exported, or connected to a matching conduit.
ⓘ	3 Info Messages	
ⓘ	**nios2_system.jtag_uart**	JTAG UART IP input clock need to be at least double (2x) the operating frequency of JTAG TCK on board
ⓘ	**nios2_system.pll**	The legal reference clock frequency is 5.0 MHz..700.0 MHz
ⓘ	**nios2_system.pll**	Able to implement PLL with user settings

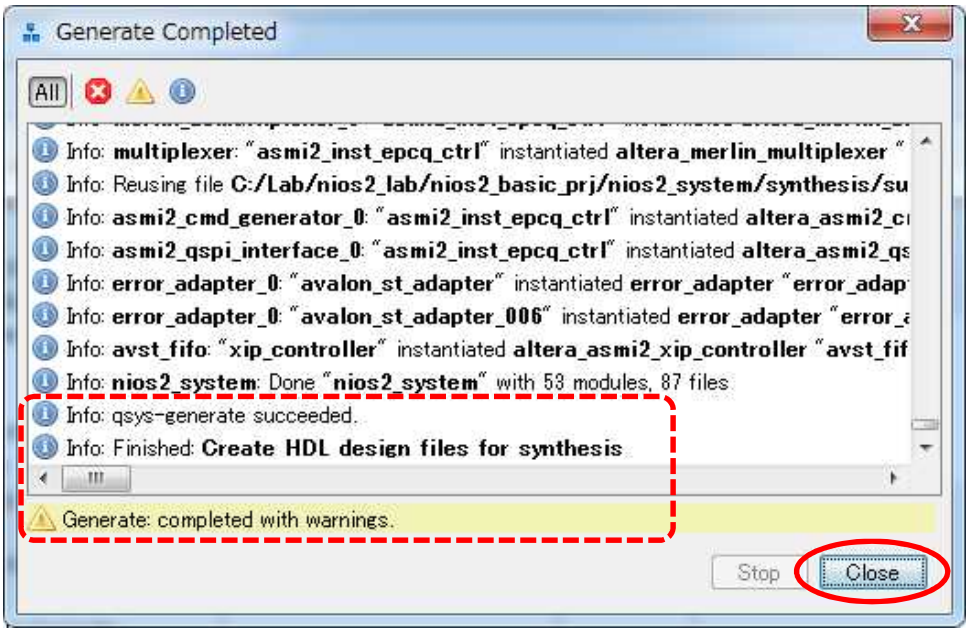
3. Platform Designer 設定ファイルの保存をするため、 **File** メニュー ⇒ **Save** を選択し、 **nios2_system.qsys** ファイルを保存します。

5-14. Platform Designer システム・モジュールの生成

1. Platform Designer のメイン・ウインドウから **Generate** メニュー ⇒ **Generate HDL** を選択します。
2. Generate ウインドウが表示されるので、**[Generate]** ボタンをクリックしてシステムを生成します。

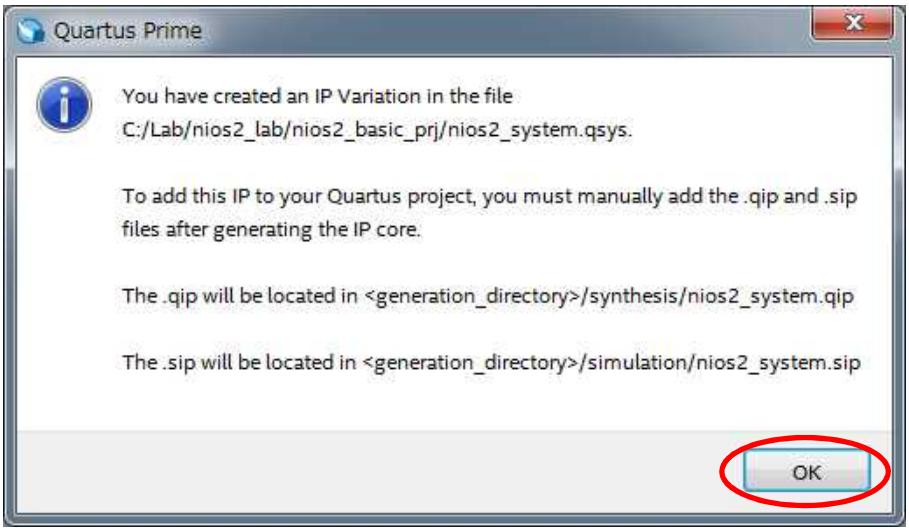


3. Platform Designer システム・モジュールの生成が正常に終了した場合、メッセージ・ボックス内に図のようなメッセージが表示されます。**[Close]** ボタンをクリックして、メッセージ・ボックス を閉じてください。



4. Platform Designer のメイン・ウインドウからから **File** メニュー ⇒ **Exit** で Platform Designer を閉じてください。

5. 以下のようなウインドウが表示されたら **[OK]** ボタンで閉じてください。

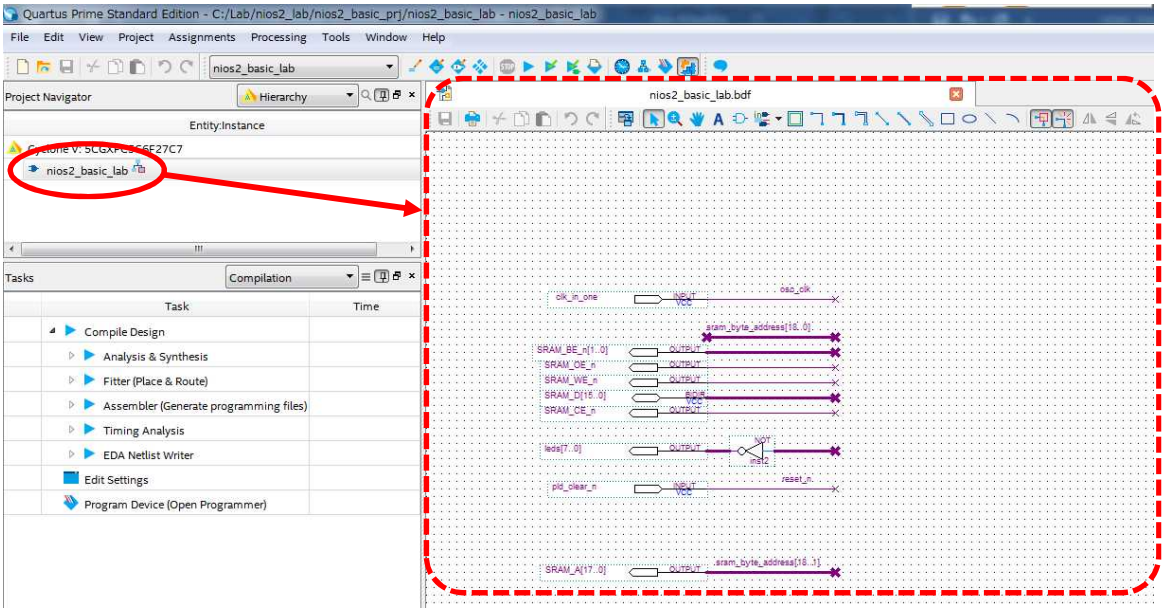


6. ハードウェア・デザインの作成

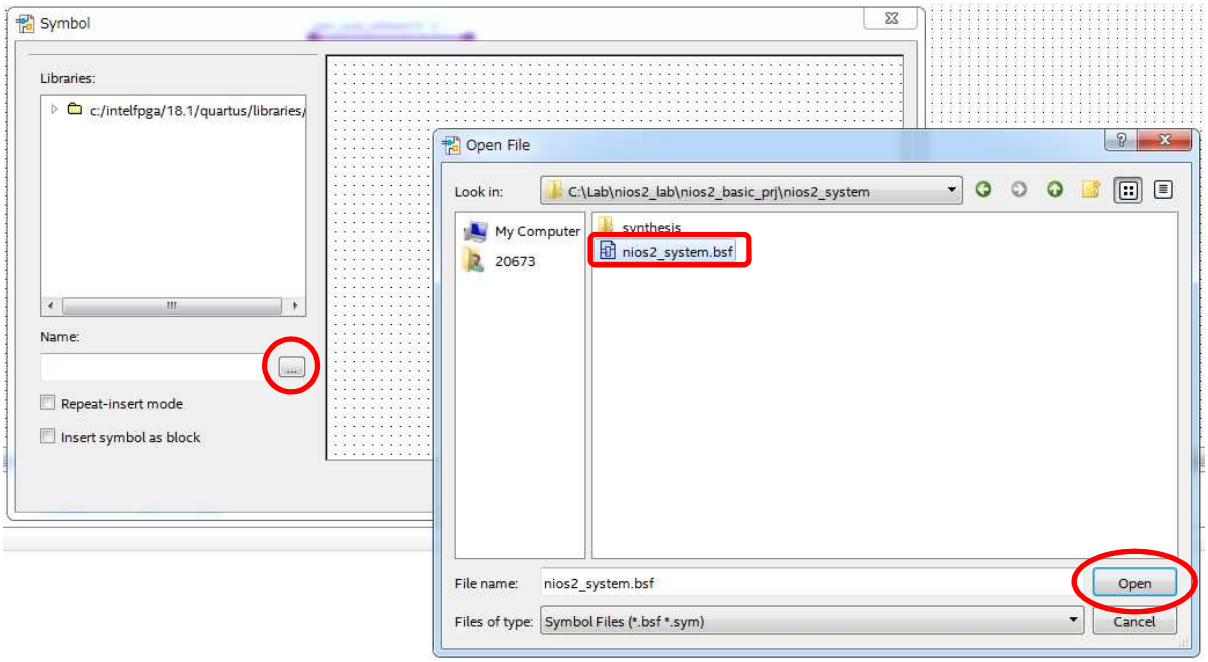
Platform Designer システム・モジュールをトップ階層に配置して、ハードウェア・デザインを作成します。

6-1. Platform Designer システム・モジュールの生成

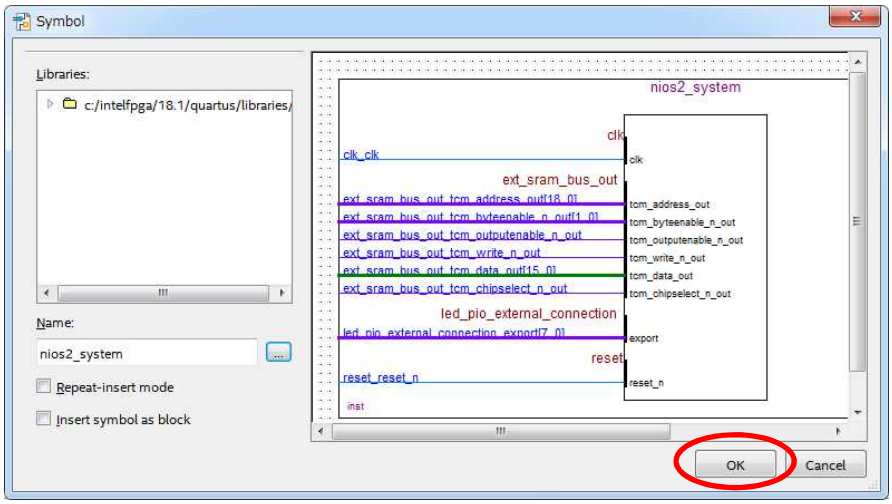
1. Quartus® Prime で **File** メニュー ⇒ **Open** を選択します。
2. トップ階層のファイルは、 **nios2_basic_lab.bdf** というファイル名で予め用意されています。
   Project Navigator ウインドウの **nios2_basic_lab** をダブル・クリックするとファイルが開きます。



3. 回路図の空白部分にマウス・ポインタを置いてダブル・クリックすると、Symbol ウインドウが起動します。
4. ブラウズ・ボタンをクリックし、**C:¥Lab¥nios2_lab¥nios2_basic_prj¥nios2_system** フォルダ内の **nios2_system.bsf** ファイルを選択し **[Open]** をクリックします。

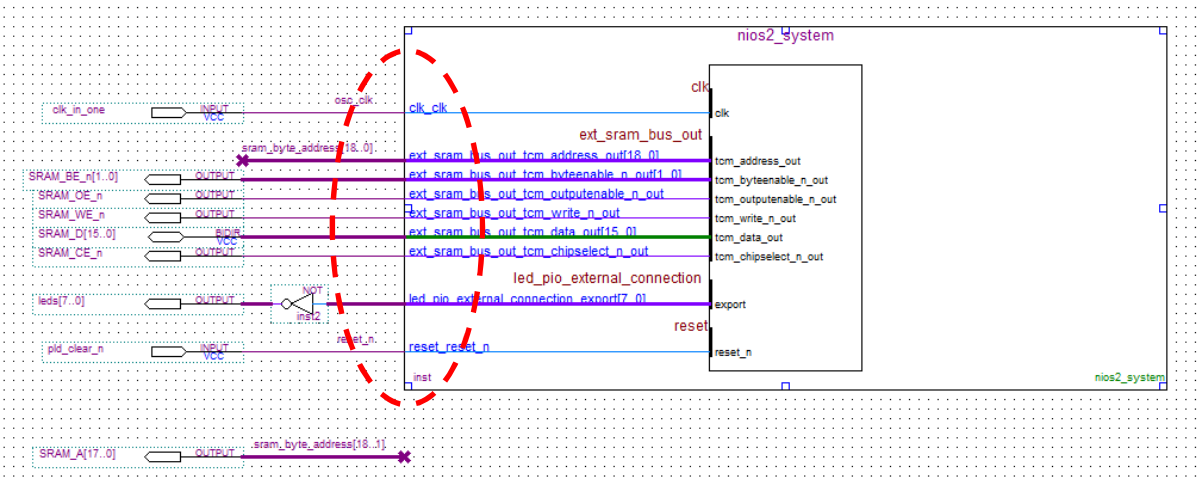


5. 次のように、 Library ウインドウに **nios2_system** のブロック図が表示されるので、 **[OK]** ボタンをクリックします。



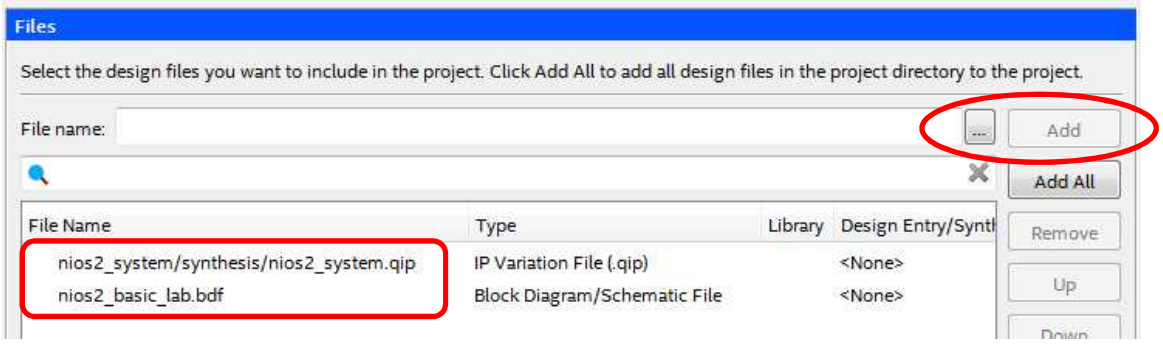
6. マウス・ポインタが Platform Designer システム・モジュールの枠型に変わります。

7. **nios2_basic_lab.bdf** ファイルの回路図上の空白に、シンボルを配置します。問題が無ければ、下図のように、ぴったりと配置できるはずです。できない場合には、何かが間違っている可能性があるので、Platform Designer システムを見直す必要があるかもしれません。



8. **File** メニュー ⇒ **Save** を選択して、回路図を保存します。

9. **Assignments** メニュー ⇒ **Settings** ⇒ **Files** にて **nios2_system.qip**、**nios2_basic_lab.bdf** を設定します。ブラウズ・ボタンで **nios2_system/synthesis** フォルダから **nios2_system.qip** を選択して、 **[Add]** ボタンで追加します。次に、同様の操作で **nios2_basic_lab.bdf** を設定します。設定できたら **[OK]** ボタンをクリックして画面を閉じます。



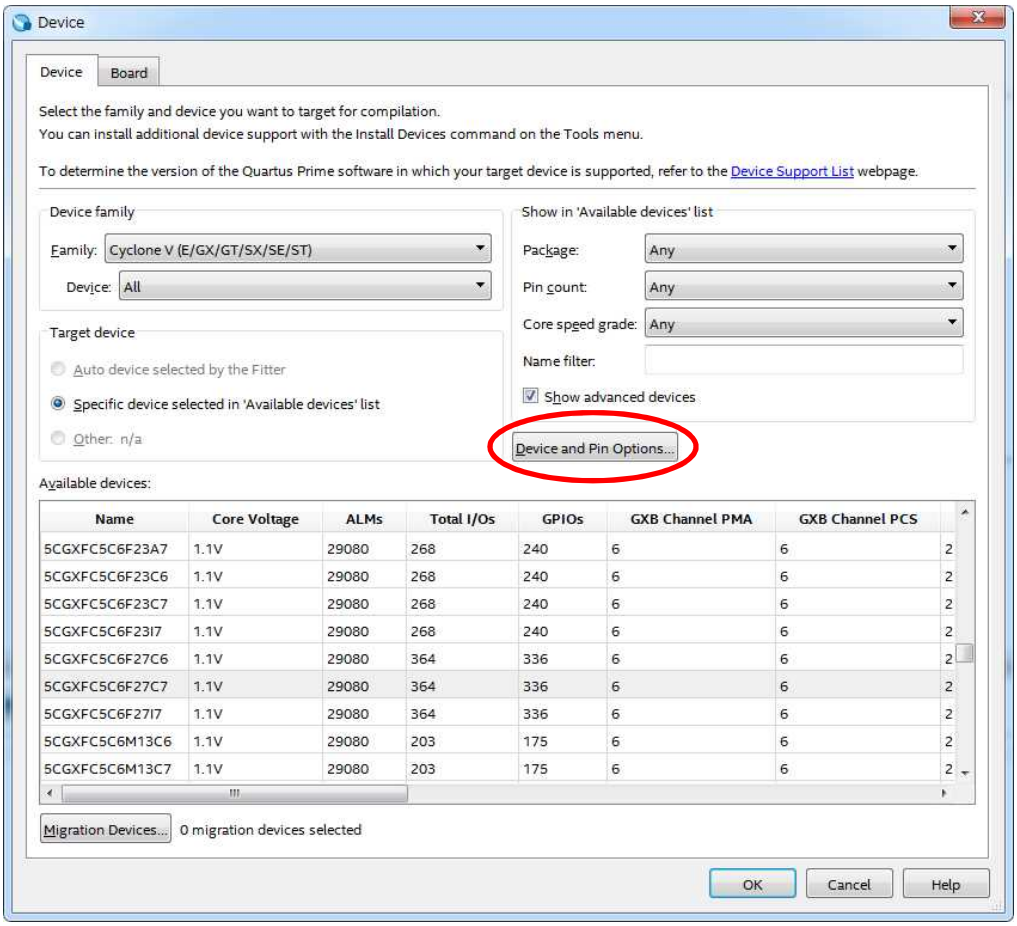
6-2. コンフィグレーション・モードの設定

CPLD とは異なり、 FPGA は SRAM で構成されている揮発性素子であり、ボードの電源を切るとユーザが作成した回路情報は全て消えてしまいますので、専用の ROM に回路情報を格納する必要があります。ボードの電源を入れると、 ROM から FPGA に回路データを転送することで FPGA 内にユーザ回路が形成されます。

