Nios II Software Build Tool を使用したマルチコア・システムの実装

ver.14

2015 年 5 月 Rev.1

Nios II Software Build Tool を使用した
マルチコア・システムの実装

目次

1. はじめに

この資料は、複数の Nios® II を使用したマルチコア・システムにおいて、メモリやペリフェラルを共有して使用する方法や注意点、マルチコア・システムならではのデバッグの方法、Flash メモリへの書き込み方や注意点などを、検証を元に解説しています。

2. 適用条件

本資料では、Common Flash Interface (CFI) Flash ROM を使用したファスト・パッシブ・パラレル・コンフィグレーション(FPP)を使用し、ハードウェア・イメージと各ソフトウェア・イメージを 1 つの Flash ROM に格納、起動させ、ソフトウェアの実行は外部の SSRAM を共有で使用して動作させます。

■ 対応バージョン

－ Quartus® II 開発ソフトウェア v14.1

－ Nios II Software Build Tools (Nios II SBT) v14.1

■ 検証ハードウェア

－ Cyclone V E FPGA Development Board

- FPGA : Cyclone® V 5CEFA7F31I7ES
- Flash Memory : 64Mbyte/16bit (CFI)
- SSRAM : 2Mbyte/18bit

※ Flash Memory と SSRAM は共有バスを使用

※ SSRAM は、バス幅 16 ビットとして使用

3. システムの構成

3-1. 検証デザインの概略

本検証では、3 つの Nios II コアを使ったマルチコア・システムを使用します。それぞれの Nios II には 1 つの BUTTON からの入力と 1 つの LED への出力、インターバル・タイマを持ちます。また、全ての Nios II に共有のペリフェラルとして 1 つの BUTTON と LED を持ちます。

検証のためのソフトウェアでは、それぞれの Nios II はタイマからの割り込みによって LED を等間隔に点滅させます。そして BUTTON の押下で、その点滅間隔を徐々に延ばす処理を行い、全ての Nios II が確実に動作していることを確認します。また、共有 BUTTON の押下では全ての LED の点滅間隔を初期状態に戻す処理を行い、全ての Nios II が共有のペリフェラルからの入力も正しく処理できることを確認します。

その他に、2 つの Nios II の間に共有メモリを実装し、プロセッサ間通信が行えることも確認しています。

3-2. ブロック・ダイアグラム

本検証では、Qsys ブロックを図 3-2-1 に示す構成としています。なお、図の中では割り込みコントローラやデバッグ関連のモジュールについては省略しています。

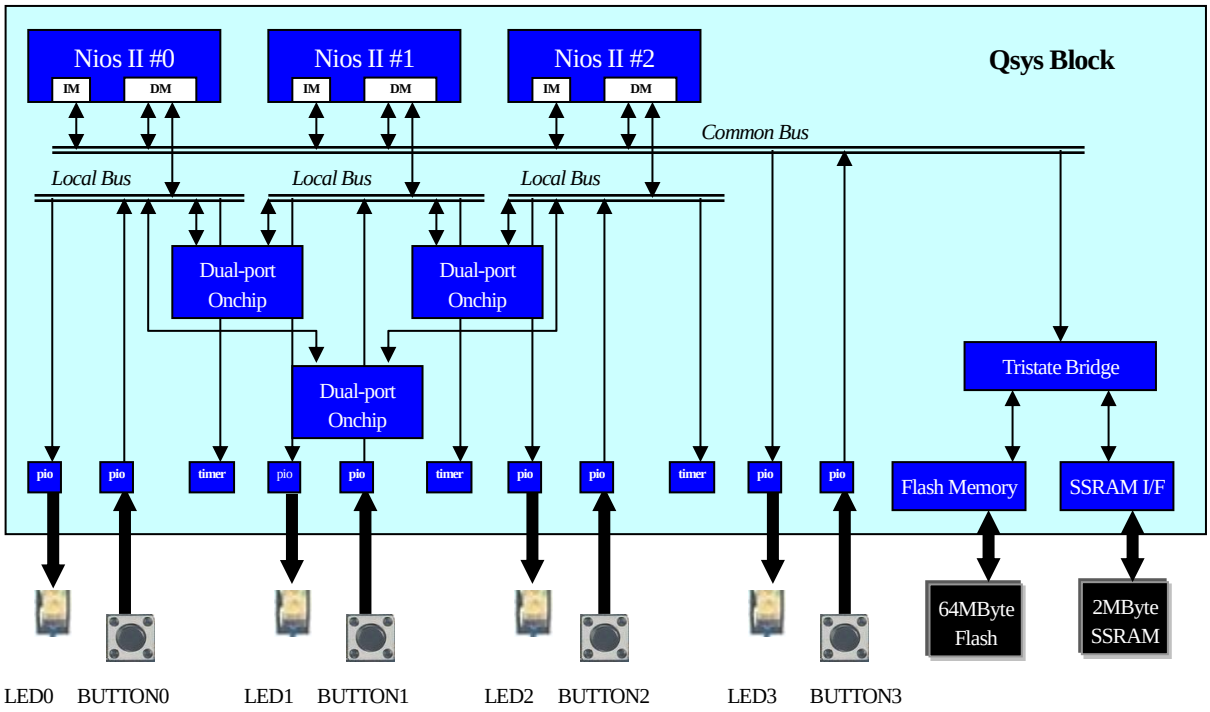

図 3-2-1

3-3. Qsys ブロック

図 3-3-1 に Qsys の接続を示します。

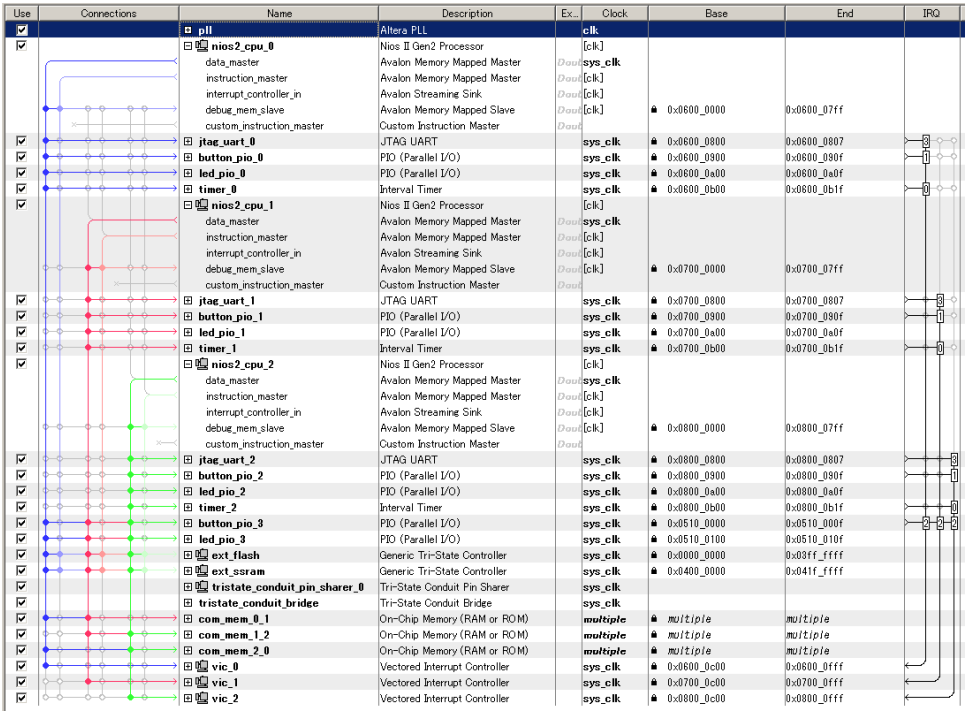

図 3-3-1

3-3-1. Nios II プロセッサの設定

下記に Nios II の設定を示します。基本的に 3 つの Nios II は同一の設定としますが、表 3-3-1-1 のとおり、Reset Vector Offset および Exception Vector Offset が異なることに注意してください。