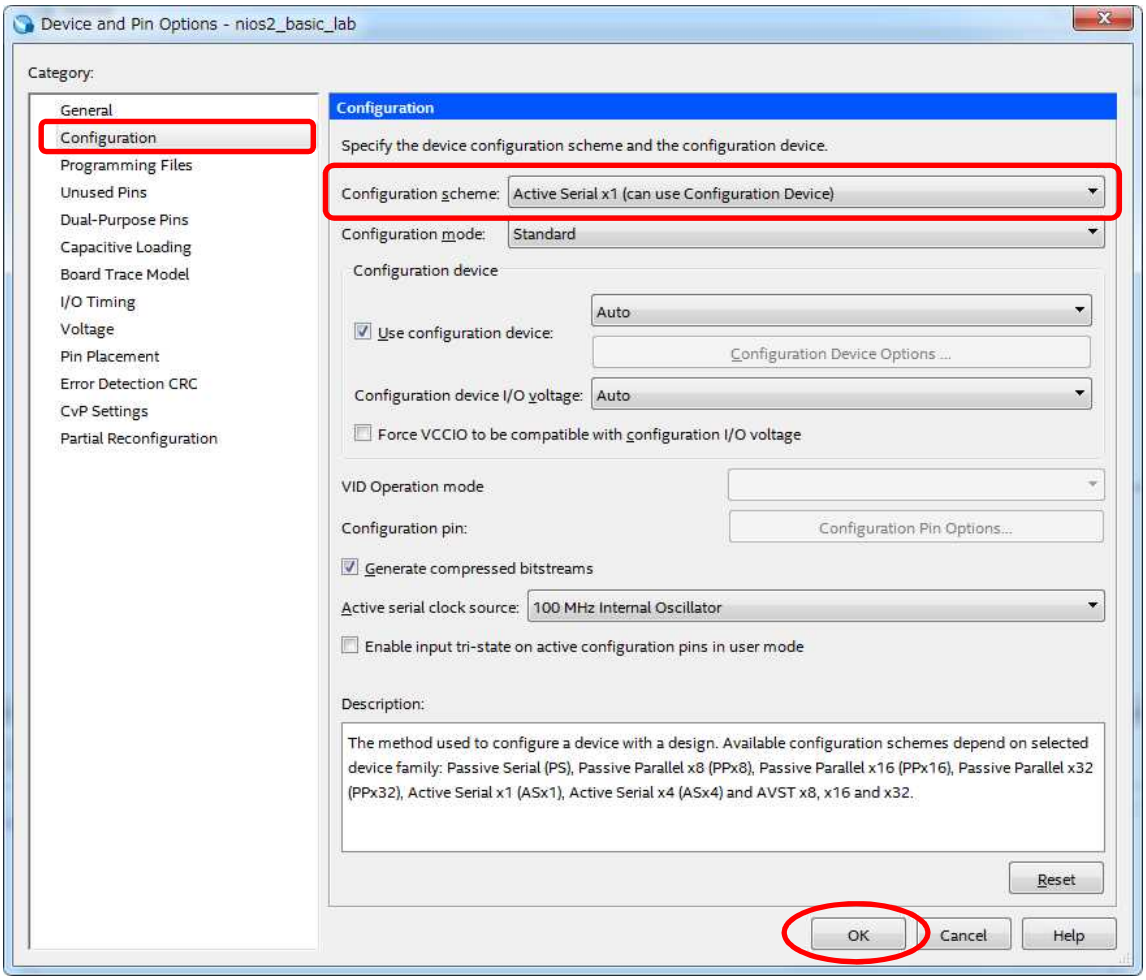
この動作は、コンフィグレーションと呼ばれています。ソフトウェア・エンジニアがボードに実装されている IC に対して初期設定やモード設定を行うときのコンフィグレーションとは意味合いが異なりますので注意してください。

インテル FPGA では様々なコンフィグレーション・モードが存在しますが、ハードウェア・デザインを作成する場合、どのコンフィグレーション・モードを使用するかをユーザ側で設定する必要があります。

1. Quartus® Prime の **Assignments** メニュー ⇒ **Device** を選択します。
2. 起動した Device ウインドウ中央の **Device and Pin Options** ボタンをクリックします。



3. Device and Pin Options ウインドウが起動します。
4. **Category** 欄より **Configuration** を選択します。
5. プルダウン・メニューより、 **Configuration Scheme** の項目に対して、 **Active Serial x1** を選択します。 **[OK]** ボタンをクリックして画面を閉じます。



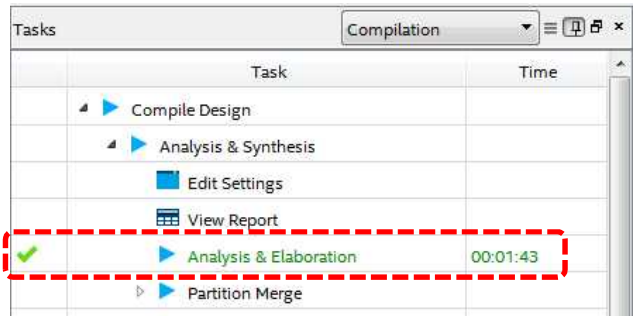
6. Device ウインドウに戻るので、この画面も **[OK]** ボタンをクリックして閉じます。

6-3. 外部端子のピン・アサイン

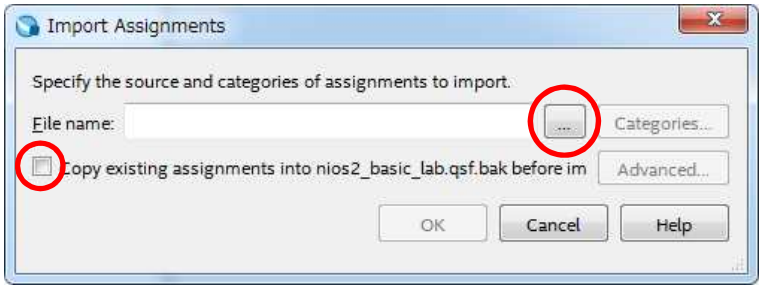
FPGA 内のユーザ回路の信号に、デバイス毎で規定されている外部端子のピン番号をアサインします。

⚠ *本演習では時間節約のため、予め用意されたピン設定用のファイルを読み込むことで、ピン配置を行います。*

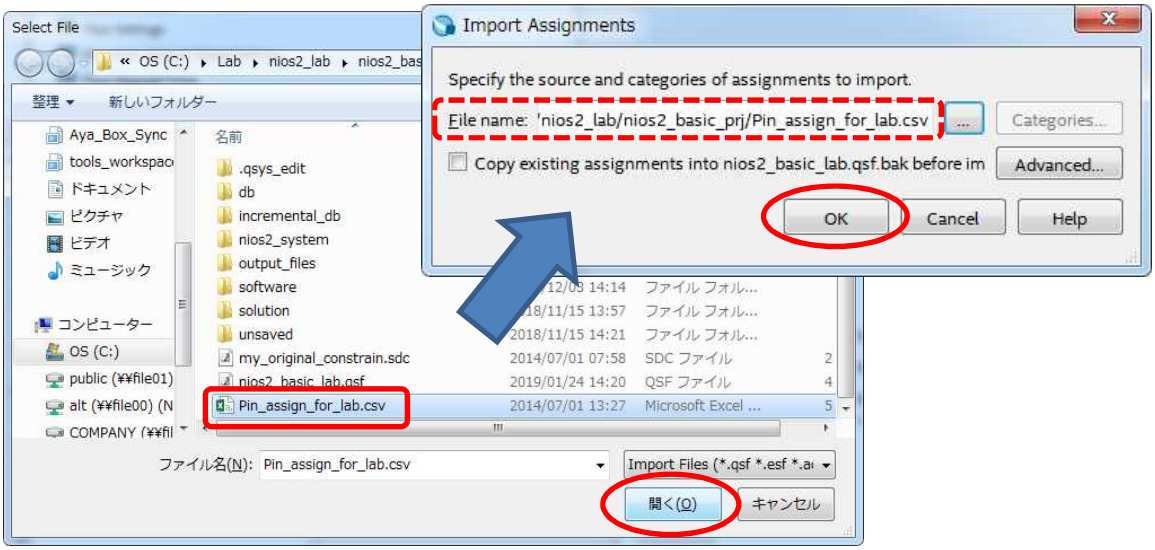
1. **Processing** メニュー ⇒ **Start** ⇒ **Start Analysis & Elaboration** を選択して、デザインの文法チェックを行います。 **Task** 画面の **Analysis & Elaboration** に緑のチェックが付いたら正常終了です。



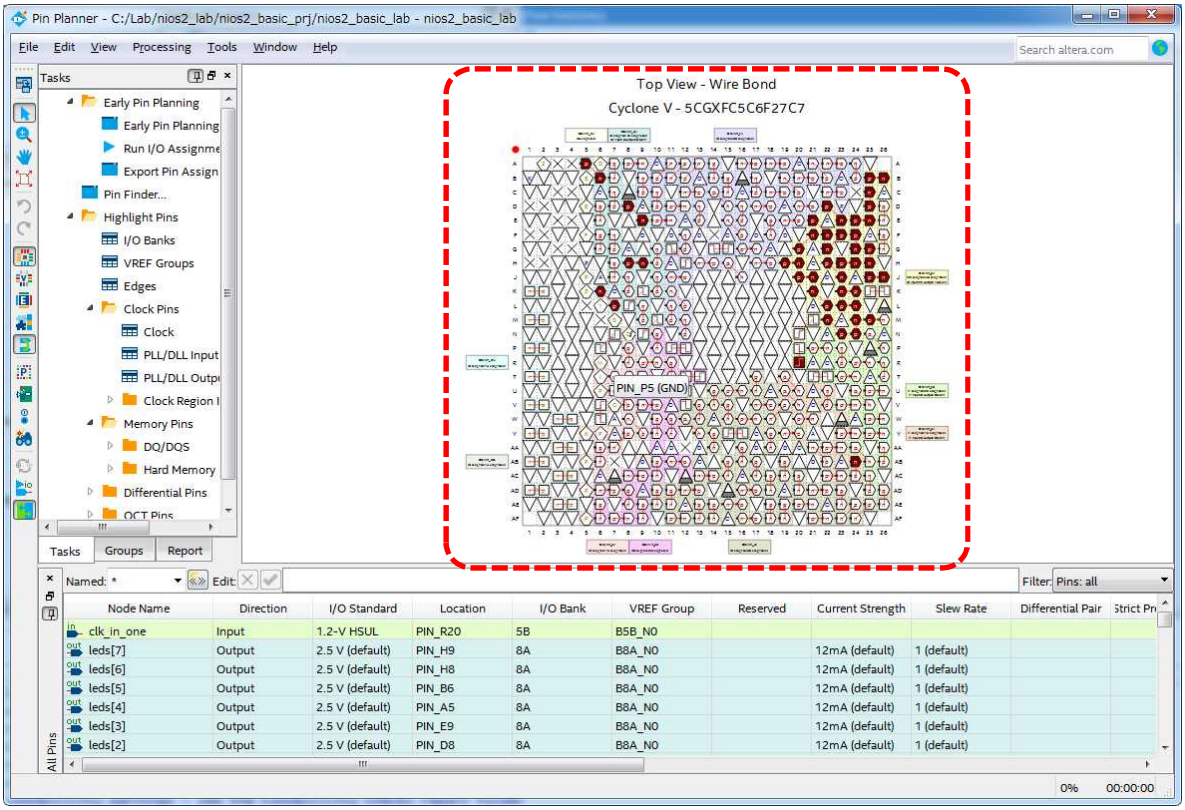
2. Quartus® Prime の **Assignments** メニュー ⇒ **Import Assignments** を選択します。
3. Import Assignments ウインドウが起動します。
4. **Copy existing assignments into nios2_basic_lab.qsf.bak before importing** のチェックを**外し**ます（QSF ファイルのバックアップを残す場合はそのままで OK）。 **File name** 横のブラウザ・ボタンをクリックします。



5. プロジェクト・フォルダから **Pin_assign_for_lab.csv** を選択し **[開く]** ボタンをクリックし、 **File name** 欄にファイル名が表示されたら **[OK]** ボタンをクリックします。



6. Quartus® Prime の **Assignments** メニュー ⇒ **Pin Planner** を選択します。次の図のようにピン情報が表示されれば、ピン配置が正常に反映されたことになります。



6-4. タイミング制約ファイルの設定

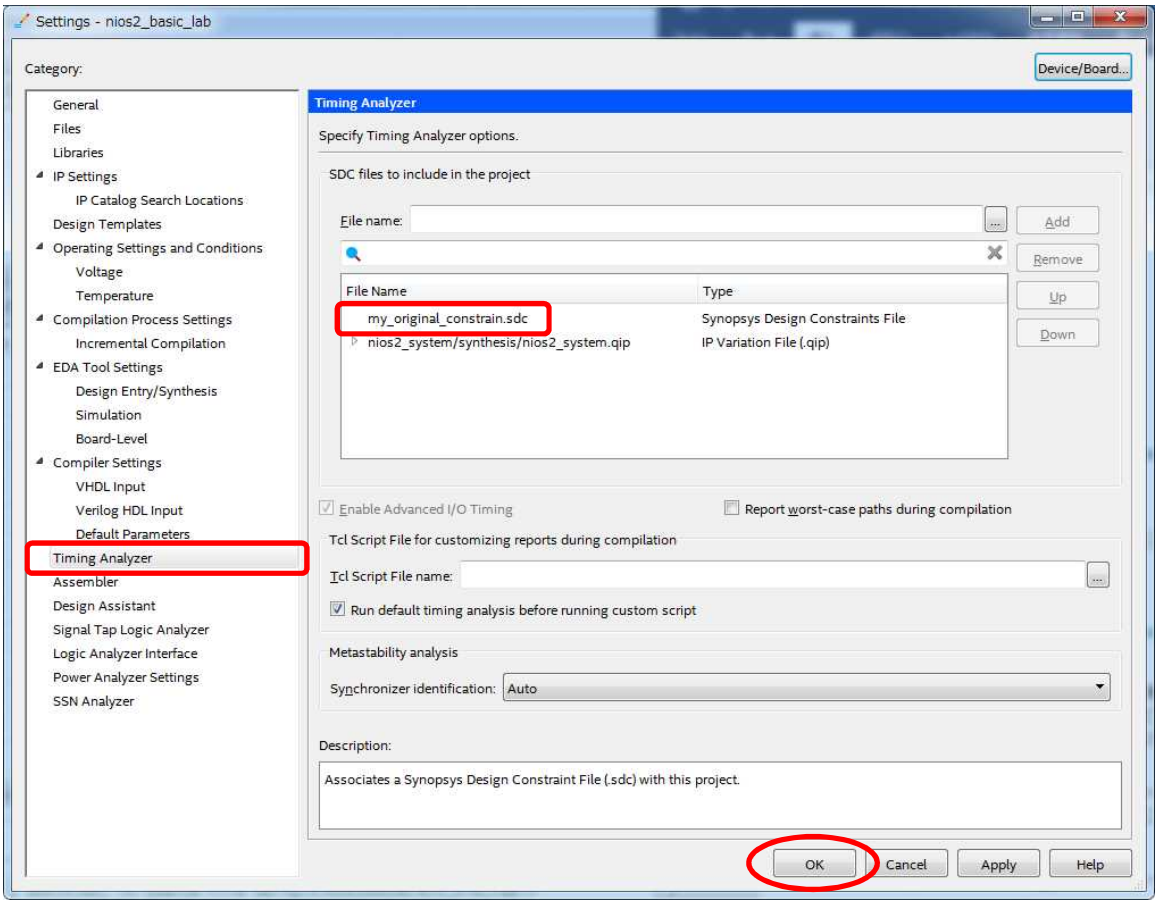
FPGA のハードウェア・デザインのタイミング制約を追加します。タイミング制約は、 Fmax やデバイス内部のレジスタのセットアップ/ホールド時間、デバイスピンからの入出力のタイミングなどの制約を指します。設計したデザインが、指定したタイミング制約を正しく満たすことで、ハードウェア・デザインが電気的に正しく動作することが保証されます。

ここでは、予め本演習用に作成した制約ファイルを読み込むことでタイミング制約を行います。制約の記述方法などについては、詳細になり過ぎるので、ここでは、割愛させて頂きます。

1. Quartus® Prime の **Assignments** メニュー ⇒ **Settings** を選択します。
2. 左欄の Category 枠より **Timing Analyzer** をハイライトします。
3. ブラウズ・ボタンをクリックして、プロジェクト・フォルダ内の **my_original_constrain.sdc** を選択後、**[Add]** ボタンをクリックして、制約ファイルを登録します。

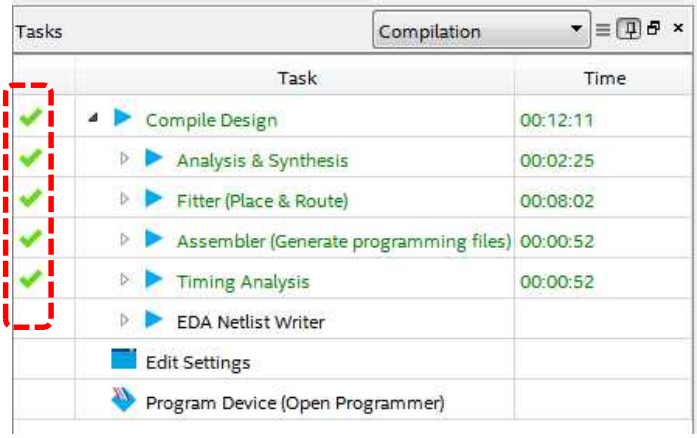
   ⚠ *my_original_constrain.sdc ファイルは、トップ回路図へのタイミング制約ファイルとなります。*