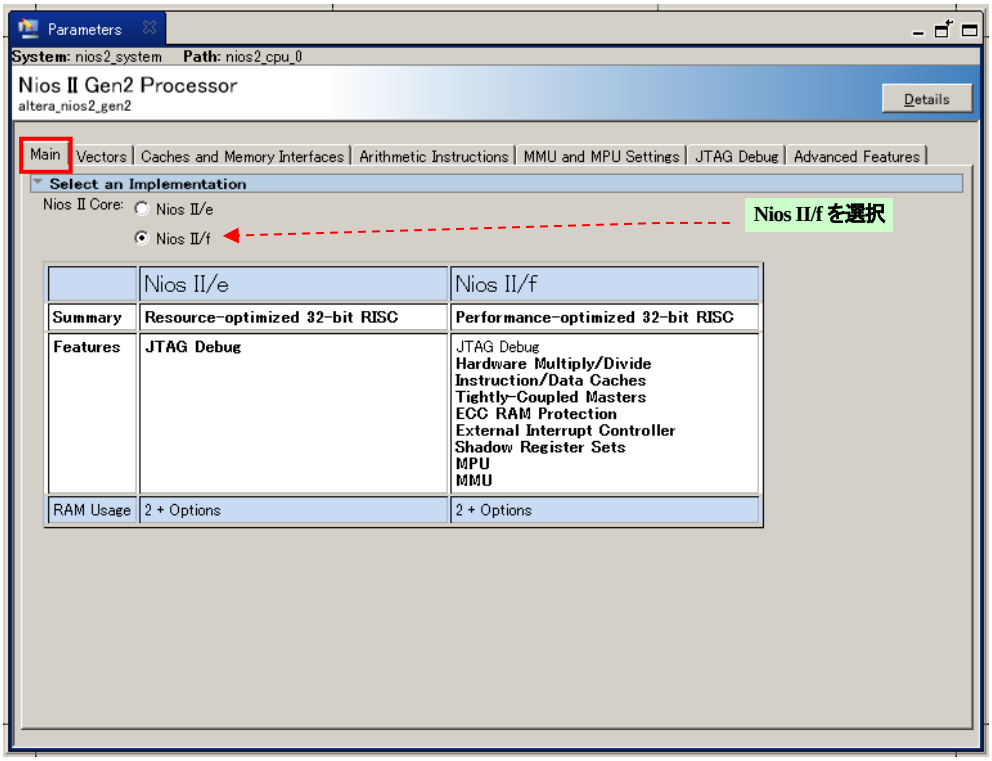

図 3-3-1-1

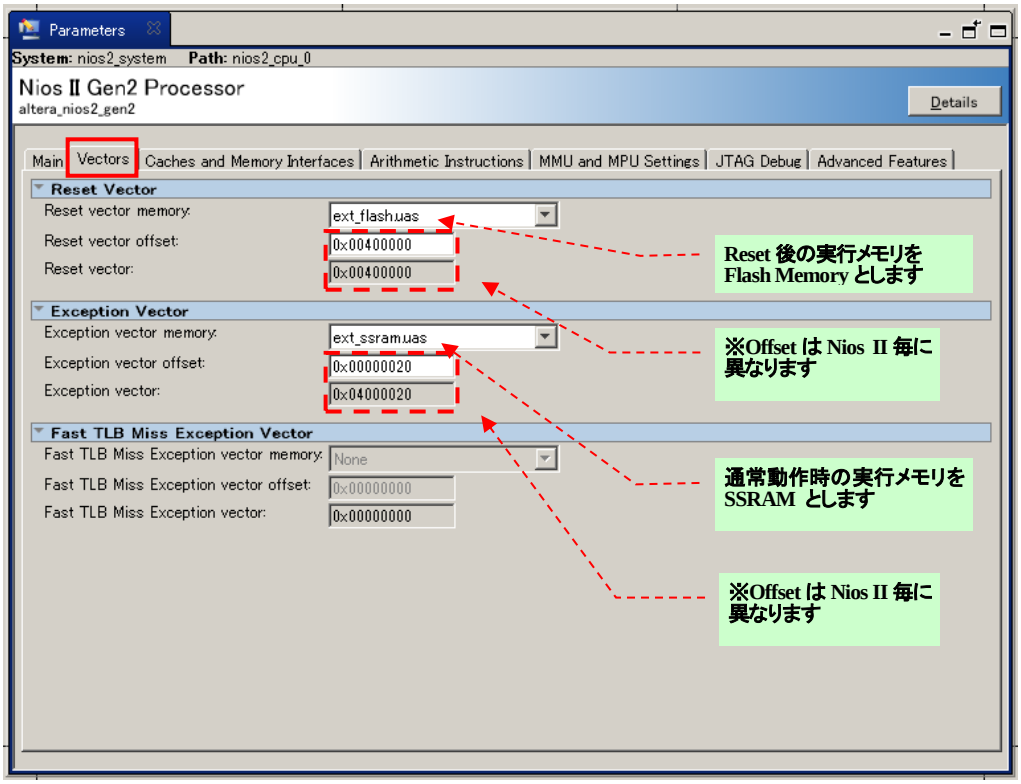

図 3-3-1-2

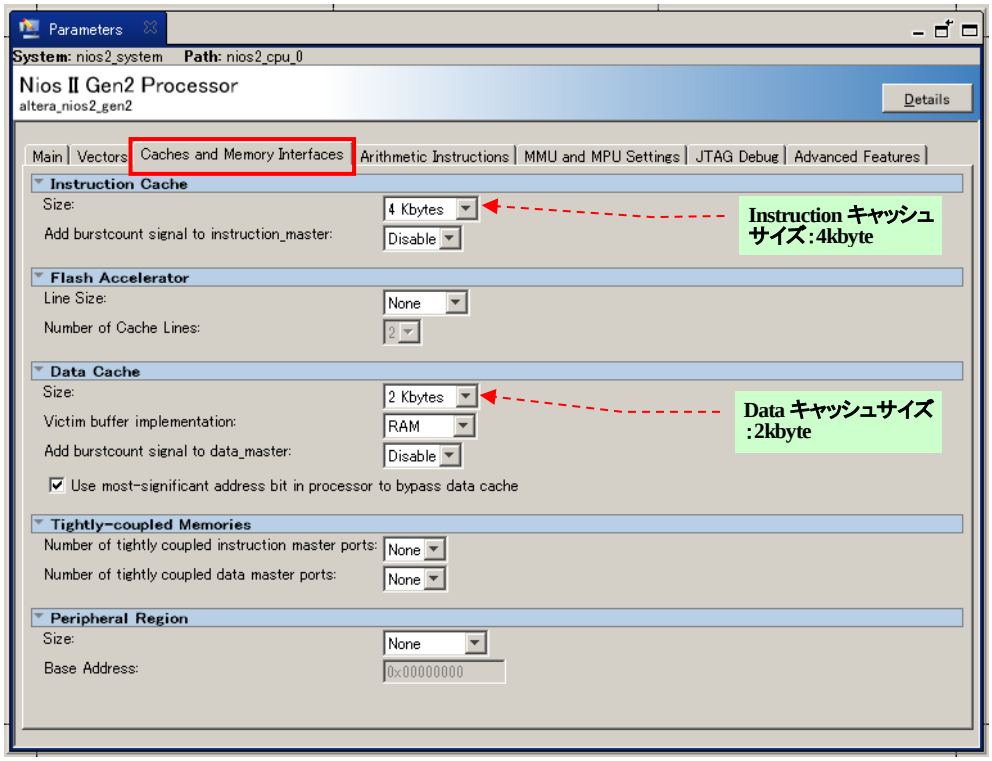

図 3-3-1-3

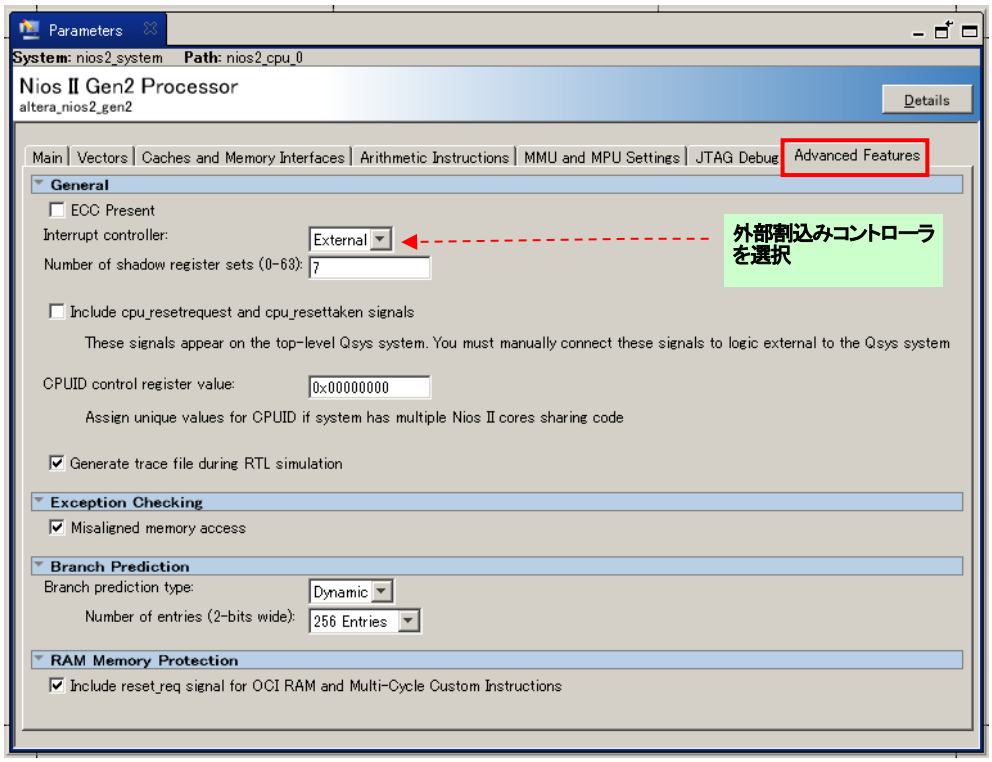

図 3-3-1-4

表 3-3-1-1

	Reset Vector Offset	ROM Size	Exception Vector Offset	RAM Size
Nios II #0	0x400000	384Kbyte	0x00020	384Kbyte
Nios II #1	0x460000	384Kbyte	0x60020	384Kbyte
Nios II #2	0x4C0000	384Kbyte	0xC0020	384Kbyte

3-3-2. BUTTON PIO の設定

下記に BUTTON PIO の設定を示します。BUTTON0～BUTTON3 の設定は同一とします。

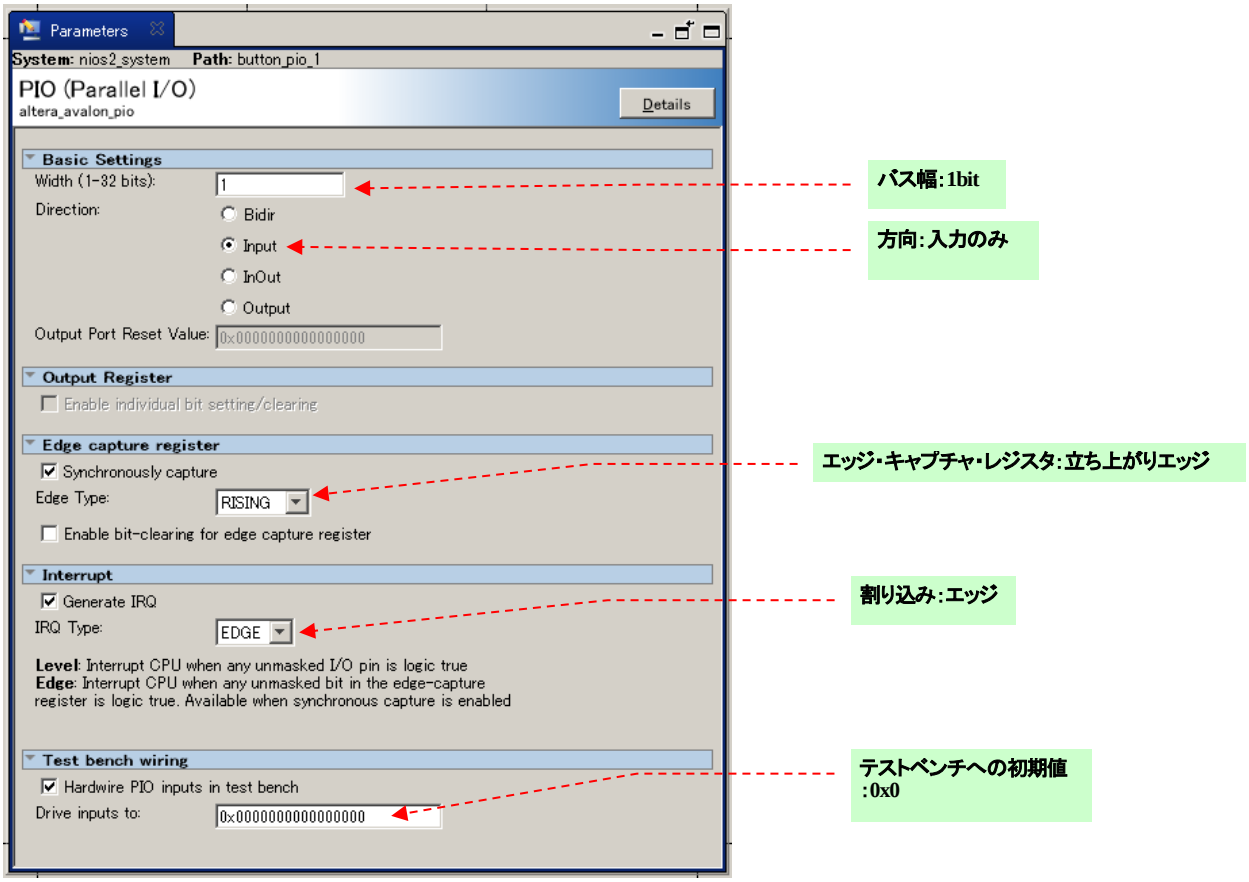

図 3-3-2

3-3-3. LED PIO の設定

下記に LED PIO の設定を示します。LED0～LED3 の設定は同一とします。

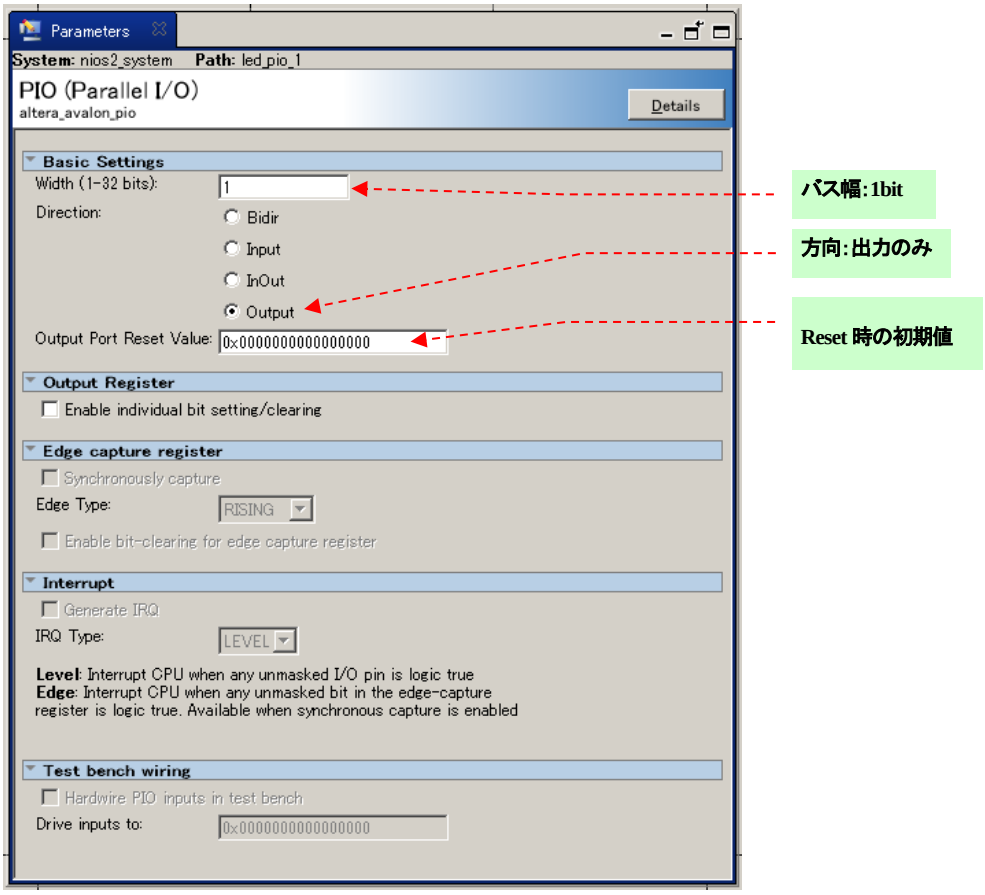

図 3-3-3

3-3-4. Interval Timer の設定

下記に Interval Timer の設定を示します。全ての Interval Timer の設定は同一とします。

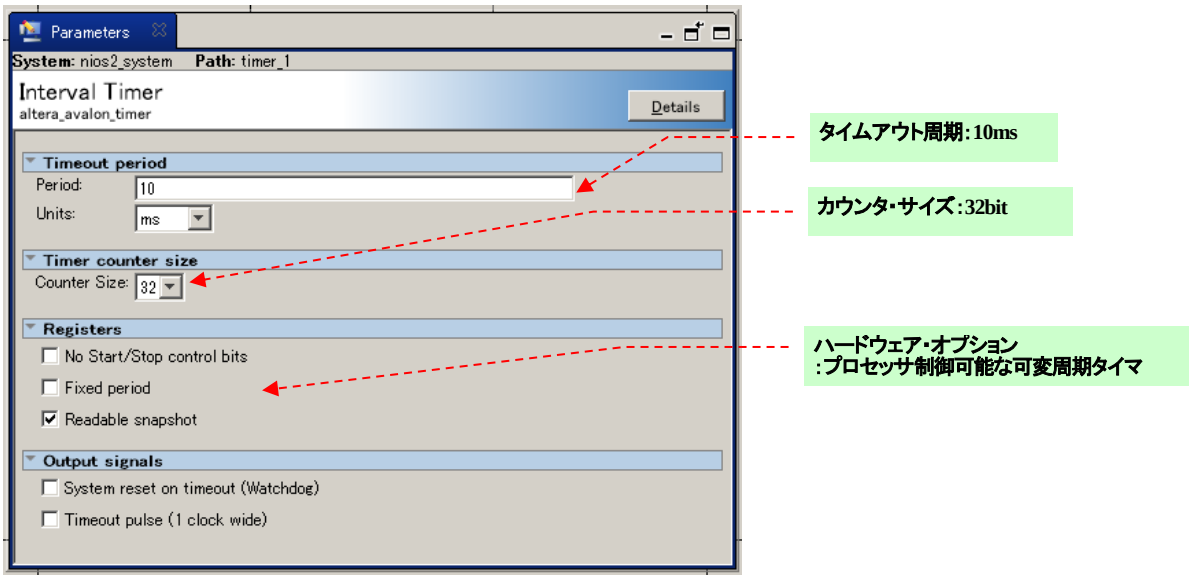

図 3-3-4

3-3-5. Dual-port Onchip RAM の設定

下記に On-Chip Memory の設定を示します。全ての On-Chip Memory の設定は同一とします。