4. ファイルの登録が完了したら、 **[OK]** ボタンをクリックして画面を閉じます。

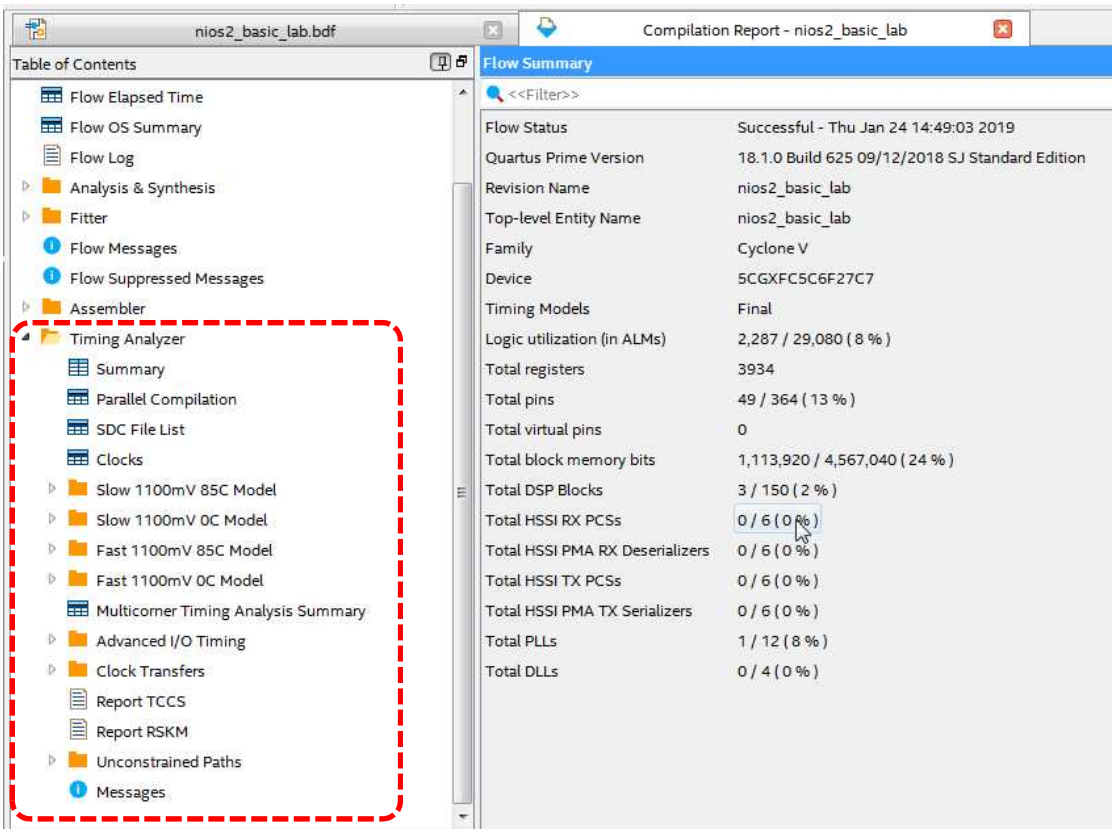


6-5. ハードウェア・デザインのコンパイル

1. Quartus® Prime の Processing メニュー ⇒ Start Compilation を選択します。
2. コンパイルには多少の時間を要します。終了し Tasks ウインドウが図のように Compile Design に緑の ☑ マークがついていれば正常終了です。



3. Timing Analyzer フォルダ内の各種レポートに、赤字になっているフォルダが無いことを確認してください。



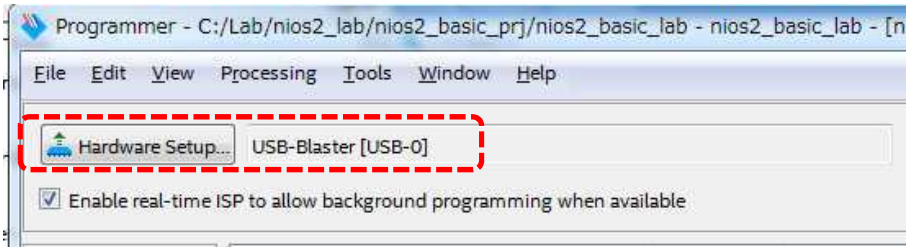
6-6. ダウンロード・ケーブルの接続

1. 今回演習で使用する評価ボード Cyclone® V GX Starter Kit の USB BLASTER ポート (J10) とホスト PC とを付属の USB ケーブルで接続します。
2. 評価用ボードに電源ケーブルを接続し、赤いボタンを押し電源を入れます。

6-7. プログラミング・ツールの設定

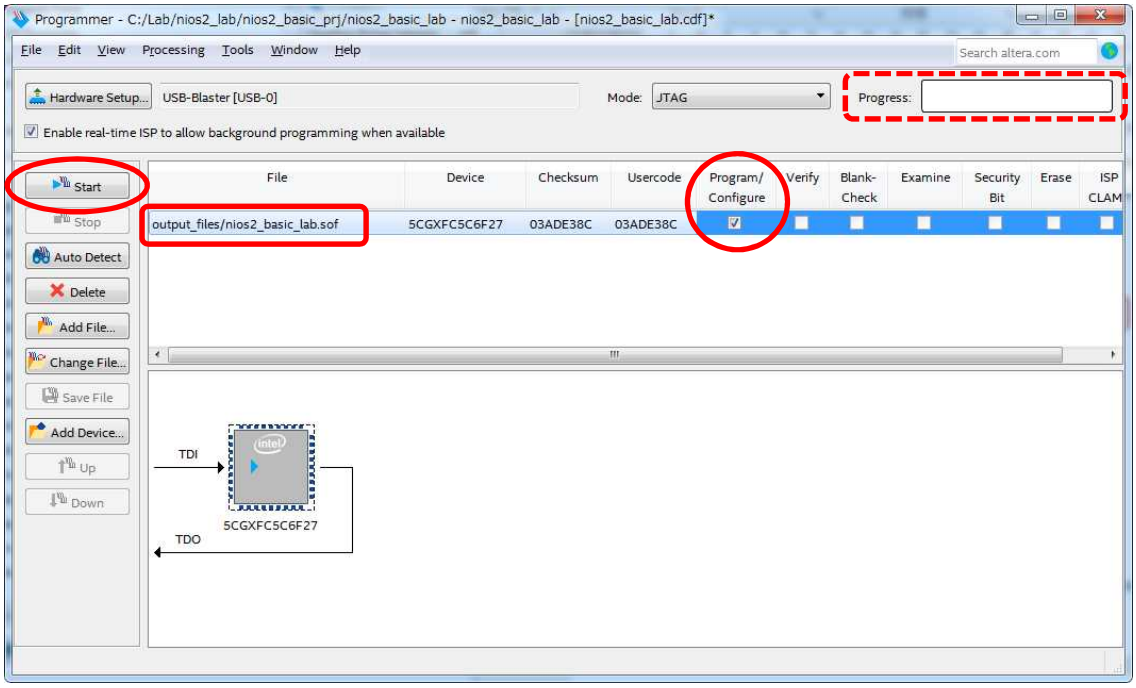
1. Quartus® Prime の **Tools** メニュー ⇒ **Programmer** 選択します。 Programmer ウインドウが表示されます。
2. **Hardware Setup** の項目に **USB-Blaster** が設定されていることを確認します。

   ⚠ *設定されていない場合には、講師に確認してください。*



3. **Auto Detect** ボタンを押して、ボード上のデバイスをサーチします。
4. Select Device が表示されたら、 **5CGXFC5C6** をチェックして **[OK]** ボタンをクリックします。
5. **File** 項目の **<none>** となっている箇所をダブル・クリックして、 **SOF ファイル**を選択します。

   output_files/nios2_basic_lab.sof
6. Program / Configure のチェック・ボックスにチェックを入れます。
7. **Start** ボタンをクリックすると、 FPGA にハードウェア・デザインの書き込みが開始されます。
8. Programmer ウインドウの **Progress** の項目が **100%** になれば書き込み完了です。



7. ソフトウェアの実行

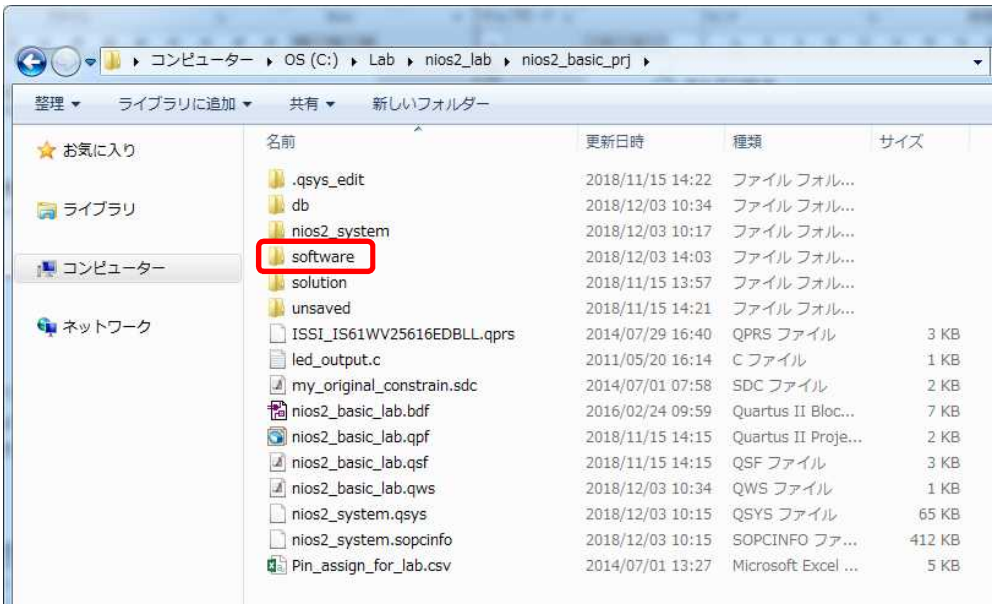
Nios® II Software Build Tool (Nios® II SBT) は、 Nios® II 向けの統合開発環境になります。今回の演習では、Nios® II SBT を使用して、 C ソース・コードの作成やビルド、およびターゲットの評価ボードへのプログラム・ダウンロードまでを実施します。

作成する C ソース・コードは Nios® II の PIO を経由して、評価ボードの LED を点滅させる簡単なプログラムになります。

7-1. Nios® II SBT の起動

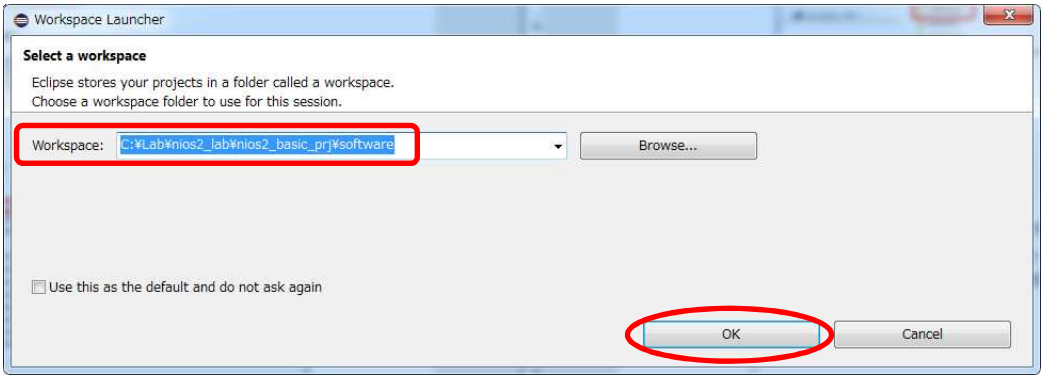
1. Quartus® Prime の現在のハードウェア・プロジェクト・フォルダ（推奨）に、 Nios® II SBT 用のワークスペースを作成します。例えば、今回の演習で使用している Quartus® Prime の現在のハードウェア・プロジェクトは、**C:¥Lab¥nios2_lab¥nios2_basic_prj** フォルダになりますが、このフォルダの下にワークスペース名 **software** というフォルダを手動で作成してください。

   C:¥Lab¥ nios2_lab¥nios2_basic_prj¥software

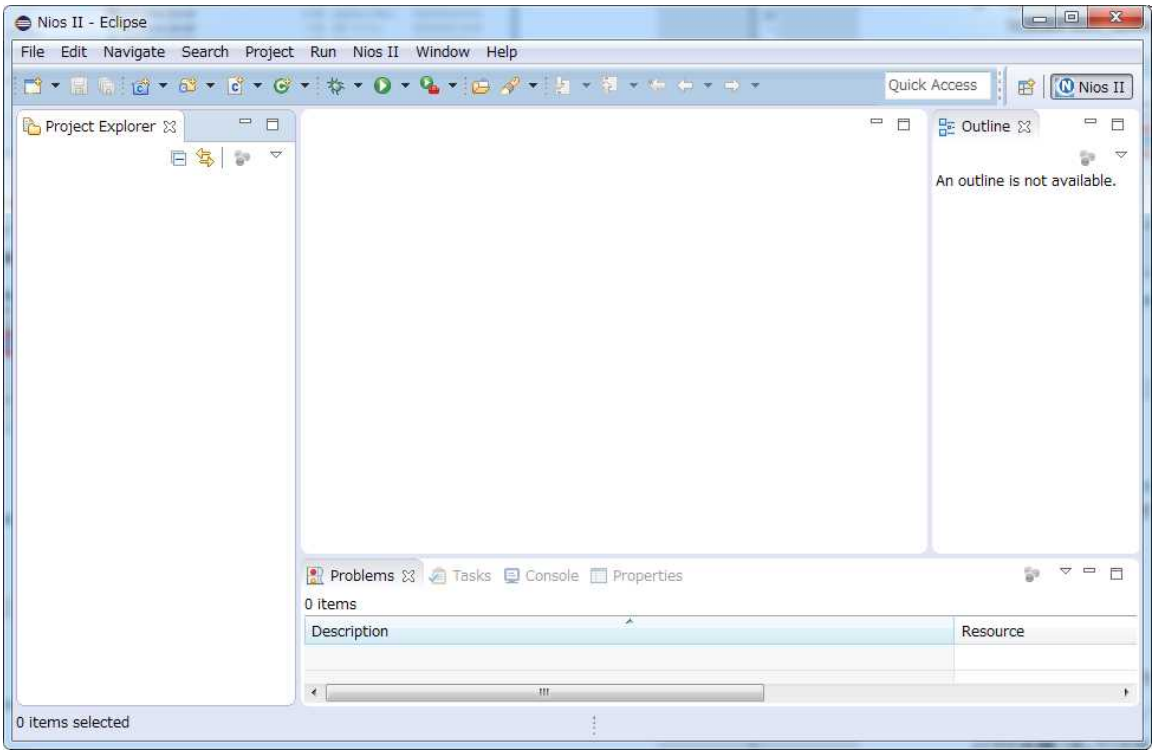


2. ショートカットや Windows® のスタートメニューから、 **Nios II SBT** を起動します。

3. Workspace Launcher が起動した場合、前述のワークスペース（**C:¥Lab¥ nios2_lab¥nios2_basic_prj¥software**）を指定して、 **[OK]** ボタンをクリックします。



4. Nios® II SBT が起動します。

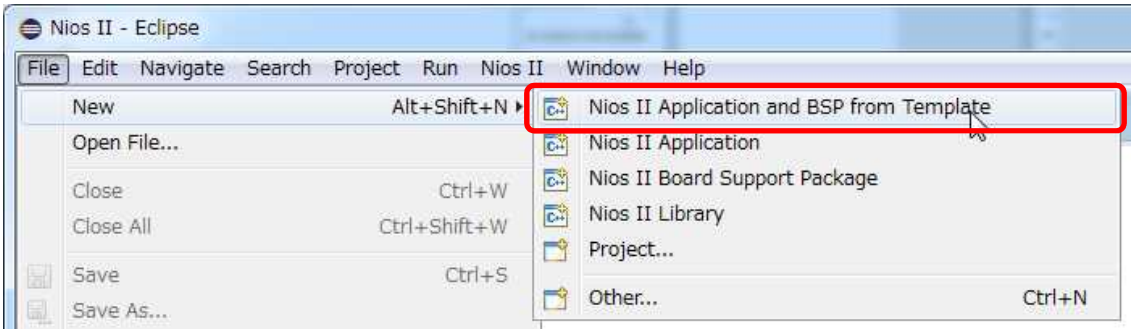


7-2. ソフトウェア・プロジェクトの作成

Nios® II SBT の New Project ウィザードを用いて、ソフトウェア・プロジェクトを作成します。

以下の手順に従ってください。

1. Nios® II SBT のメイン画面の **File** メニュー ⇒ **New** ⇒ **Nios II Application and BSP from Template** を選択します。

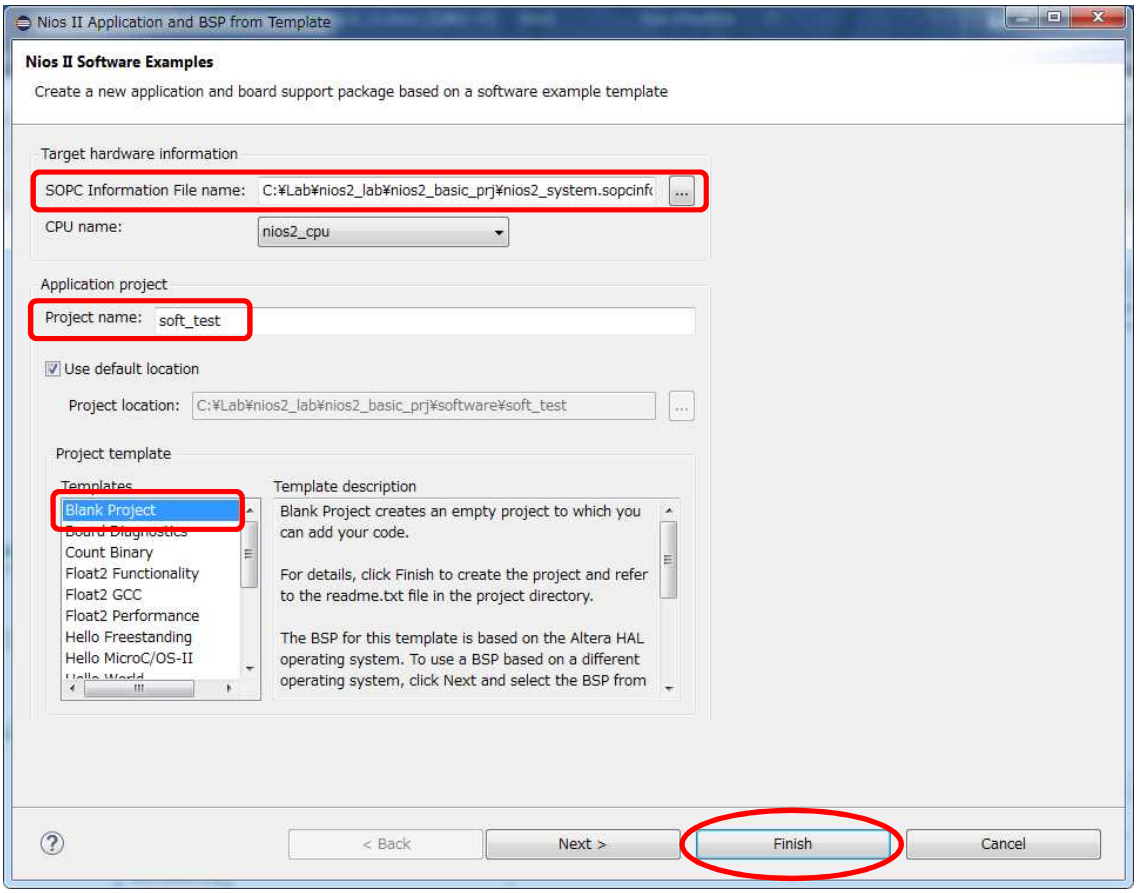


2. Nios® II Applocation and BSP from Template ウインドウが起動します。

3. ソフトウェア・プロジェクトを作成する際に、組み込みシステムのハードウェア情報が記述されたシステム定義ファイルを指定します。 **Target hardware information** 欄の **SOPC Information File name** の右側にあるボタンをクリックして、ハードウェア・デザインを作成したフォルダにある Platform Designer が生成した **nios2_system.sopcinfo** を選択します。