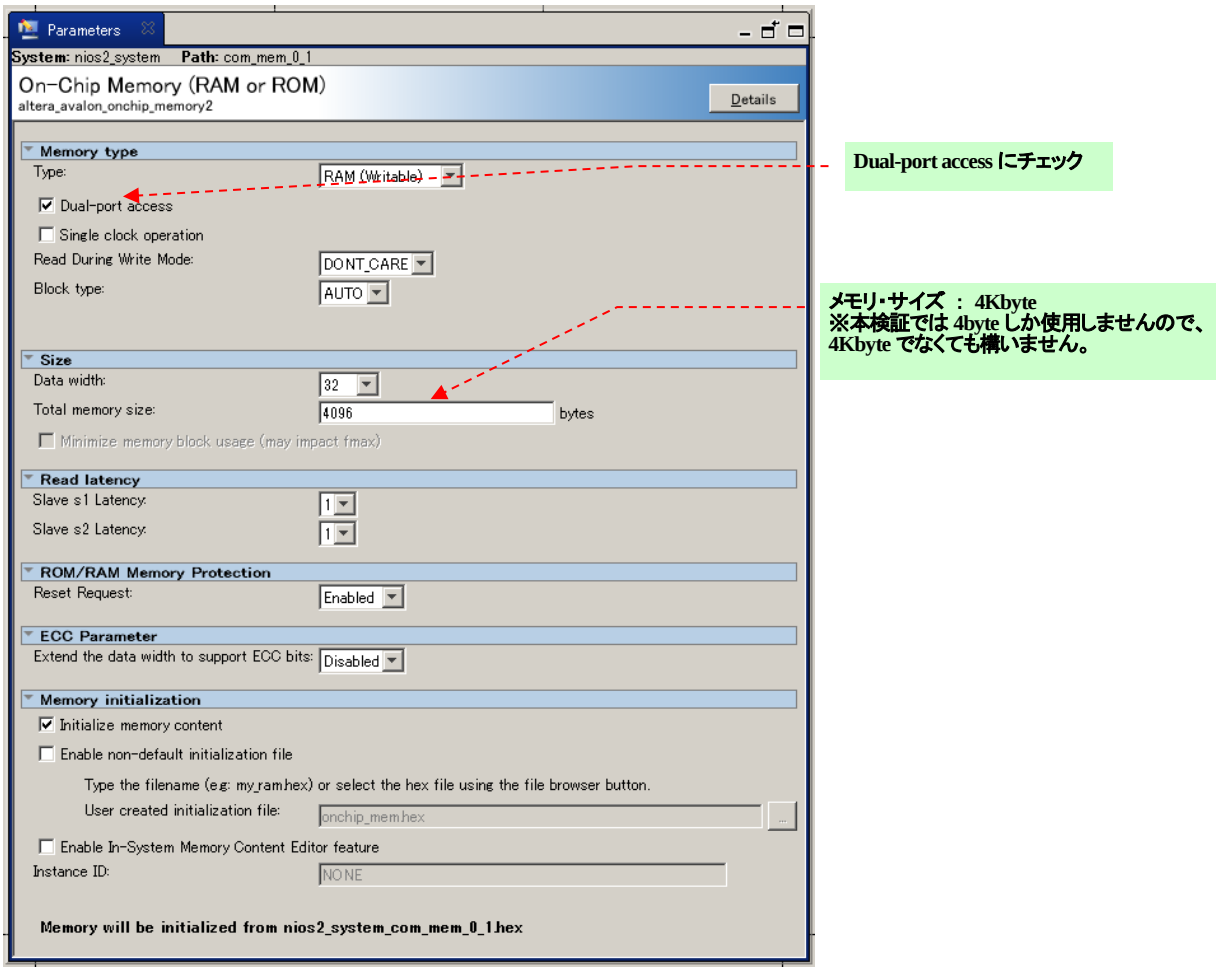

図 3-3-5

3-3-6. Vectored Interrupt Controller の設定

下記に Vectored Interrupt Controller の設定を示します。全ての Vectored Interrupt Controller の設定は同一とします。

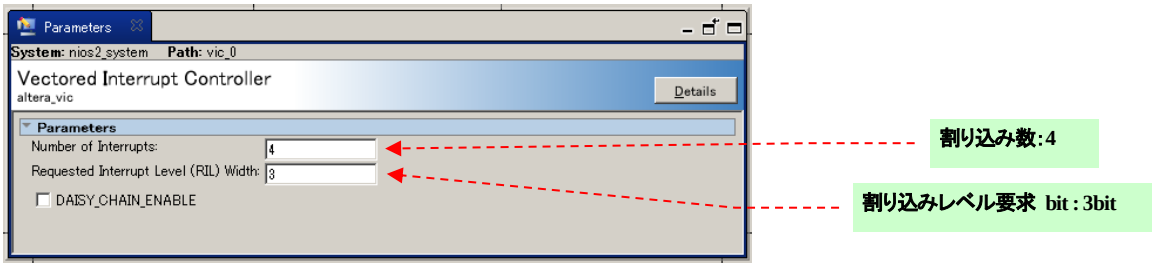

図 3-3-6

3-3-7. Flash Memory Interface (CFI) の設定

Flash Memory Interface の設定については、付属のデザインを確認ください。また、詳細については、担当する代理店の技術情報サイトにおいて、以下の資料をご参照ください。

「Qsys におけるオフチップ・メモリ・インタフェースの接続方法」

3-3-8. SSRAM の設定

SSRAM Interface の設定については、付属のデザインを確認ください。また、詳細については、担当する代理店の技術情報サイトにおいて、以下の資料をご参照ください。

「Qsys におけるオフチップ・メモリ・インタフェースの接続方法」

3-4. システム・メモリ・マップ

図 3-4-1 にシステム全体のメモリ・マップを示します。

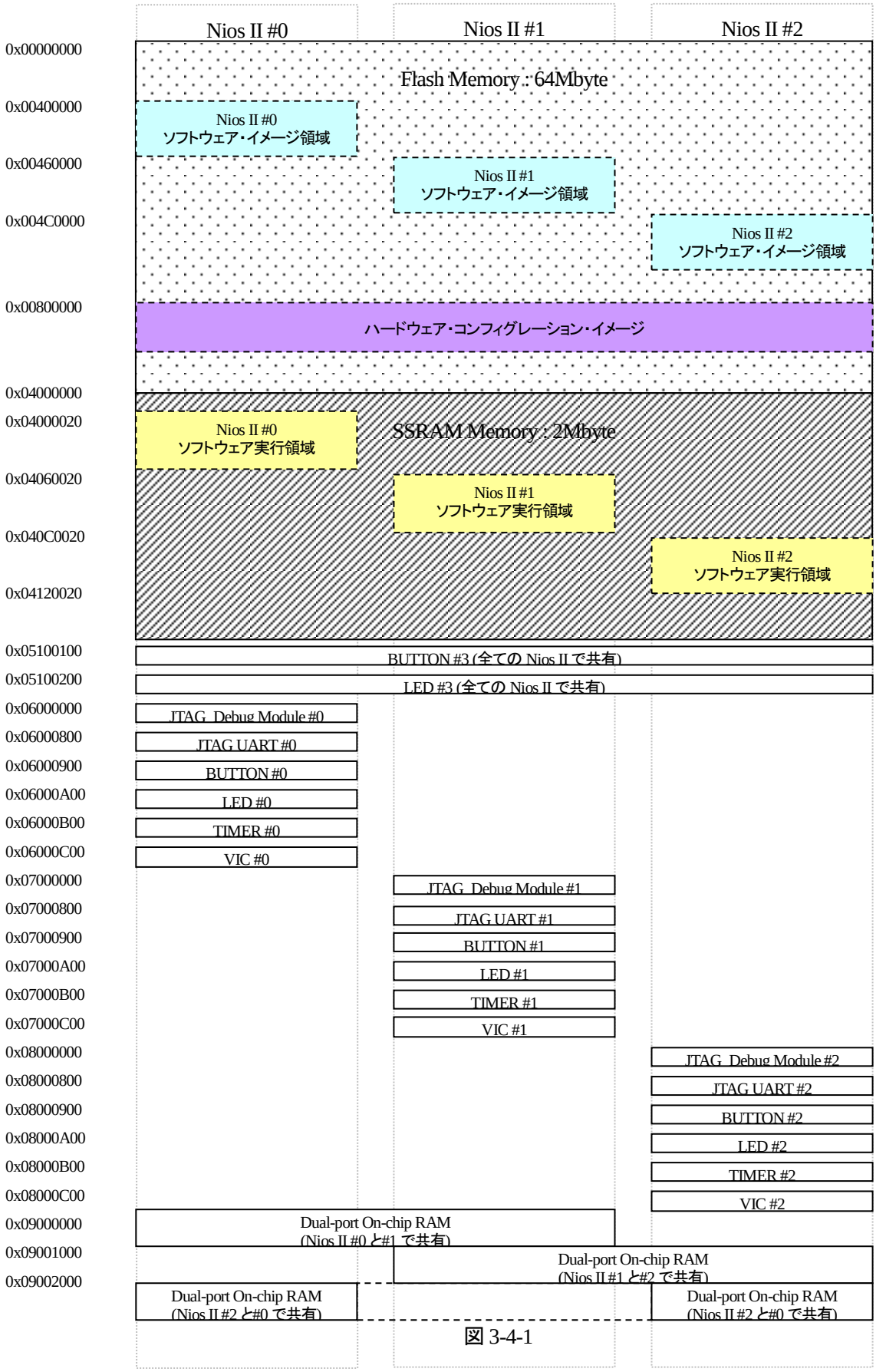

図 3-4-1

4. ソフトウェアの構成

4-1. 検証ソフトウェアの概略

本検証のソフトウェアは、main()関数の他にインターバル・タイマ割り込み関数、専用 BUTTON 割り込み関数、共有 BUTTON 割り込み関数の 4 つ関数で構成されています。main()関数では各ペリフェラルの設定と割り込み関数の登録を行っています。また、共有 On-chip RAM を使ったプロセッサ間でのデータの受け渡しも行っています。専用 BUTTON 割り込み関数では LED の点滅間隔を決定する変数のインクリメント処理をインターバル・タイマ割り込み関数では点滅間隔を決定する変数に従い LED の点滅処理を、共有 BUTTON 割り込み関数では LED の点滅間隔を決定する変数を初期化しています。

```
unsigned long   src __attribute__ ((section (".com_mem_xxx"))) = 0;
unsigned long   dst __attribute__ ((section (".com_mem_yyy"))) = 0;

short nCount = 0;
short nLed = 0;
short nTime = 1;

int main()
{
    // 共有メモリの初期化
    dst = 0;
    // 10msタイマ割り込みISRの登録
    alt_ic_isr_register(TIMER_x_IRQ_INTERRUPT_CONTROLLER_ID,TIMER_x_IRQ, timer_isr, 0, 0);
    // 10msタイマの起動
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TIMER_x_BASE, (ALTERA_AVALON_TIMER_CONTROL_ITO_MSK
                                                  | ALTERA_AVALON_TIMER_CONTROL_CONT_MSK
                                                  | ALTERA_AVALON_TIMER_CONTROL_START_MSK));

    // 専用ボタンISRの登録
    alt_ic_isr_register(BUTTON_x_IRQ_INTERRUPT_CONTROLLER_ID, BUTTON_x_IRQ, button_isr, 0, 0);
    // 専用ボタンの割り込みを許可
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_x_BASE, 0x1);
    // エッジ・キャプチャ・レジスタのリセット
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_x_BASE, 0x0);

    // 共有ボタンISRの登録
    alt_ic_isr_register(BUTTON_3_IRQ_INTERRUPT_CONTROLLER_ID, BUTTON_3_IRQ, common_isr, 0, 0);
    // 共有ボタンの割り込みを許可
    IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BUTTON_3_BASE, 0x1);
    // エッジ・キャプチャ・レジスタのリセット
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_3_BASE, 0x0);

    while(1)
    {
        // 共有メモリ<src>の値をインクリメントして<dst>に書き込む
        dst = src + 1;
    }

    return 0;
}
```

変数を共有メモリに割り当て※1

各 Nios II に接続されている
ペリフェラルに合わせて変更※2

<< main 関数 >>

```
void button_isr(void* context)
{
    // LED点滅間隔の変数をインクリメント
    nTime ++;
    if(nTime > 10)
    {
        nTime = 1;
    }
    // エッジ・キャプチャ・レジスタのリセット
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_x_BASE, 0x00);
}
```

<< 専用 BUTTON 割り込み関数 >>

```
void timer_isr(void* context)
{
    nCount ++;
    if(nCount > (nTime * 10))
    {
        // LEDを反転
        nLed ^= 0x01;
        // LED出力
        IOWR_ALTERA_AVALON_PIO_DATA(LED_x_BASE, nLed);

        nCount = 0;
    }
    // タイマ割り込みのリセット
    IOWR_ALTERA_AVALON_TIMER_STATUS(TIMER_x_BASE, 0x00);
    return;
}
```

<< インターバル・タイマ割り込み関数 >>

```
void common_isr(void* context)
{
    // LED点滅間隔の変数を初期化
    nTime = 1;
    // LEDの状態を初期化
    nLed = 0;
    // エッジ・キャプチャ・レジスタのリセット
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BUTTON_3_BASE, 0x0);
}
```

<< 共有 BUTTON 割り込み関数 >>

※ 1. ソースコード中の ***com_mem_xxx*** および ***com_mem_yyy***、IRQ や BASE で定義されている ***x*** は、表 4-1 のとおり、Nios II ごとに変更する必要があります。

表 4-1

	com_mem_xxx	***com_mem_yyy***	***x***
Nios II #0	com_mem_2_0	com_mem_0_1	0
Nios II #1	com_mem_0_1	com_mem_1_2	1
Nios II #2	com_mem_1_2	com_mem_2_0	2

※ 2. 共有メモリが正しくインクリメント処理されていることを確認できるように、共有メモリの変数を LED に出力する処理を実装します。いずれかの Nios II の main()関数のループに下記のコードを追加してください。

```
    :
    :
while(1)
{
    // 共有メモリ<src>の値をインクリメントして<dst>に書き込む
    dst = src + 1;
    // 共有メモリ間のインクリメント処理が正しく動作していることをLEDの点滅で確認
    IOWR_ALTERA_AVALON_PIO_DATA(LED_3_BASE, dst >> 17);
}
```

<< プロセッサ間通信動作状況出力処理 >>

※ 3. ソフトウェア処理の詳細につきましては、別途、各ペリフェラルの資料などを参照ください。

4-2. ソフトウェア・プロジェクトの生成

① Nios II SBT を起動して Workspace フォルダを Quartus II プロジェクトのフォルダの直下に作成します。そして、Project Explorer の上にマウスを移動し右クリックし図 4-2-1 のように New ⇒ Nios II Application and BSP from Template を選択します。