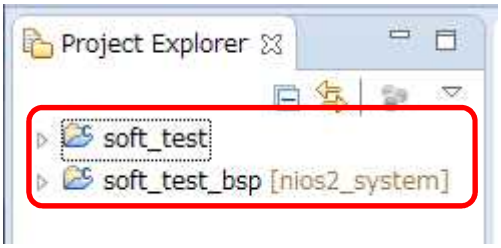
4. **Application project** 欄の **Project name** には **soft_test** をタイプします。

5. **Templates** 欄からリストの一番上の **Blank Project** を選択して、 **[Finish]** ボタンをクリックします。



6. 新しいプロジェクトが作成されて、 **Project Explorer** の項目に **soft_test** と **soft_test_bsp** が追加されます。

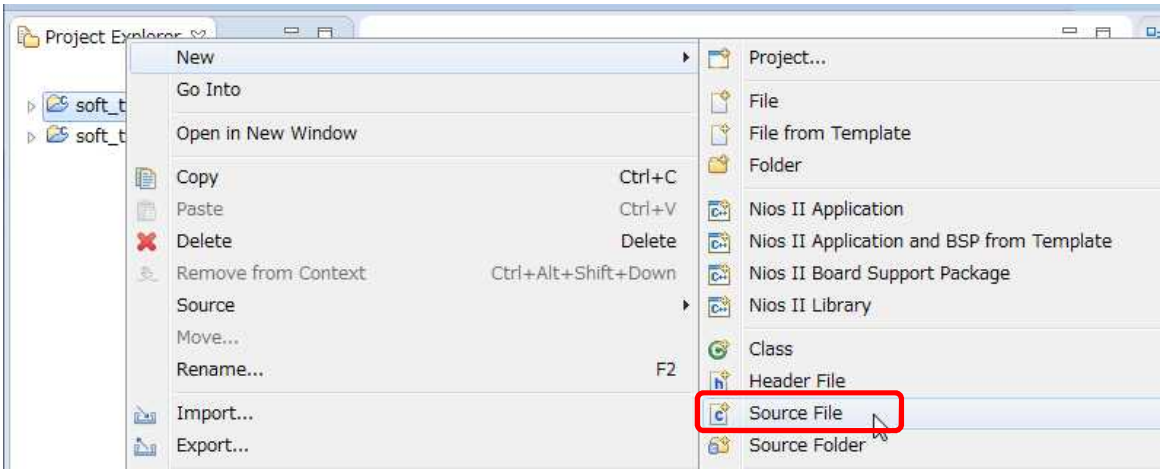
⚠ *プロジェクトの本体は、 C:¥Lab¥nios2_lab¥nios2_basic_prj¥software 以下に作成されます。*



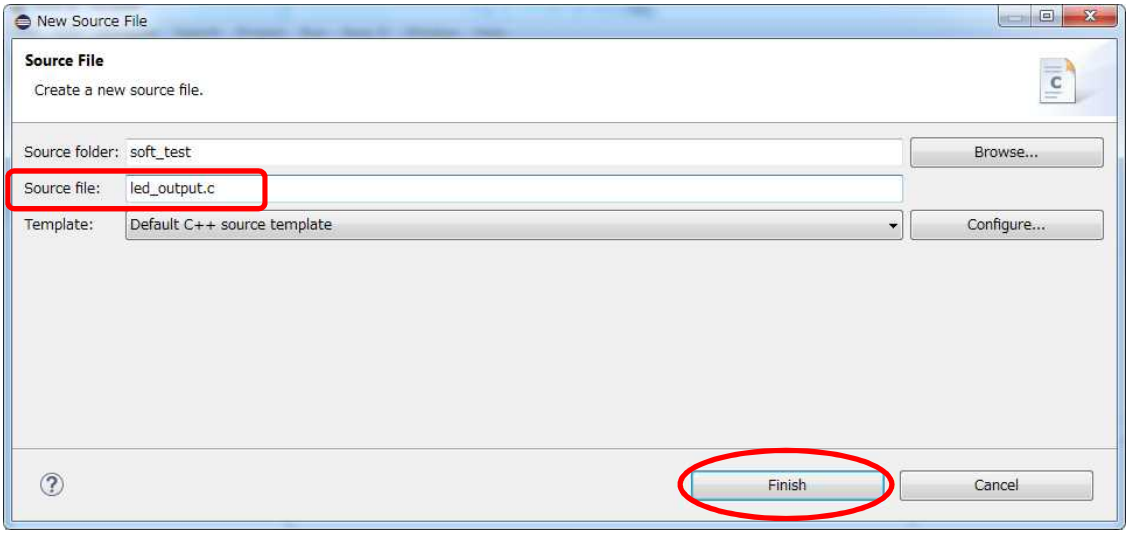
7-3. C ソース・コードの作成

Nios® II SBT の New Source File ウィザードを用いて、ソース・コードを作成します。

1. **soft_test** フォルダをハイライトして、**右クリック**から **New** ⇒ **Source File** を選択します。

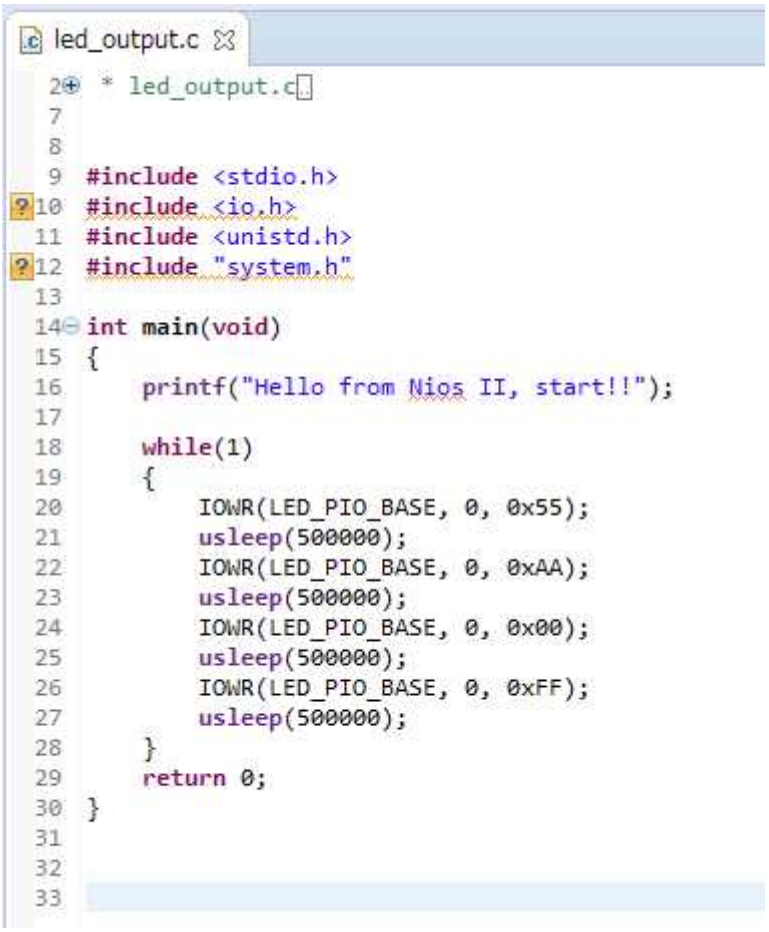


2. New Source File ウインドウの **Source folder** 欄へ、**Browse** ボタンよりソフトウェア・プロジェクトのアプリケーション・プロジェクト（_bsp がついていない方のフォルダ）を選択します。 **Source file** 欄へ、**led_output.c** のソフトウェアのファイル名を拡張子 .c をつけて入力し、**[Finish]** ボタンをクリックします。



3. 次の図のように、 **led_output.c** をハイライトして、図に掲載されている内容と同じ内容の C ソース・コードを、右側のメイン・エディタ内に直接記述してください。

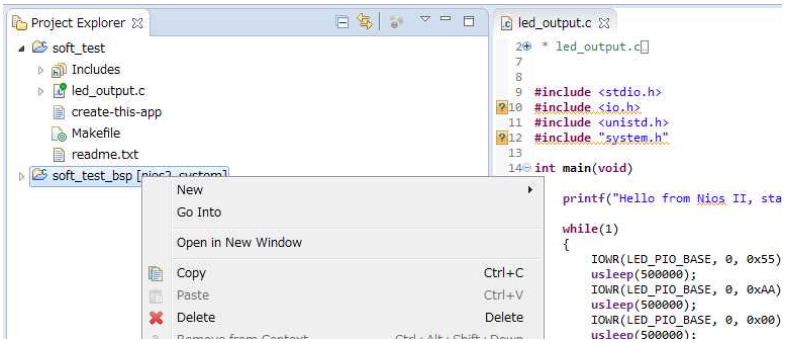
⚠ *時間短縮のため、ハードウェア・プロジェクト・フォルダ内に完成した led_ouput.c ファイルがありますので、それをテキスト・エディタで開いて、コピー&ペーストしても構いません。*



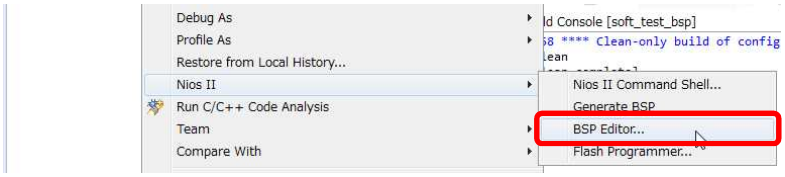
4. 記述が終了したら、 **File** メニュー ⇒ **Save** を選択して、ファイルを保存します。

7-4. ソフトウェア・プロジェクトのビルドとプログラムの実行

1. システムの設定を確認します。 **soft_test_bsp** フォルダを**右クリック**して、 **Nios II ⇒ BSP Editor** を選択します。

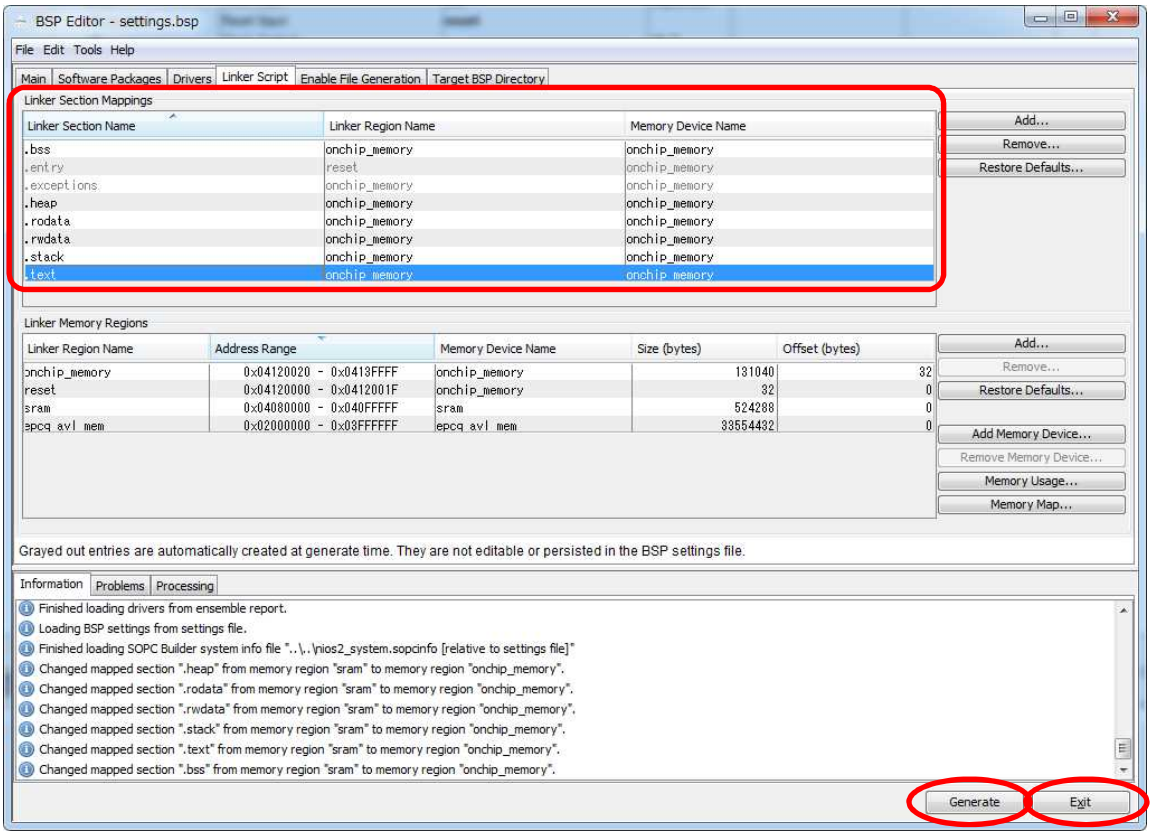


～～～～～～～～～～～～～≪中略≫～～～～～～～～～～～～～



2. BSP Editor が起動します。

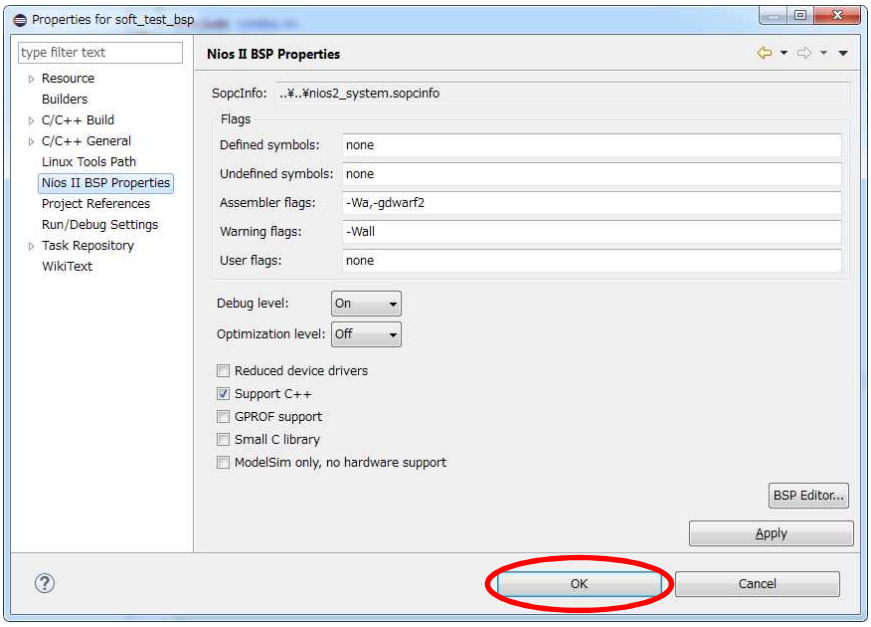
3. **Linker Script** タブを選択して、各リンカ・セクション （.bss, .text 等） の設定を下図のように全て、**onchip_memory** に設定（Linker Region Name をクリック ⇒ プルダウンから選択）したら、**[Generate]** ボタンをクリックし、 **[Exit]** ボタンで画面を閉じます。



⚠ *この設定によって本プログラムは、ワーク・メモリとして onchip_memory のみを使用する設定となります。現時点では、外部 SRAM 等のハードウェアの動作が保証されないため、オンチップ・メモリを使用して Nios® II の初期動作の確認を行います。*

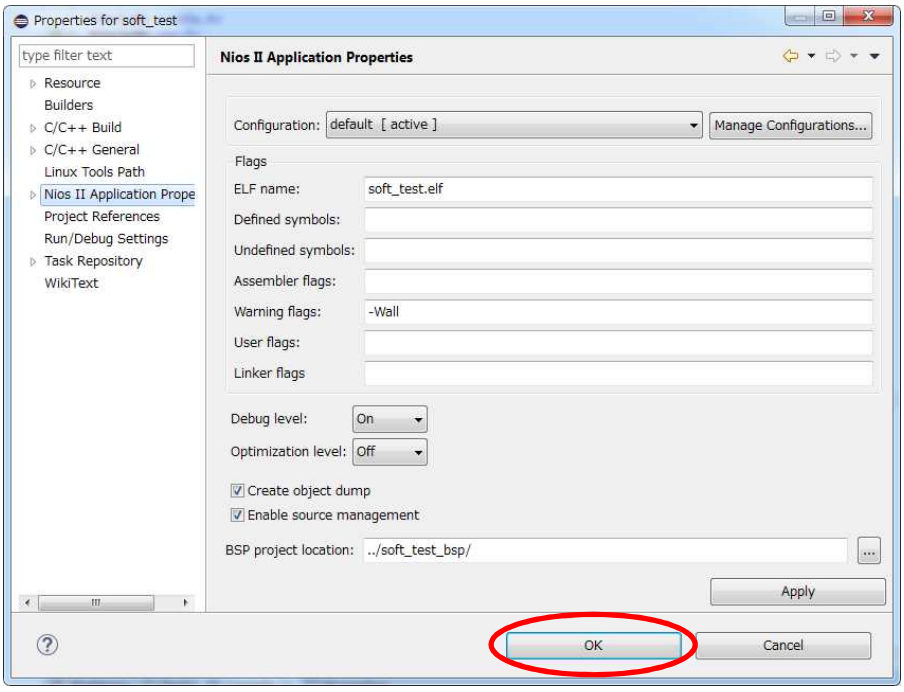
4. **soft_test_bsp** フォルダを右クリックして、 **Properties** を選択します。起動画面内のカテゴリで **Nios II BSP Properties** をハイライトして設定項目を確認します。ここでは、 BSP プロジェクトのデバッグ・レベルの設定や最適化レベルの設定などを行います。