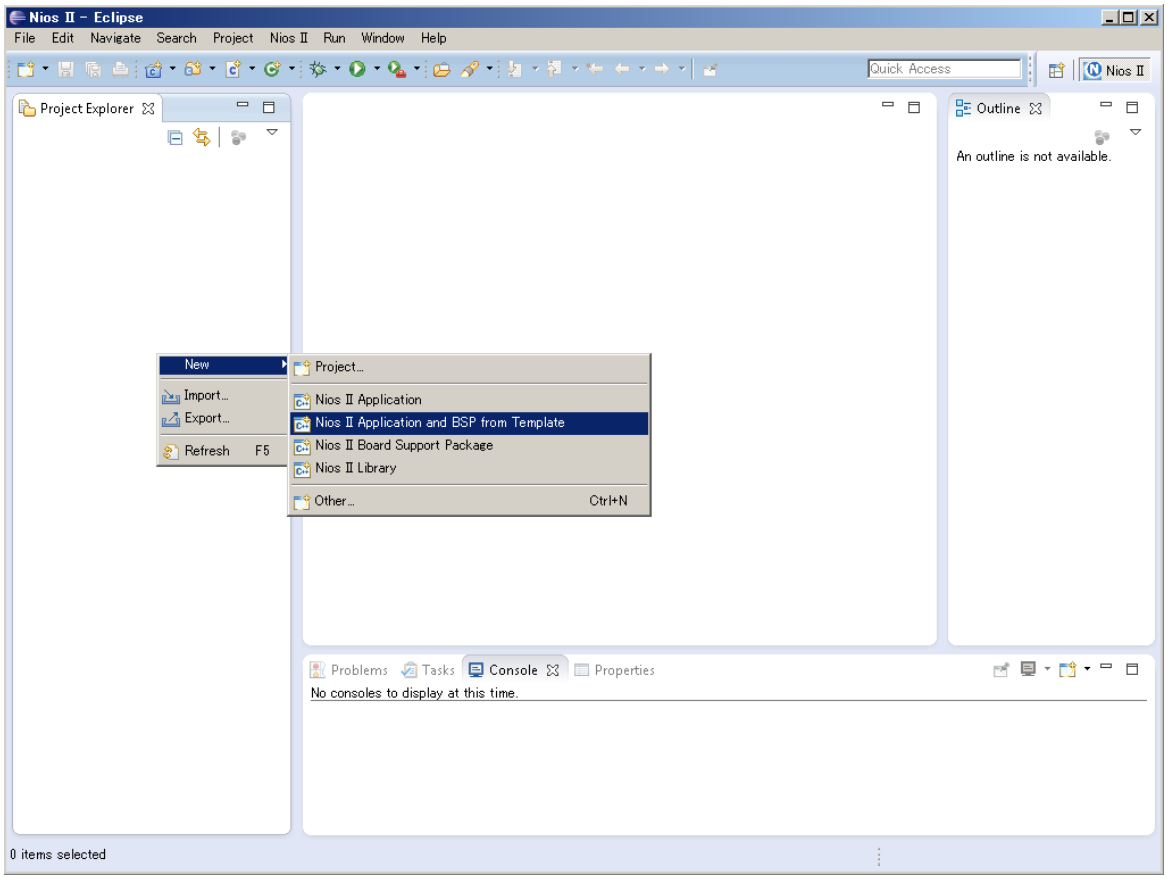

図 4-2-1

② 表示された Nios II Application and BSP from Template ウィンドウに図 4-2-2 のように設定し Finish をクリックします。

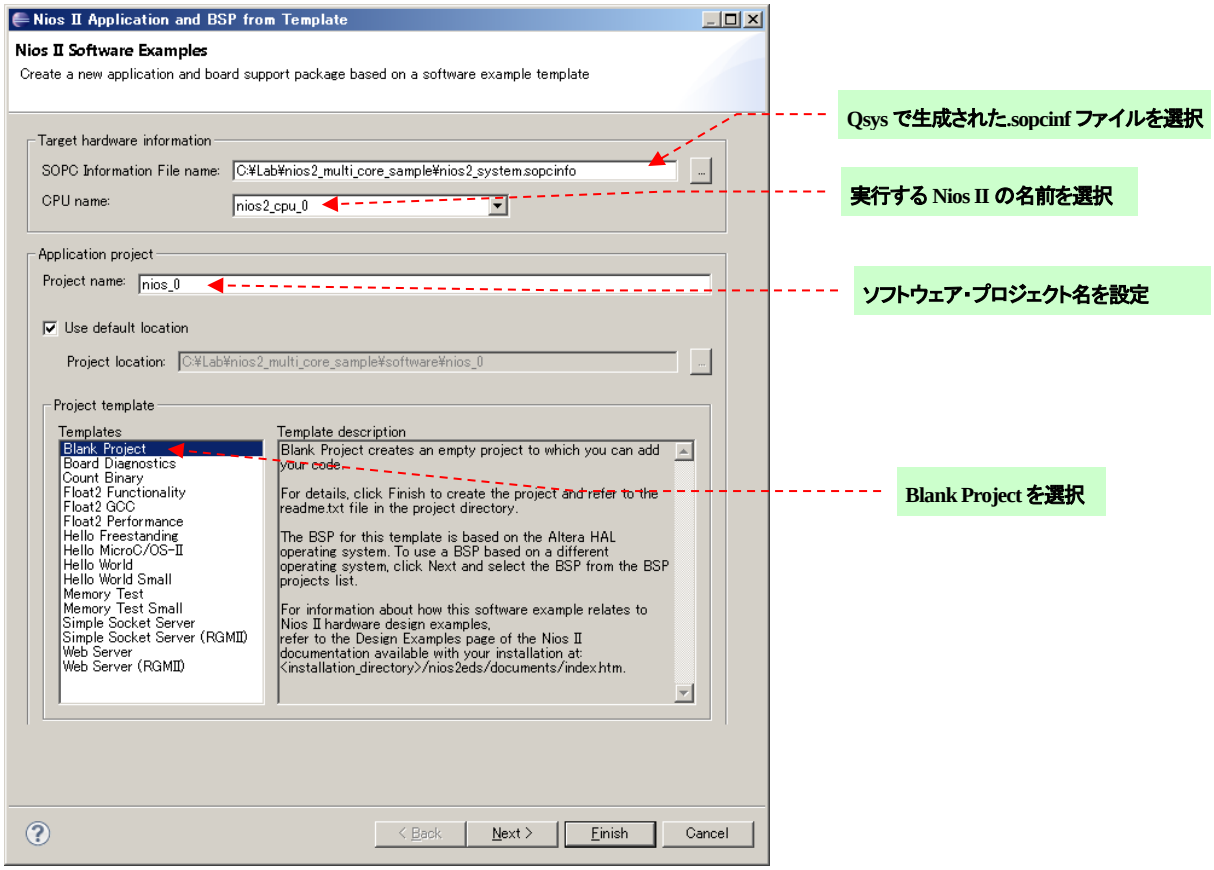

図 4-2-2

③ Project Explorer のプロジェクトを選択した状態で右クリックし、図 4-2-3-1 のように New ⇒ File を選択します。New File ウィンドウが表示されたら図 4-2-3-2 のようにファイル名を入力して Finish をクリックします。

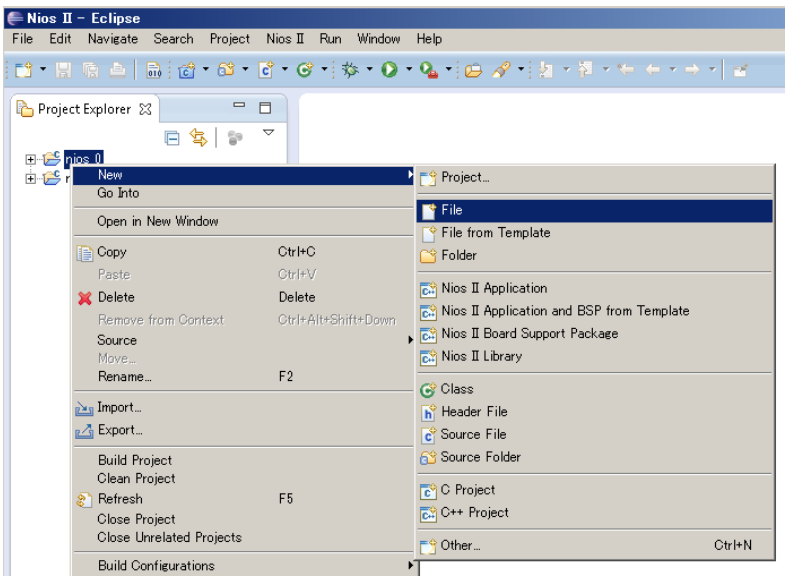

図 4-2-3-1

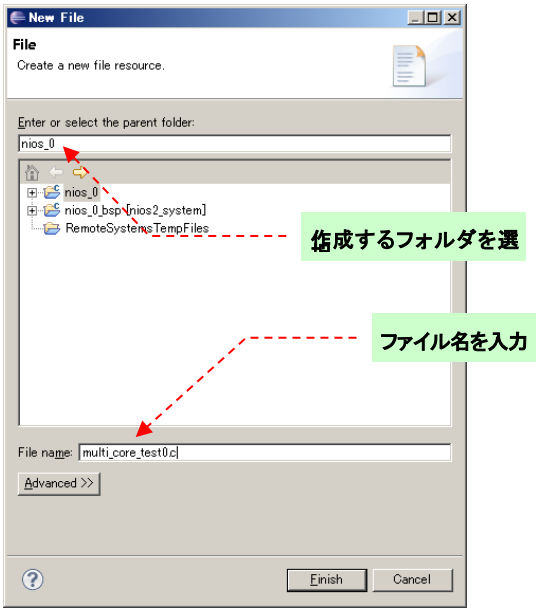

図 4-2-3-2

④ 作成したファイルが開くのでソースコードを入力します。前述のソースコードの他にも適時インクルード・ファイルやプロトタイプ宣言なども追加してください。

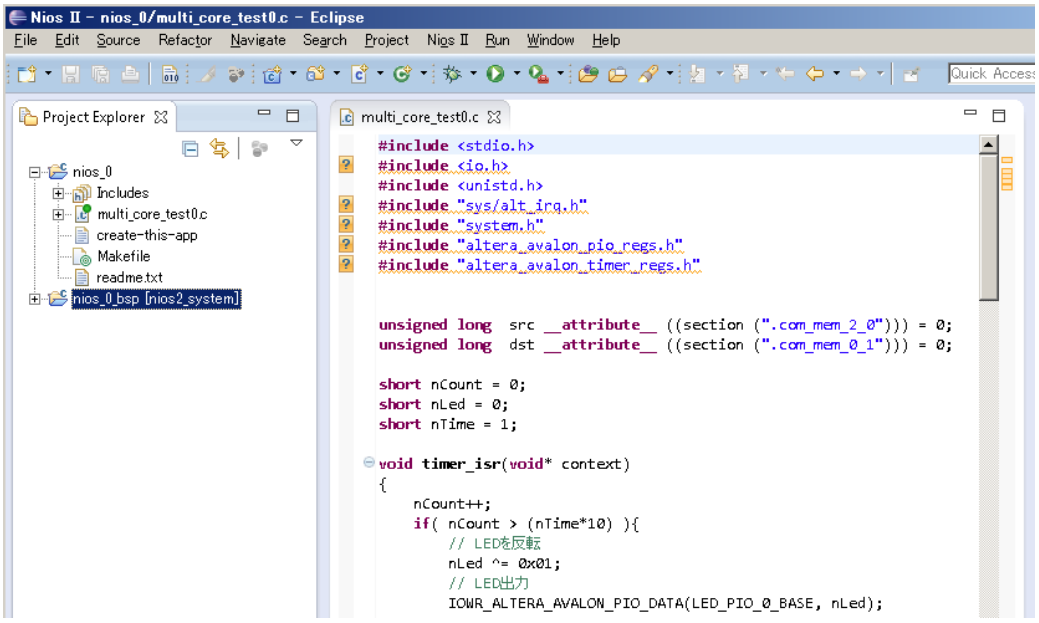

図 4-2-4

⑤ Project Explorer の BSP プロジェクト（nios_0_bsp の方）を選択した状態で右クリックし、Nios II ⇒ BSP Editor... を選択します。

⑥ Nios II BSP Editor ウィンドウが表示されたら図 4-2-5 のように Main タブの sys_clk_timer: を none に設定します。

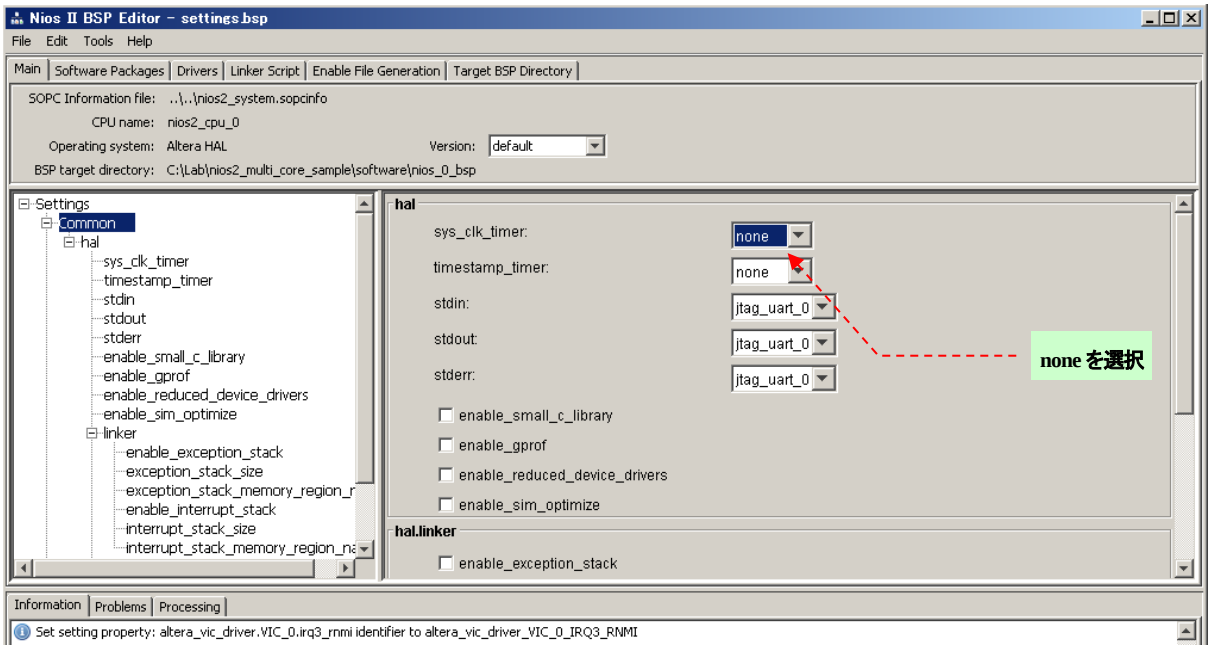

図 4-2-5

⑦ Linker Script タブを開いて図 4-2-6 のようにセクションとリージョンを確認します。Linker Section Name では .entry を除く全てのセクション（.bss,.heap,.text 等）が ext_ssram に設定されていることを確認し、Linker Region Name では ext_ssram がソフトウェア実行領域に、ext_flash がソフトウェア・イメージ領域（ヘッダ部除く）に設定されていることを確認します。※1

※ 1. 複数の Nios II を持つシステムでも、BSP はあらかじめリージョンが重ならないようなメモリ・スケジューリングを行いますので、本検証では変更の必要はありません。

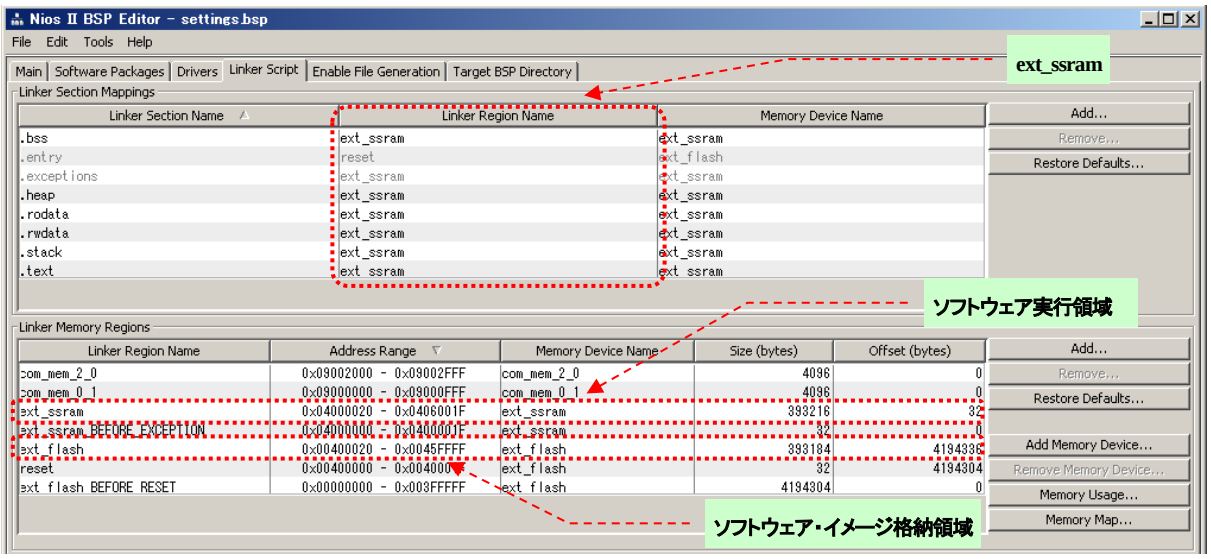

図 4-2-6

⑧ 続いて Drivers タブを開きます。本検証のハードウェア構成では割り込み制御をベクタ割込みコントローラ（Vectored Interrupt Controller）で実装していますので、多重割り込みの許可やシャドウ・レジスタの設定、ノンマスカブル割り込みの割り当てなどが行えます。本検証では変更の必要はありません。