   本演習では、デフォルトの状態で進めていきますので、 **[OK]** ボタンをクリックします。



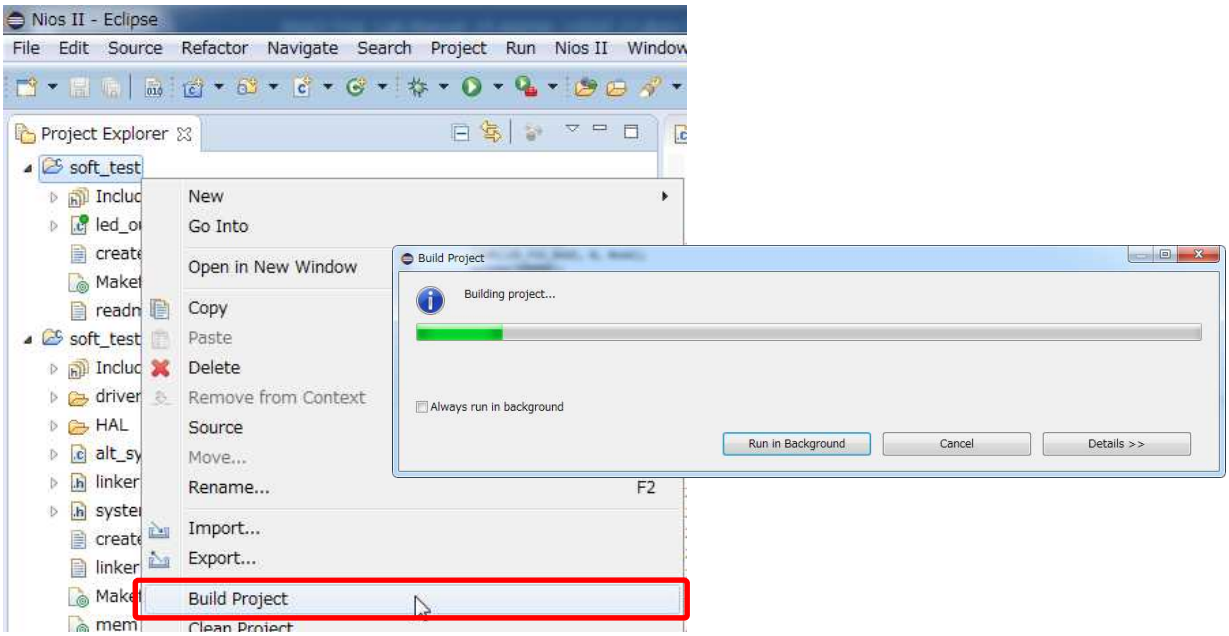
5. 次に **soft_test** フォルダを右クリックして、 **Properties** を選択します。起動画面内のカテゴリで **Nios II Application Properties** をハイライトして設定項目を確認します。ここでは、アプリケーション・プロジェクトのデバッグ・レベルの設定や最適化レベルの設定などを行います。

   本演習では、デフォルトの状態で進めていきますので、 **[OK]** ボタンをクリックします。



6. ソフトウェアをビルド（Build）します。 Nios® II SBT の左枠のアプリケーション・プロジェクトのフォルダ（_bsp がついていない方）を選択し、右クリック ⇒ Build Project をクリックします。

   Console にエラーメッセージが出なければビルド完了です。

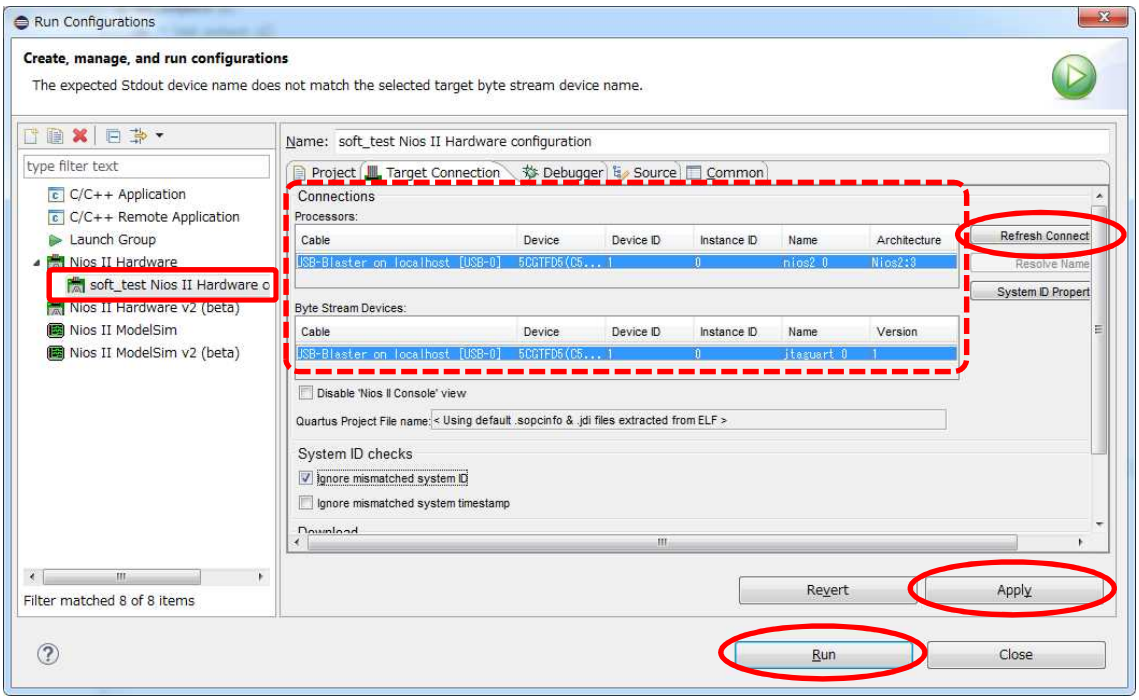


7. ビルドしたソフトウェアをターゲットのシステムで実行します。 Nios® II SBT の左枠のアプリケーション・プロジェクトのフォルダ（_bsp がついていない方）を選択し、右クリック ⇒ **Run As** ⇒ **Nios II Hardware** をクリックします。

8. Run Configurations 画面が起動したら、 **Nios II Hardware** フォルダの **soft_test Nios II Hardware configuration** がハイライトされていることを確認します。

9. **Target Connection** タブを選択して、右上の **[Refresh Connections]** ボタンをクリックします。図のように USB-Blaster が検出して、ターゲット・ボードとの接続が確認できます。

10. 右下にある **[Apply]** ボタンをクリックして、その下の **[Run]** ボタンをクリックします。



11. Console ウインドウに、C 言語ソース上に記載した printf() 関数の出力キャラクタ（例：Hello from Nios II, start!!）が確認できます。同時に、開発ボード上の緑色 LED の点滅も確認してください。確認したら、赤いボタンをクリックしてプログラムを終了します。

～ 以上で演習本編はすべて終了です。 ～

～ 時間が余った方は、次のオプション演習へ進んでください ～

【 補足 】 システム・ヘッダ・ファイルの自動生成 とアルテラ専用マクロ

Nios® II ソフトウェアでは、 system.h というシステム・ヘッダ・ファイルをツールが自動生成します。

Nios® II や Platform Designer のハードウェア情報（各ペリフェラルのベース・アドレス等のアドレス・マップ情報や IRQ の優先順位など）が記述されており、アプリケーション・コードでインクルードするだけでペリフェラルにアクセスができるようになります。

インテル FPGA では専用マクロが幾つか用意されており、今回の題材では、その中の一つとして IOWR マクロを使用しました。これらのマクロは、ヘッダ・ファイル io.h をインクルードすることで利用できます。

IOWR(LED_PIO_BASE, 0, 0x55)

ベース・アドレス 0x02201020（本演習では、別のアドレス値にアサインされていますが、このアドレスは構成の仕方によって異なり、ほとんどのケースでユーザが任意に指定可能です）にマッピングされている led_pio ペリフェラル内のレジスタ・アドレス 0 に 0x55 をライトします。 led_pio のベース・アドレスは、 system.h 内で LED_PIO_BASE として定義されています。

led_output.c

```
#include <stdio.h>
#include <io.h>
#include "system.h"

int main(void)
{
    printf("Hello from Nios II, start!!");

    while(1)
    {
        IOWR(LED_PIO_BASE, 0, 0x55);
        usleep(500000);
        IOWR(LED_PIO_BASE, 0, 0xAA);
        usleep(500000);
        IOWR(LED_PIO_BASE, 0, 0x00);
        usleep(500000);
        IOWR(LED_PIO_BASE, 0, 0xFF);
        usleep(500000);
    }
    return 0;
}
```

system.h

```
/*
 * led_pio configuration
 *
 */

#define LED_PIO_NAME "/dev/led_pio"
#define LED_PIO_TYPE "altera_avalon_pio"
#define LED_PIO_BASE 0x02201020
#define LED_PIO_SPAN 16
#define LED_PIO_DO_TEST_BENCH_WIRING 0
#define LED_PIO_DRIVEN_SIM_VALUE 0
#define LED_PIO_HAS_TRI 0
#define LED_PIO_HAS_OUT 1
#define LED_PIO_HAS_IN 0
#define LED_PIO_CAPTURE 0
#define LED_PIO_DATA_WIDTH 4
#define LED_PIO_EDGE_TYPE "NONE"
#define LED_PIO_IRQ_TYPE "NONE"
#define LED_PIO_BIT_CLEARING_EDGE_REGISTER 0
#define LED_PIO_FREQ 100000000
#define ALT_MODULE_CLASS_led_pio altera_avalon_pio

/*
 * pll configuration
```

usleep(500000) は、 500000 us つまり 0.5 秒だけプログラムの実行を停止する為の関数（HAL API）です。

8. Memory Test の実行（オプション）

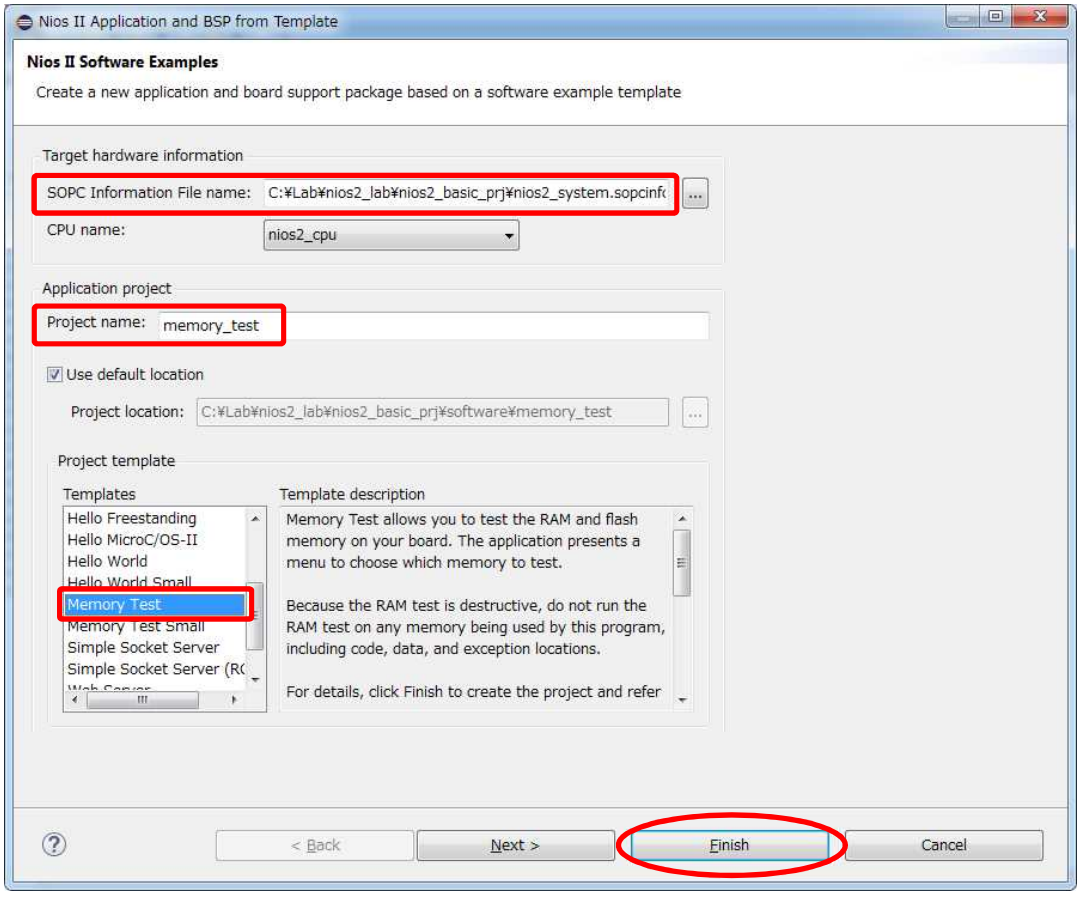
これまでの手順で Nios® II のプログラムを JTAG 経由でオンチップ・メモリにダウンロードし動作させるところまで成功しました。ここでは、外部メモリの動作が正しいことを確認するために予め用意されている Memory Test プログラムを動作させて、外部 SRAM の動作について確認を行います。

8-1. ソフトウェア・プロジェクトの作成

Nios® II SBT の New Project ウィザードを用いて、ソフトウェア・プロジェクトを作成します。

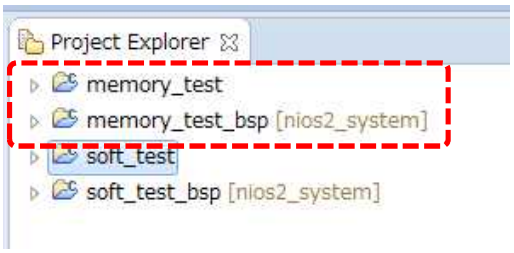
以下の手順に従ってください。

1. Nios® II SBT のメイン画面の **File** メニュー ⇒ **New** ⇒ **Nios II Applocation and BSP from Template** を選択します。
2. Nios II Applocation and BSP from Template ウインドウが起動します。
3. ソフトウェア・プロジェクトを作成する際に、組み込みシステムのハードウェア情報が記述されたシステム定義ファイルを指定します。 **Target hardware information** 欄の **SOPC Information File name** の右側にあるブラウザ・ボタンをクリックして、ハードウェア・デザインを作成したフォルダにある Platform Designer が生成した **nios2_system.sopcinfo** を選択します。
4. **Application project** 欄の **Project name** には **memory_test** とタイプします。
5. **Templates** 欄から **Memory Test** を選択して、 **[Finish]** ボタンをクリックします。



6. 新しいプロジェクトが作成されて、 **Project Explorer** の項目に **memory_test** と **memory_test_bsp** が追加されます。

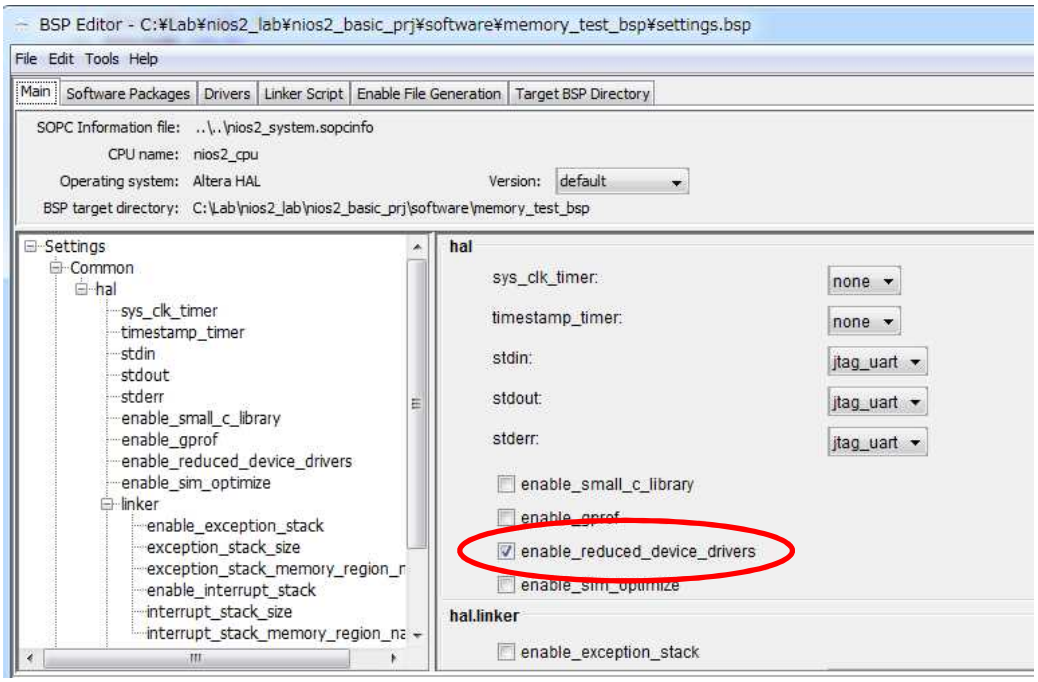
   ※プロジェクトの本体は、 C:¥Lab¥nios2_lab¥nios2_basic_prj¥software 以下に作成されます。



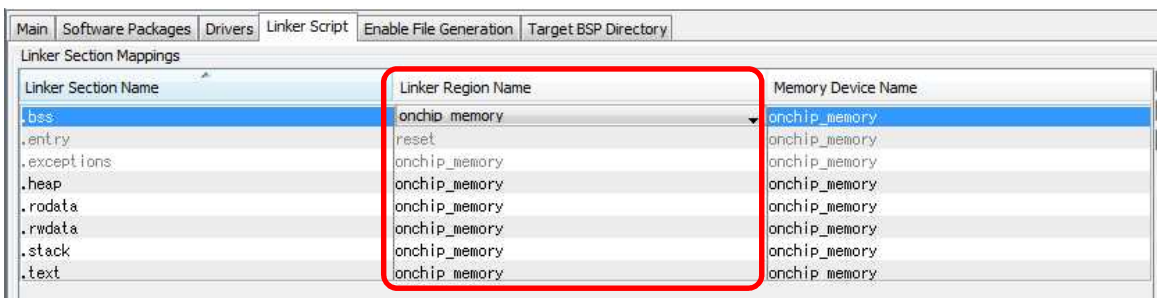
8-2. ソフトウェア・プロジェクトのビルドとプログラムの実行

1. システムの設定を確認します。 **memory_test_bsp** フォルダを右クリックして、 **Nios II ⇒ BSP Editor** を選択します。

2. BSP Editor が起動したら、 **Main** タブにて **enable_rediced_device_driver** の項目をチェックします。この設定で、ビルドするソフトウェアのサイズを小さくすることができます（ただし、一部機能が制限されます）。



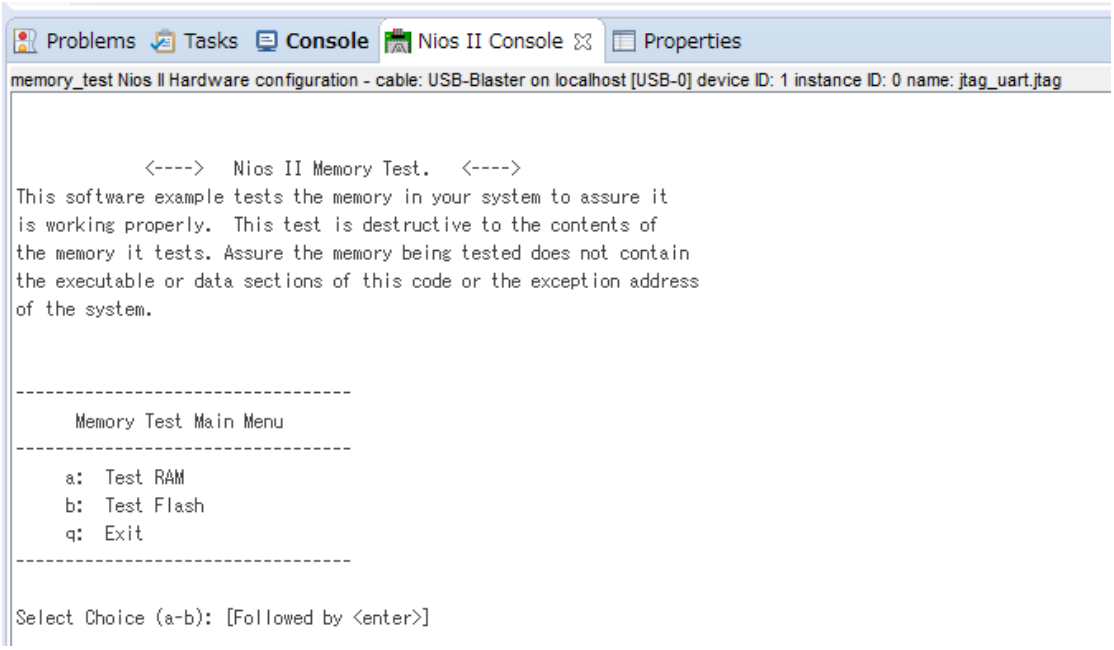
3. 次に **Linker Script** タブを選択して、各リンカ・セクション（.bss, .text 等）の設定を下図のように全て、**onchip_memory** に設定（**Linker Region Name** をクリックで選択）したら、右下の **[Generate]** ボタンをクリックし、 **[Exit]** ボタンで画面を閉じます。



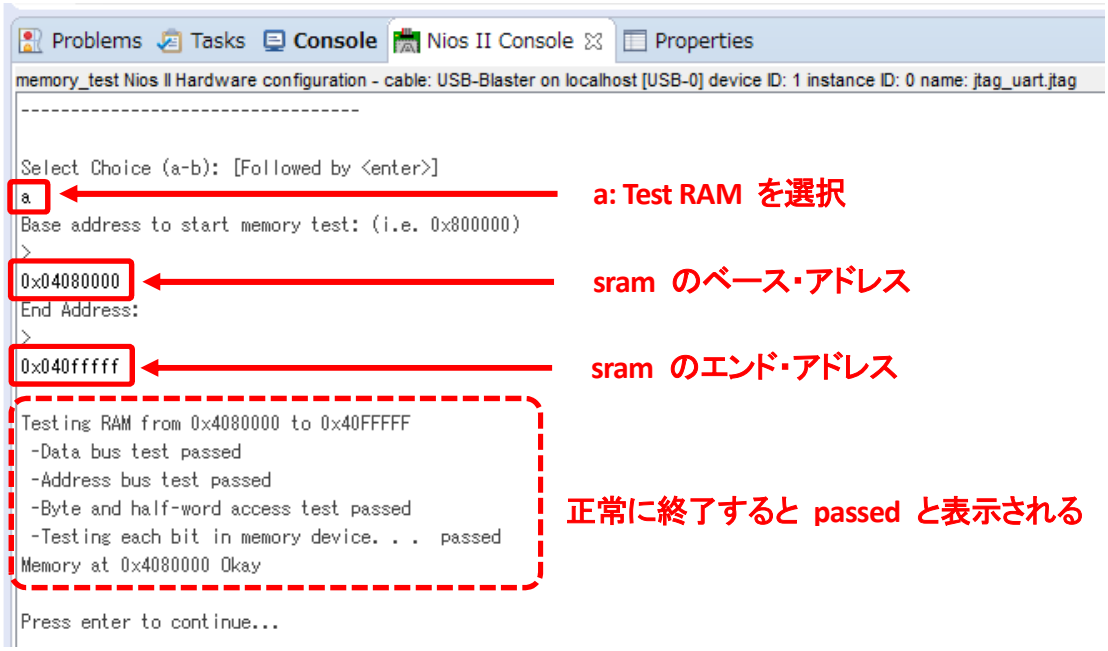
4. ソフトウェアをビルドします。 Nios® II SBT の左枠のアプリケーション・プロジェクトのフォルダ（**memory_test**）を選択し、右クリック ⇒ **Build Project** をクリックします。

5. ビルドが完了したら、ビルドしたソフトウェアをターゲットのシステムで実行します。 Nios® II SBT の左枠のアプリケーション・プロジェクトのフォルダ（**memory_test**）を再度選択し、右クリック ⇒ **Run As** ⇒ **Nios II Hardware** をクリックします。

6. Memory Test プログラムが実行され Nios® II のコンソール画面に以下のように表示されます。



7. まずは、外部 SRAM インタフェースの動作を確認します。以下のように入力します。正常終了すると以下のように表示されます。全て passed となっていれば問題ありません。

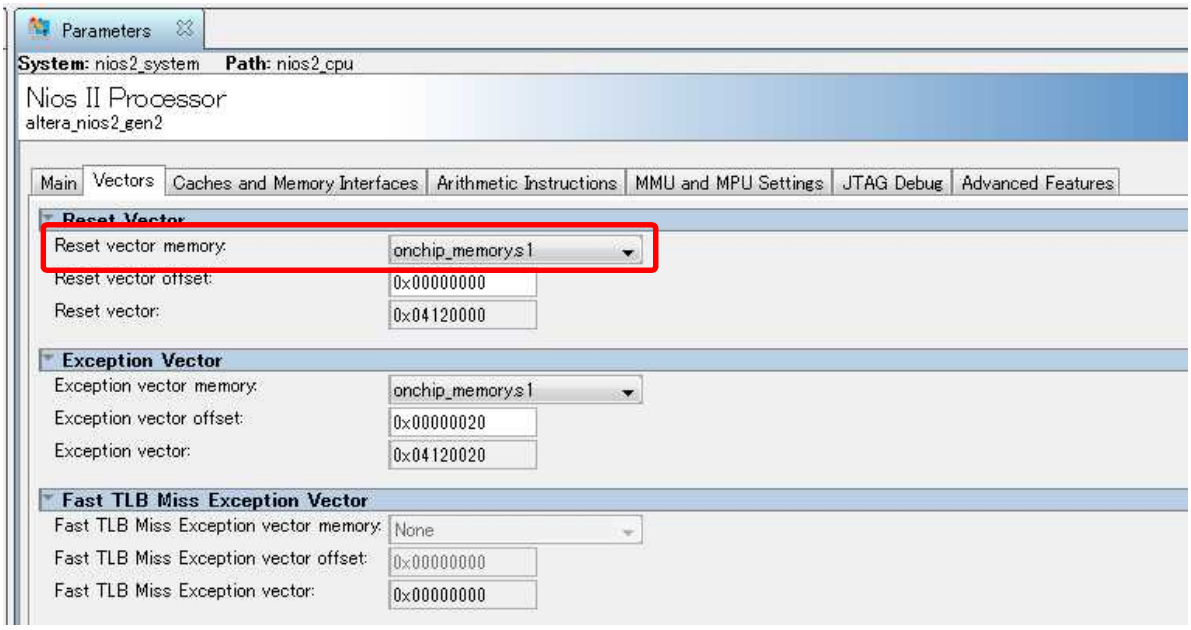


8. SRAM の動作を確認することができたので、プログラムを終了  します。

9. オンチップ・メモリからのブート（オプション）

使用する文字は、以下のルールに従ってください。これまでの手順で、 Nios® II のソフトウェア・プログラムを JTAG 経由のダウンロードで動作させることに成功しました。また、 Memory Test にて外部 SRAM が正常にアクセスできることを確認しました。次に Nios II のソフトウェア・プログラムを FPGA デバイス内のオンチップ・メモリからブートする方法について進めていきます。

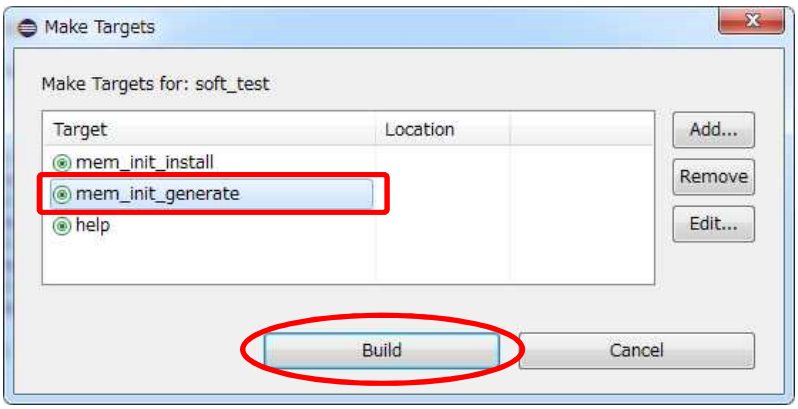
まず前提として、下図のように Platform Designer 上にある Nios® II のリセット・ベクタの設定がオンチップ・メモリに設定されていることを確認してください。この設定により、 FPGA 起動後に Nios® II がリセット・ベクタで指定されたアドレスからソフトウェア・プログラムをロードしブートが開始します。



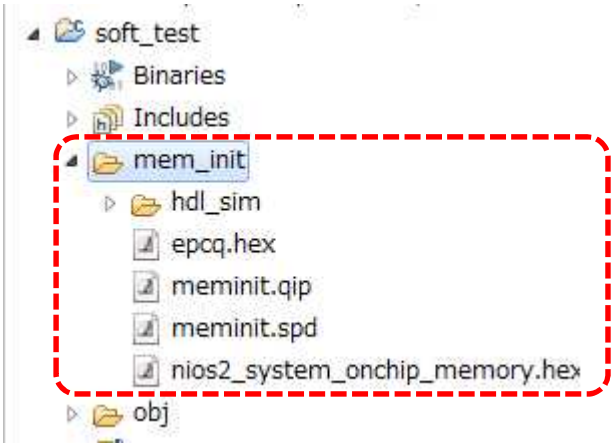
9-1. Nios® II SBT での HEX ファイルの生成

1. Nios® II SBT にてビルド済のアプリケーション・ソフトウェア・プロジェクト（本例では先の演習で動作確認した soft_test を使用します）を右クリックし、プルダウン・メニューから **Make Targets** ⇒ **Build** を選択します。

2. Make Targets 画面が起動したら、 **mem_init_generate** をハイライトして、 **[Build]** ボタンをクリックします。

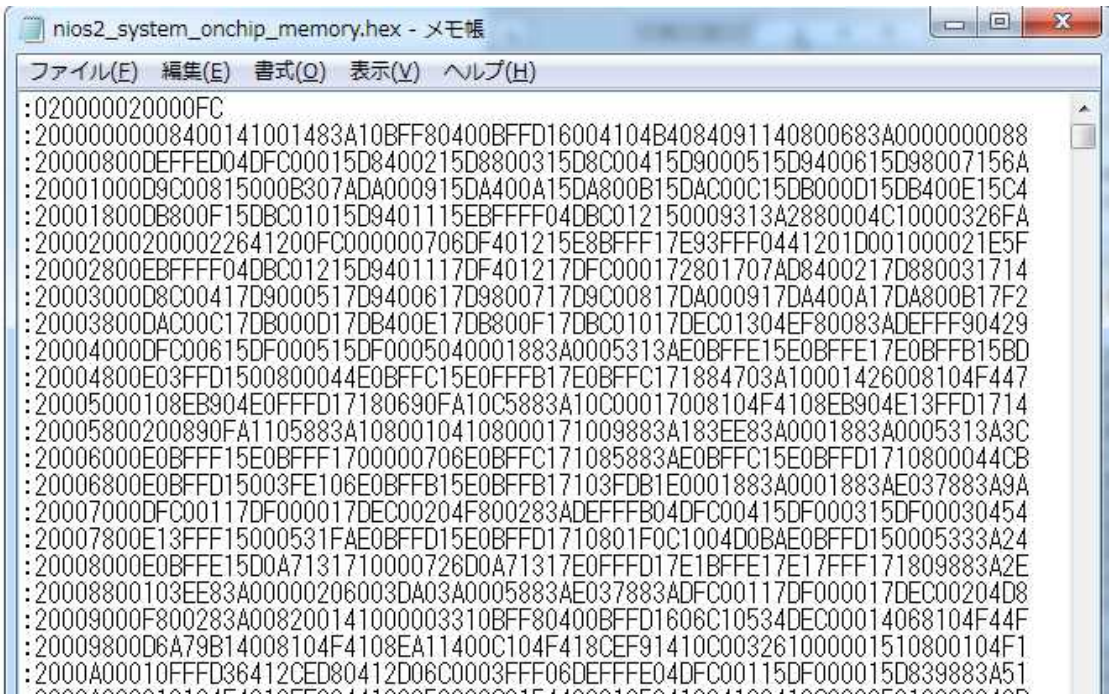


3. この操作によって、 mem_init フォルダがアプリケーション・プロジェクトの中に生成されます。



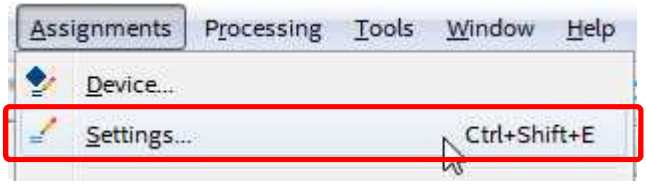
4. mem_init フォルダ内には、**meminit.qip** ファイルと **nios2_system_onchip_memory.hex** ファイル（このファイル名は、プロジェクトごとに異なります）を確認することができます。

5. nios2_system_onchip_memory.hex ファイルをテキスト・エディタで開くと、 Hex フォーマットでの記述となりますが、ソフトウェア・コードがデータとして反映されていることが確認できます（下図は一例となります）。

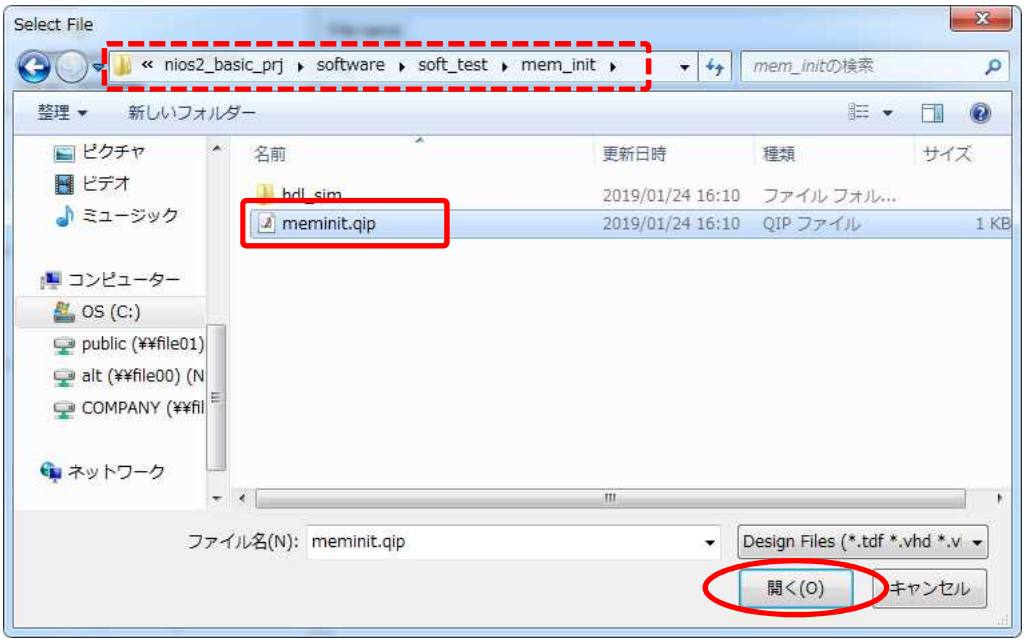


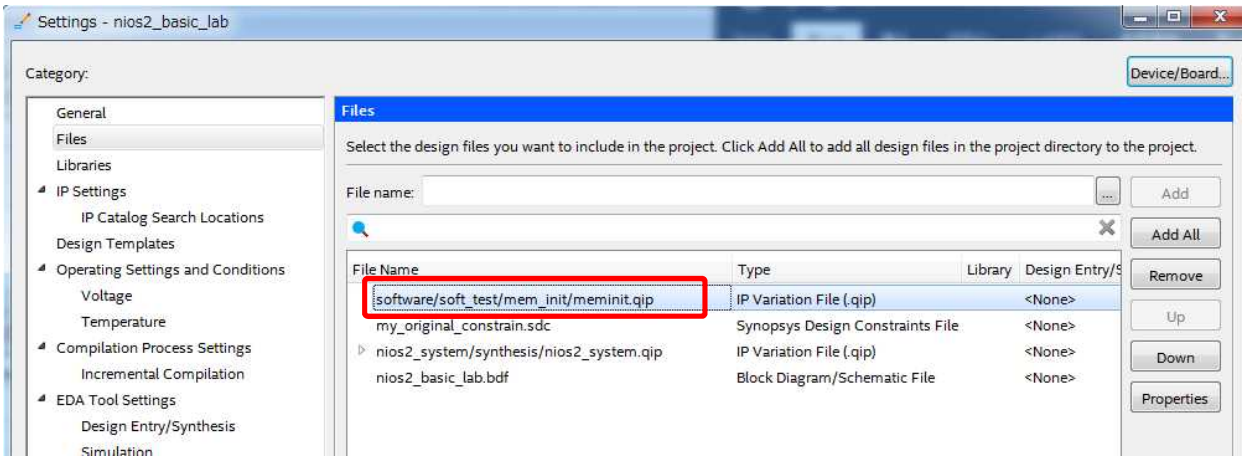
9-2. Quartus® Prime での設定とコンパイルから実行まで

1. Quartus® Prime で **Assignments** メニュー ⇒ **Settings** を選択します。

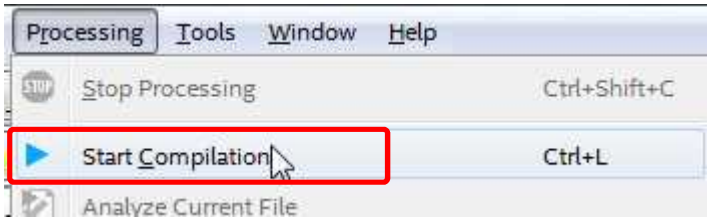


2. Settings ウインドウが起動します。画面左側の **Category** 欄から **Files** を選択して、右側の **File name** 欄に、前述の **meminit.qip** ファイルを指定します。 **File name** 枠内に追加されていることが確認できたら、右下の **[OK]** ボタンをクリックして Settings ウインドウを閉じます。

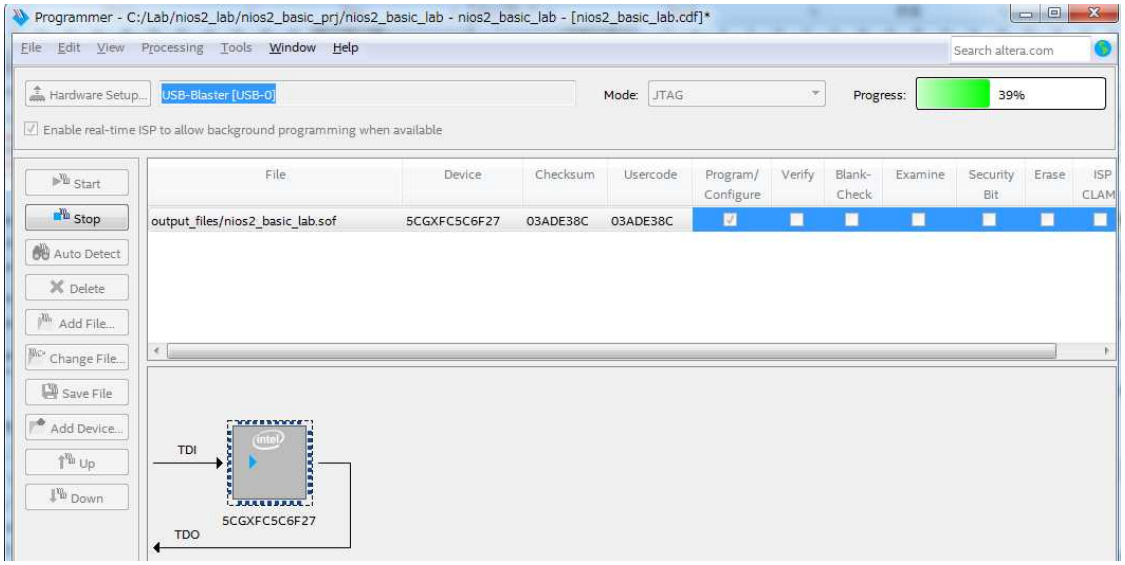




3. **Processing** メニュー ⇒ **Start Compilation** を選択して、フル・コンパイルを行います。



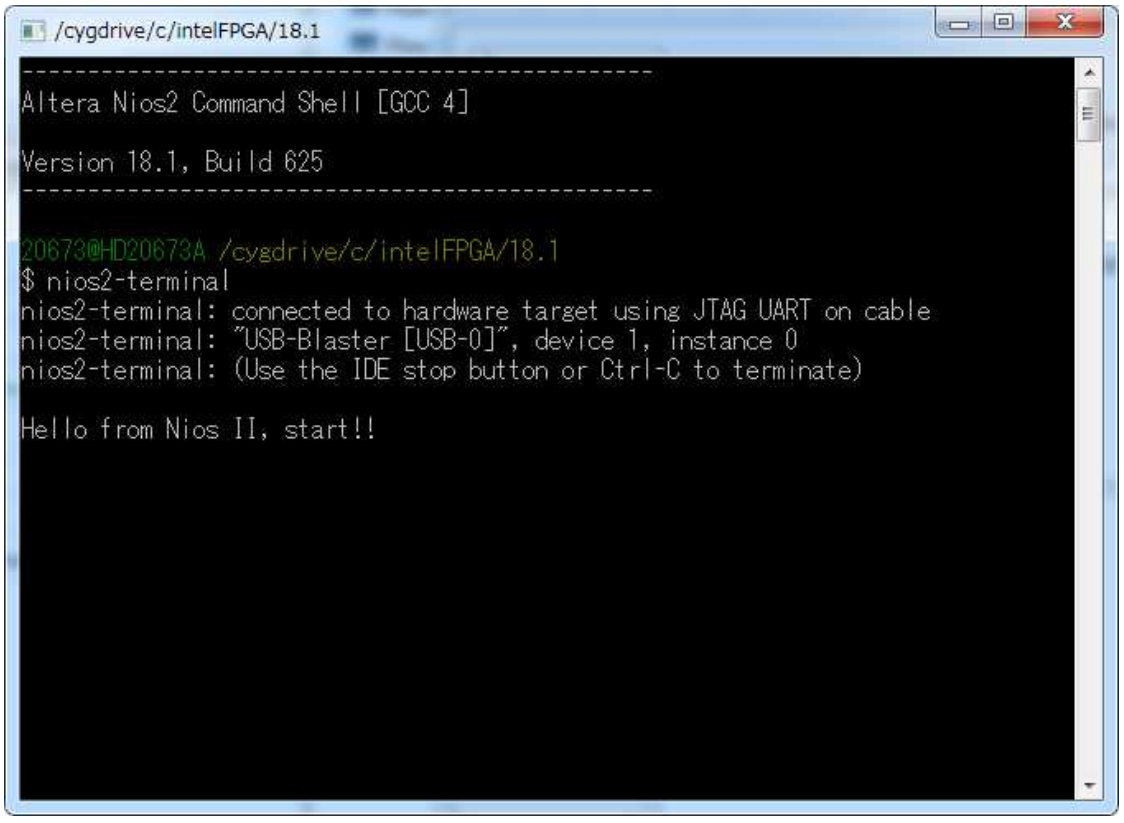
4. コンパイルが完了すると作成した HEX ファイルの内容を含んだ **SOF/POF** ファイルが生成されるので Quartus® Prime Programmer で **SOF** ファイルを書き込みます。



5. 正常であれば、 FPGA への書き込みが完了した時点で LED の点滅が確認できます。

6. また soft_test では、標準出力に JTAG_UART を使用しているため、 Nios® II Command Shell から標準出力への出力を確認してください。すべてのプログラム ⇒ **Intel FPGA 18.1.0.625 Standard Edition** ⇒ **Nios II EDS 18.1.0.625** ⇒ **Nios II Command Shell (Quartus Prime 18.1)** を選択すると、 Nios® II Command Shell が起動しますので、コンソールから **nios2-terminal** と入力します。

   「Hello from Nios II, start!!」のメッセージが表示されます。

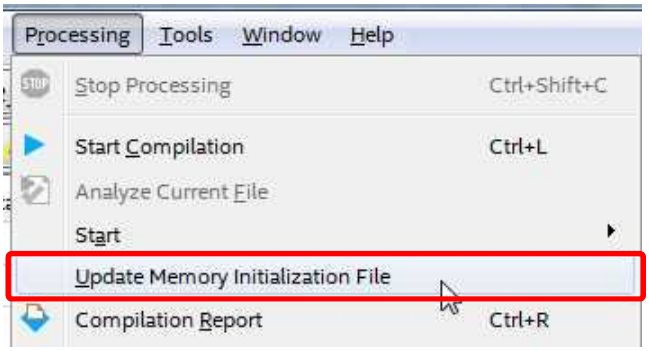


9-3. HEX ファイル更新時の Quartus® Prime プロジェクトへの反映方法

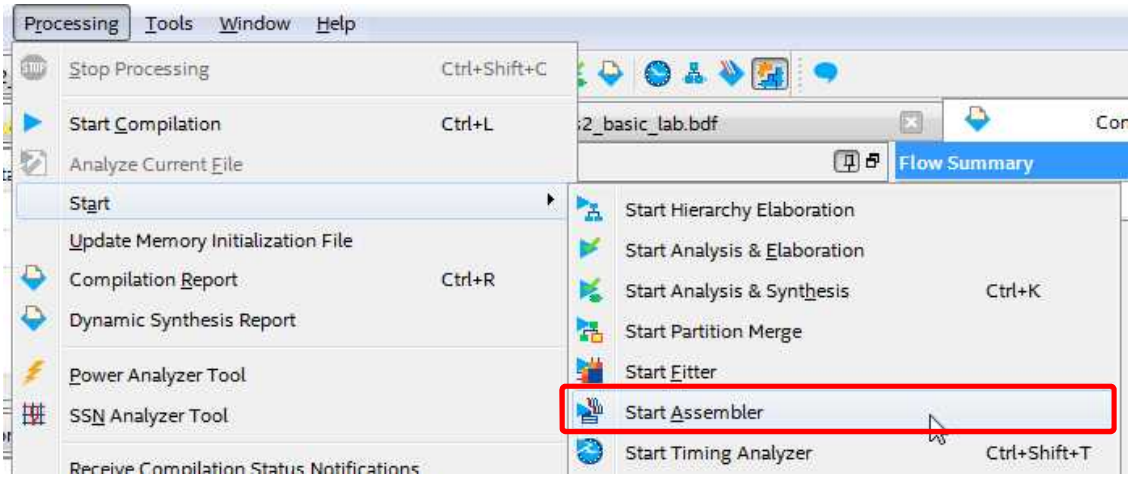
Nios® II の実行ファイルである ELF ファイルが変更された際には、**mem_init_generate** を再度実行することで HEX ファイルを更新しますが、そのたびに Quartus® Prime でフル・コンパイルするのは効率的ではありません。ここでは HEX ファイルに対応するオンチップ・メモリの内容のみ更新する方法を実行します。

1. Nios® II SBT で soft_test プロジェクトのプログラム（led_output.c）を編集して、LED の点滅速度や点滅パターンを変えてみましょう。編集したらファイルを保存し、soft_test フォルダを右クリック ⇒ **Build Project** を実行します。
2. soft_test フォルダを右クリック ⇒ **Make Targets** ⇒ **Build** を選択し、**mem_init_generate** を実行します。
3. Quartus® Prime の **Processing** メニューから **Update Memory Initialization File** を選択します。この処理で更新された HEX ファイルを解析してファイルに問題があるかどうかをチェックします。

   この時点では、HEX ファイルの内容は SOF/POF ファイルには反映されていません。



4. 次に、**Processing** メニューから **Start** ⇒ **Start Assembler** を選択します。この処理で SOF/POF ファイルに新しい HEX ファイルの内容が反映されます。フル・コンパイルに比べて、FPGA 内部のオンチップ・メモリのみ更新するので処理が短時間で済みます。



5. この処理が完了したら、SOF/POF ファイルのタイムスタンプが更新されていることを確認してください。また、改めて FPGA に書き込んで実機での動作を確認してください。

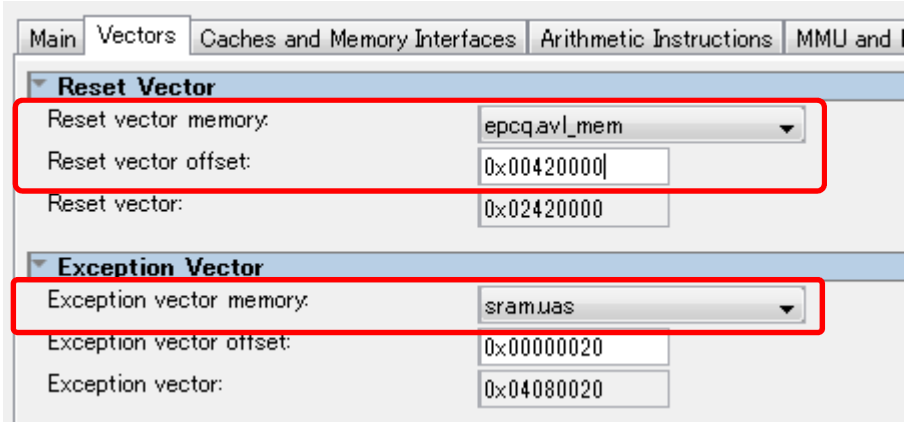
10. EPCQ からのブート（オプション）

前章ではオンチップ・メモリからのブート方法を実施しましたが、この章ではボードに実装されている EPCQ デバイスからのブートを実施します。合わせて、今までの内容では Nios® II のソフトウェアのワーク・メモリとしてオンチップ・メモリを使用しましたが、この章では SRAM デバイスをワーク・メモリとして設定し、動作を見ていきます。これにより、オンチップ・メモリの使用量を少なくすることが可能になります。 Platform Designer 上のオンチップ・メモリの設定でサイズの設定を小さくすることで、オンチップ・メモリの使用リソースが削減されますので、他のロジックでさらにオンチップ・メモリを有効に使用できるようになります。

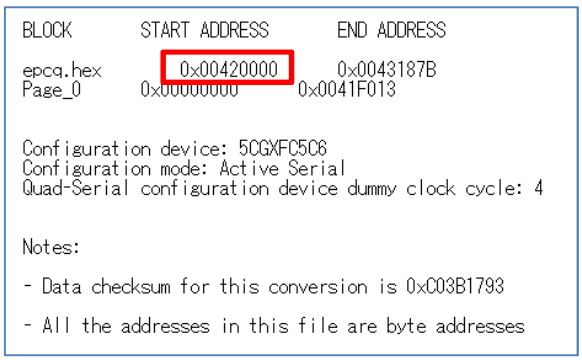
10-1.リセット・ベクタと例外ベクタの設定

使用する文字は、以下のルールに従ってください。

1. Platform Designer を起動し、 **nios2_cpu** をハイライト後、マウスで右クリックし **[Edit]** を選択します。 Nios® II プロセッサ・コアの GUI 画面が起動します。
2. **Vectors** タブの **Reset Vector** の項目のプルダウン・メニューより、 **epcq.avl_mem** を選択し、 **Reset vector offset** には、 **0x00420000** を設定します。
3. **Exception Vector** のプルダウン・メニューより **sram.uas** を選択します。



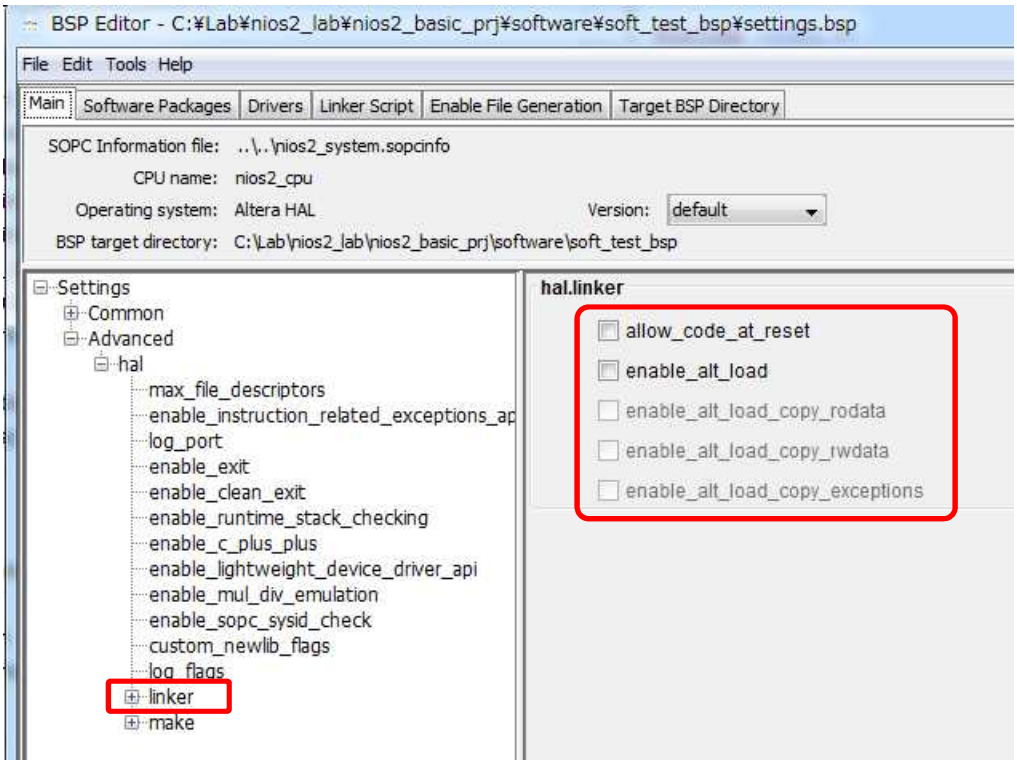
⚠ *0x00420000 というアドレス値は、後述の JIC ファイルの生成時に生成されるマップ・ファイル（*.map）の値から参照した値となります。 EPCS/EPCQ デバイスから FPGA のハードウェアデータ（SOF ファイル）をコンフィグレーションする場合には 0x00000000 番地からと決まっており、 Nios® II のソフトウェアデータ（ELF ファイル）は、そのあとの番地に置かれます。どこの番地に置かれるかはデバイスの規模により変わってくるので、本来は JIC ファイルを先に生成して、マップ・ファイルからソフトウェアデータのスタート・アドレスを確認しておく必要があります。*



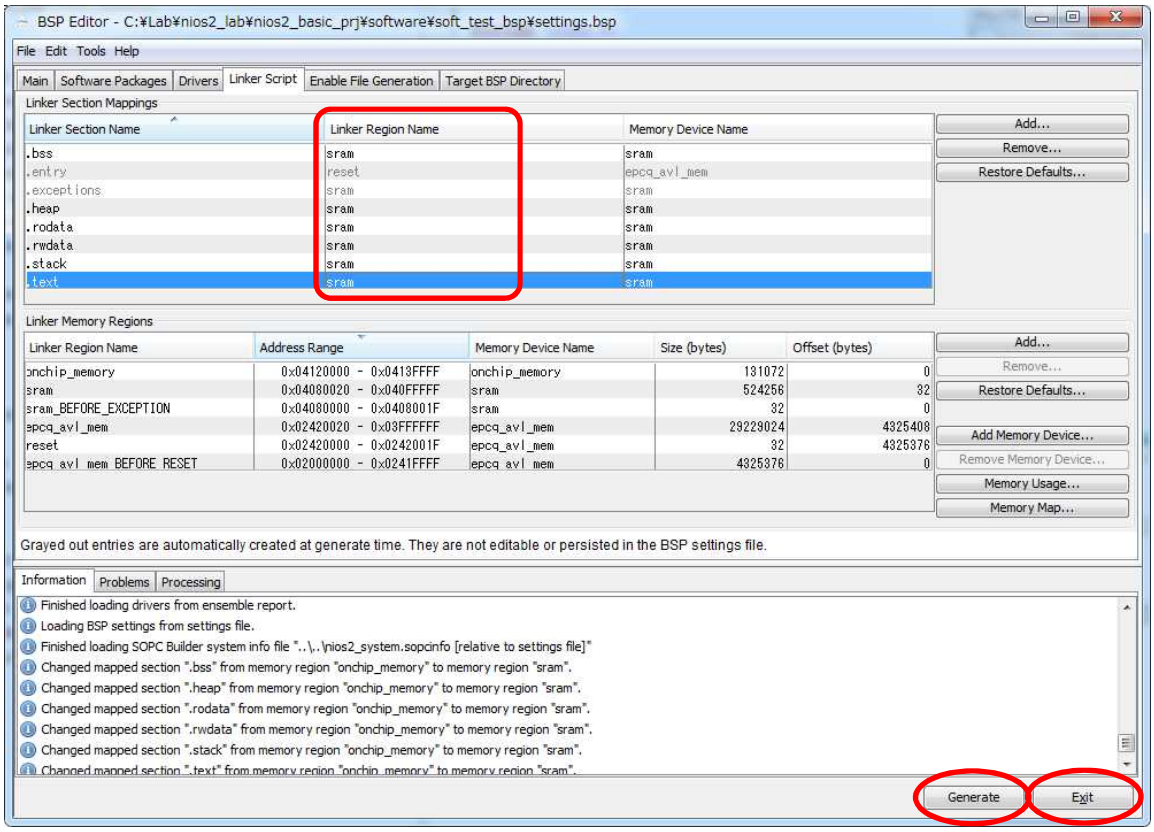
4. Nios® II プロセッサ・コアの画面を **[Finish]** で閉じ、 **File** メニュー ⇒ **Save** より変更を保存します。
5. Platform Designer のメイン・ウインドウから **Generate** メニュー ⇒ **Generate HDL** を選択します。
6. Generation ウインドウが表示されるので、 **[Generate]** ボタンをクリックしてシステムを生成します。
7. Platform Designer の生成が終わったら、 Quartus® Prime の **Processing** メニュー ⇒ **Start Compilation** を選択し、ハードウェア・デザインのコンパイルを実行します。ここでは、コンパイルの完了を待たずに、次にステップに進んでください。

10-2.ソフトウェア・プロジェクトのビルド

1. ハードウェアを変更したので、ソフトウェア・プロジェクトをクリーンします。 Nios® II SBT にて **soft_test** フォルダを右クリックして、 **Nios II** ⇒ **Clean Project** を選択します。
2. 次に、 Nios® II SBT にて **soft_test_bsp** フォルダを右クリックして、 **Nios II** ⇒ **BSP Editor** を選択します。 BSP Editor が起動します。
3. **Main** タブの **Setting** ⇒ **Advanced** ⇒ **hal** ⇒ **linker** を表示します。図のように全てのチェックを外してください。



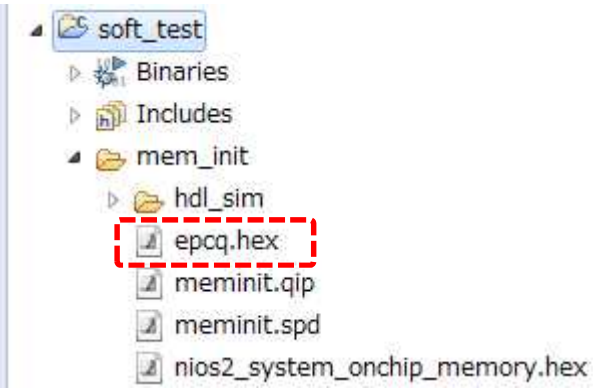
4. **Linker Script** タブを選択して、各リンカ・セクション（.bss, .text 等）の設定を下図のように **entry、exceptions** 以外を全て **sram** に設定したら、 **[Generate]** ボタンをクリックし、 **[Exit]** ボタンで画面を閉じます。



5. **soft_test** を右クリック ⇒ **Build Project** をクリックしソフトウェアをビルドします。

6. **soft_test** を右クリックして、プルダウン・メニューから **Make Targets** ⇒ **Build** を選択します。 Make Targets 画面が起動したら、 **mem_init_generate** をハイライトして、 **[Build]** ボタンをクリックします。

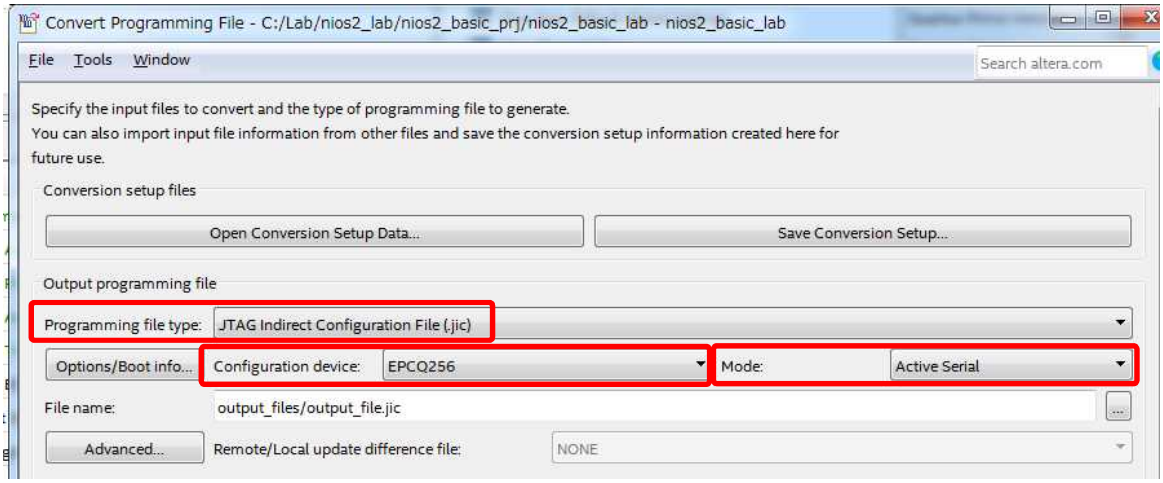
7. この処理で **mem_init** フォルダ内には、新たに **epcq.hex** というファイルが生成されていることが確認できます。



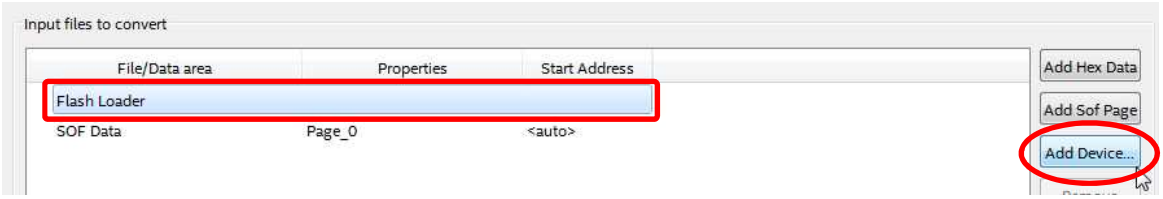
10-3.Convert Programming File を使用した書き込みファイルの生成

1. Quartus® Prime のコンパイルが完了していることを確認し、**File** メニュー ⇒ **Convert Programming File** を選択します。

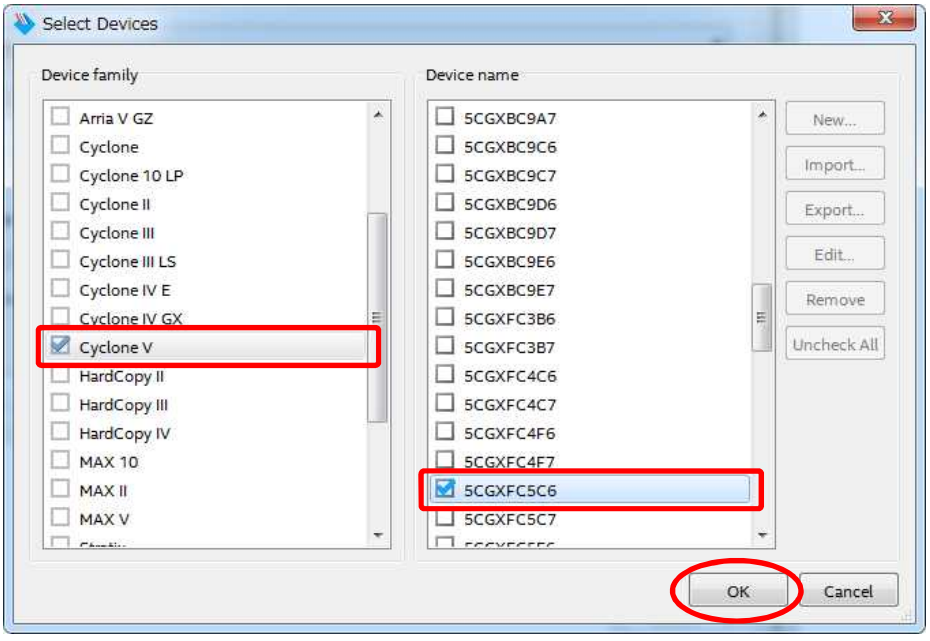
2. **Output programming file** の設定を行います。ここでは JIC 形式でファイルを生成するので、**Programming File Type** には **JTAG Indirect Configuration File (.jic)** を選択します。**Mode** は **Active Serial** を選択します。**Configuration device** には **EPCQ256** を選択します。



3. 次に **Input files to convert** の設定を行います。**Flash Loader** の行をハイライトした状態で、**[Add Device]** ボタンをクリックして、Selected Devices 画面を起動します。

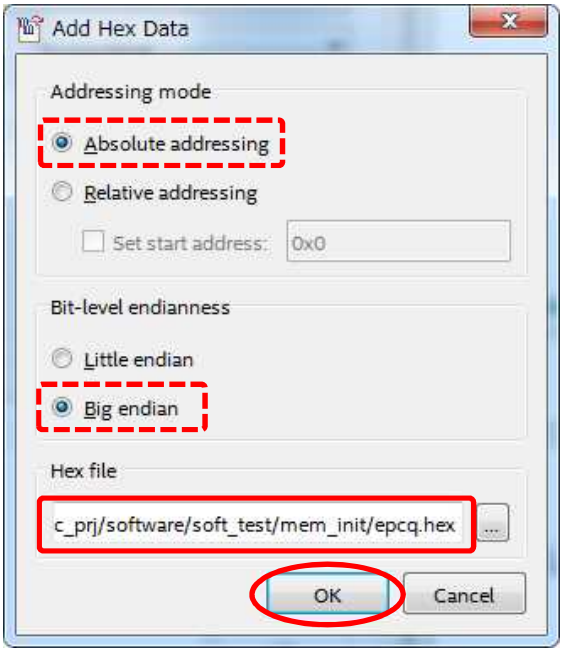


4. Selected Devices 画面で、**Device family** 欄から **Cyclone V** をハイライトし、**Device name** 欄の **5CGXFC5C6** を選択して、**[OK]** ボタンをクリックして設定を反映させます。

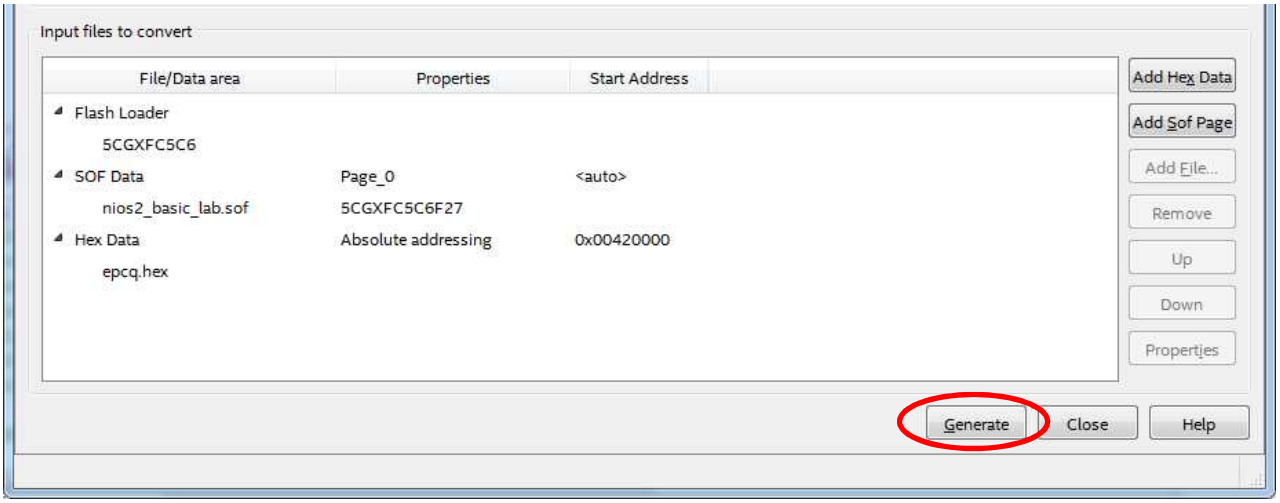


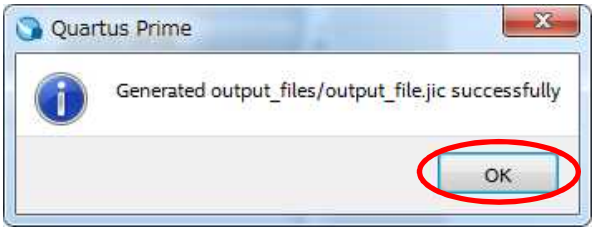
5. **SOF Data** の行をハイライトして、 **[Add File]** ボタンをクリックし、 SOF ファイルを選択します。

6. **[Add Hex Data]** ボタンをクリックして、 Nios® II SBT で作成した **epcq.hex** ファイルを指定します。 **Absolute addressing** と **Big endian** が選択されていることも確認し、 **[OK]** ボタンで設定を反映させます。



7. 最後に右下の **[Generate]** ボタンをクリックすると、 **output_file.jic** ファイルが生成されます。



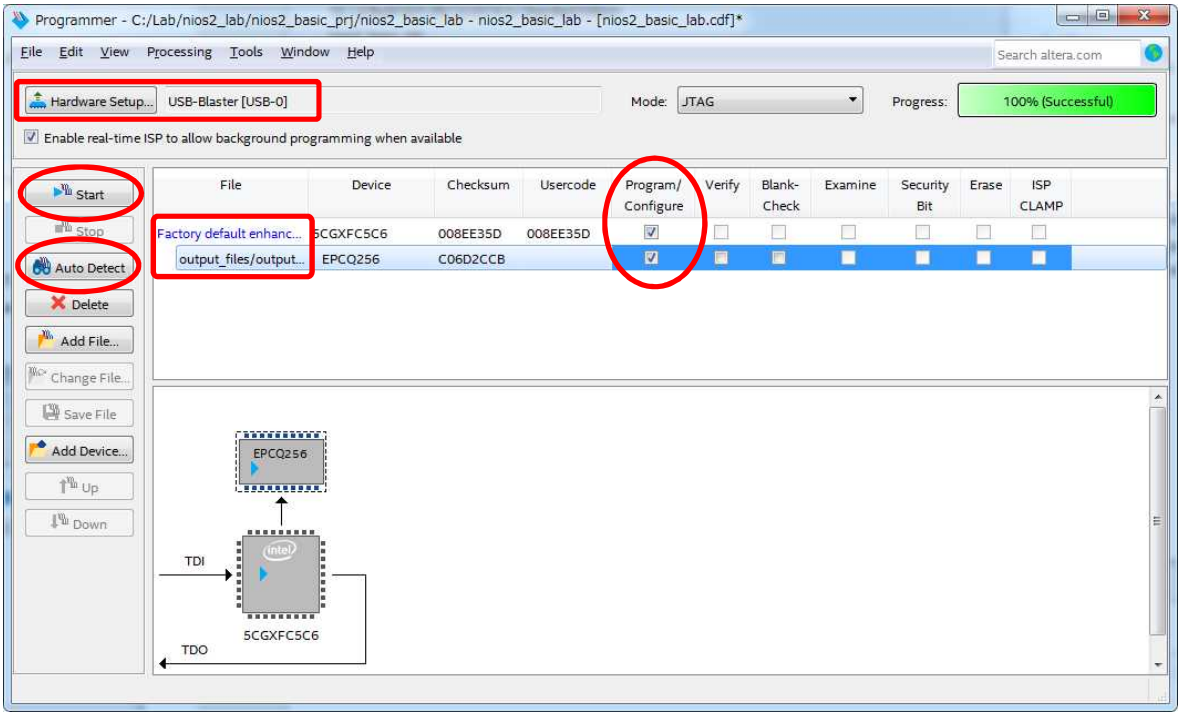


10-4.Quartus® Prime Programmer での JIC ファイルの書き込み

1. Quartus® Prime の **Tools** メニュー ⇒ **Programmer** を選択します。 Programmer ウインドウが表示されます。
2. **Hardware Setup** の項目に **USB-Blaster** が設定されていることを確認します。
3. **[Auto Detect]** ボタンを押して、ボード上のデバイスをサーチします。
4. **Select Device** が表示されたら、 **5CGXFC5C6** をチェックして **[OK]** ボタンをクリックします。
5. **File** 項目の **<none>** となっている箇所をダブル・クリックして、生成した **output_file.jic** ファイルを指定します。

   output_files/output_file.jic

6. **output_files/output_file.jic** 行の **Program / Configure** のチェック・ボックスにチェックを入れます。
   Serial Flash Loader (SFL) 用のファイルが自動的に設定されて、以下の画面のようになります。
7. **[Start]** ボタンをクリックして書き込みを開始します。



8. SOF ファイルの書き込みがはじめに行われ、次に HEX（ELF）ファイルの書き込みが行われます。処理には、 1～2 分程度かかります。
9. EPCQ デバイスへの書き込みが終わったら、ボードの電源を入れ直してプログラムが正常に動作することを LED の点滅にて確認してください。

   ボードの電源を入れた後、 EPCQ デバイスから AS モードで SOF をロードし FPGA が起動します。次に、FPGA 上の Nios® II がリセット・ベクタとして設定した EPCQ デバイスのアドレス位置からソフトウェア・プログラムをロードして動作を開始します。
10. また、 Nios® II Command Shell を使用して標準出力への出力を確認してください。すべてのプログラム ⇒ **Intel FPGA 18.1.0.625 Standard Edition** ⇒ **Nios II EDS 18.1.0.625** ⇒ **Nios II Command Shell (Quartus Prime 18.1)** と選択し、 Nios® II Command Shell を起動します。コンソールから **nios2-terminal** と入力します。

改版履歴

Revision	年月	概要
1	2017 年 7 月	初版
2	2018 年 4 月	ALTIMA テンプレートに変更
3	2019 年 1 月	ver18.1 対応
4	2019 年 7 月	Web 公開用に体裁修正（p26, p27 の図を微修正）

免責およびご利用上の注意

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

1. 本資料は非売品です。許可無く転売することや無断複製することを禁じます。
2. 本資料は予告なく変更することがあります。
3. 本資料の作成には万全を期していますが、万一ご不明な点や誤り、記載漏れなどお気づきの点がありましたら、本資料を入手されました下記代理店までご一報いただければ幸いです。

   株式会社マクニカ アルティマ カンパニー https://www.alt.macnica.co.jp/　　技術情報サイト アルティマ技術データベース http://www.altima.jp/members/
4. 本資料で取り扱っている回路、技術、プログラムに関して運用した結果の影響については、責任を負いかねますのであらかじめご了承ください。
5. 本資料は製品を利用する際の補助的な資料です。製品をご使用になる際は、各メーカ発行の英語版の資料もあわせてご利用ください。