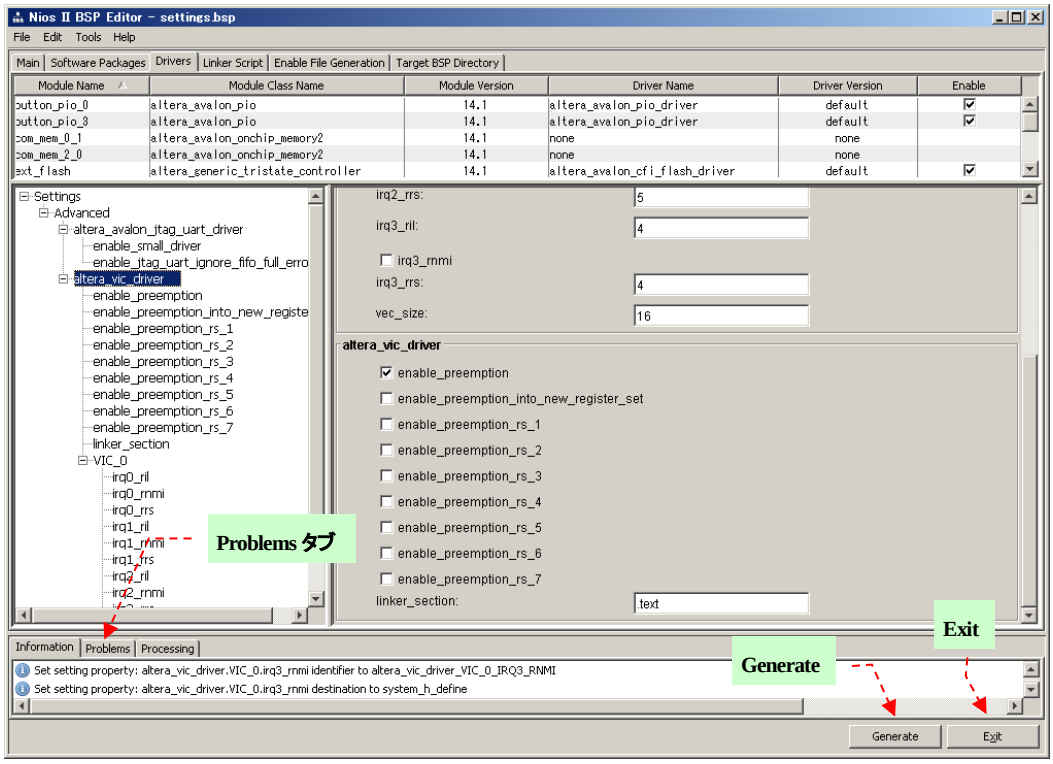

図 4-2-7

⑨ Problems タブを開いてエラーが無いことを確認したら Generate をクリックし、Exit で BSP Editor を閉じます。

⑩ Generate が正常に終了すると、BSP フォルダにリンカ・スクリプト・ファイル（linker.x）が生成されます。このファイルを開いて（開く際に確認画面が開く場合がありますが、F5 を押して開いてください）、共有メモリのセクションに (NOLOAD) を加えてください。

```
.com_mem_0_1 (NOLOAD) :
{
      PROVIDE (_alt_partition_com_mem_0_1_start = ABSOLUTE(.));
      *(.com_mem_0_1. com_mem_0_1.*)
      . = ALIGN(4);
      PROVIDE (_alt_partition_com_mem_0_1_end = ABSOLUTE(.));
} > com_mem_0_1

PROVIDE (_alt_partition_com_mem_0_1_load_addr = LOADADDR(.com_mem_0_1));

.com_mem_2_0 (NOLOAD) :
{
      PROVIDE (_alt_partition_com_mem_2_0_start = ABSOLUTE(.));
      *(.com_mem_2_0. com_mem_2_0.*)
      . = ALIGN(4);
      PROVIDE (_alt_partition_com_mem_2_0_end = ABSOLUTE(.));
} > com_mem_2_0

PROVIDE (_alt_partition_com_mem_2_0_load_addr = LOADADDR(.com_mem_2_0));
```

(NOLOAD) を追加

<< リンカ・スクリプト >>

複数のプロセッサが共有メモリを相互にアクセスする処理を実装した際、Nios II SBT からのソフトウェア・イメージのロード完了後に他のプロセッサが共有メモリの領域を書き替えてしまうため、ベリファイ・エラーが発生する場合があります。そのため、共有メモリをロード対象から外すことでベリファイを実行させないようにする必要があります。

なお、上記の処置を行うことで、ソフトウェア・イメージの中に共有メモリの初期化データが格納されなくなりますので、ソフトウェア処理の中で動的に初期化を行う必要があることに注意してください。

⑪ 再び BSP Editor を起動し、Enable File Generation タブを開きます。Generated Files の下の linker.x を選択した状態で右クリックし、Disable generation を選択します。こうすることで、BSP の Generate 対象から外すことができます。

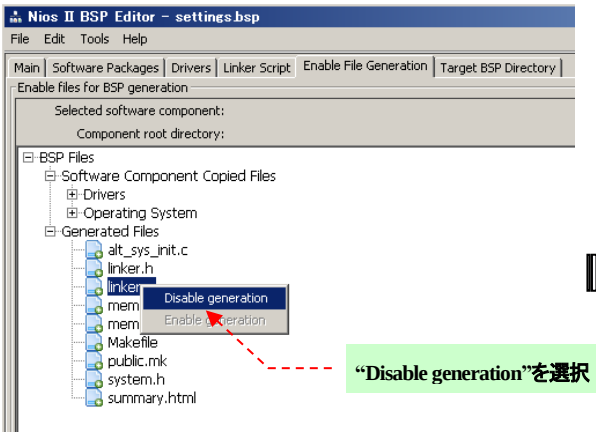

図 4-2-8-1

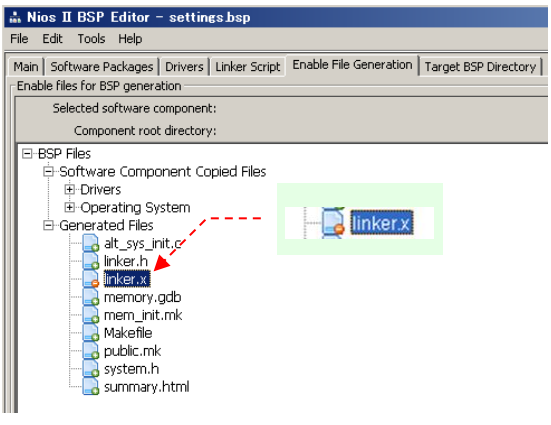

図 4-2-8-2

⑫ Problems タブを開いてエラーが無いことを確認したら Generate をクリックし、Exit で BSP Editor を閉じます。

⑬ Nios II SBT に戻り、Project Explorer のアプリケーション・プロジェクトを選択した状態で右クリックし、Build Project を選択し、ソフトウェアの Build を行います。

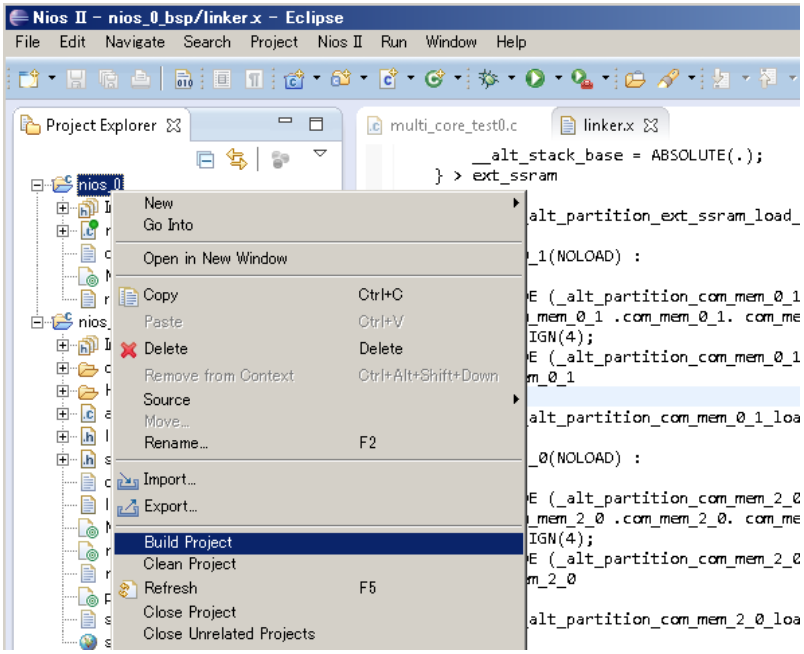

図 4-2-9

⑭ ソフトウェアのビルドが成功したら、再び 4-2 の先頭からその他の Nios II (nios_1 および nios_2) のプロジェクトを作成します。

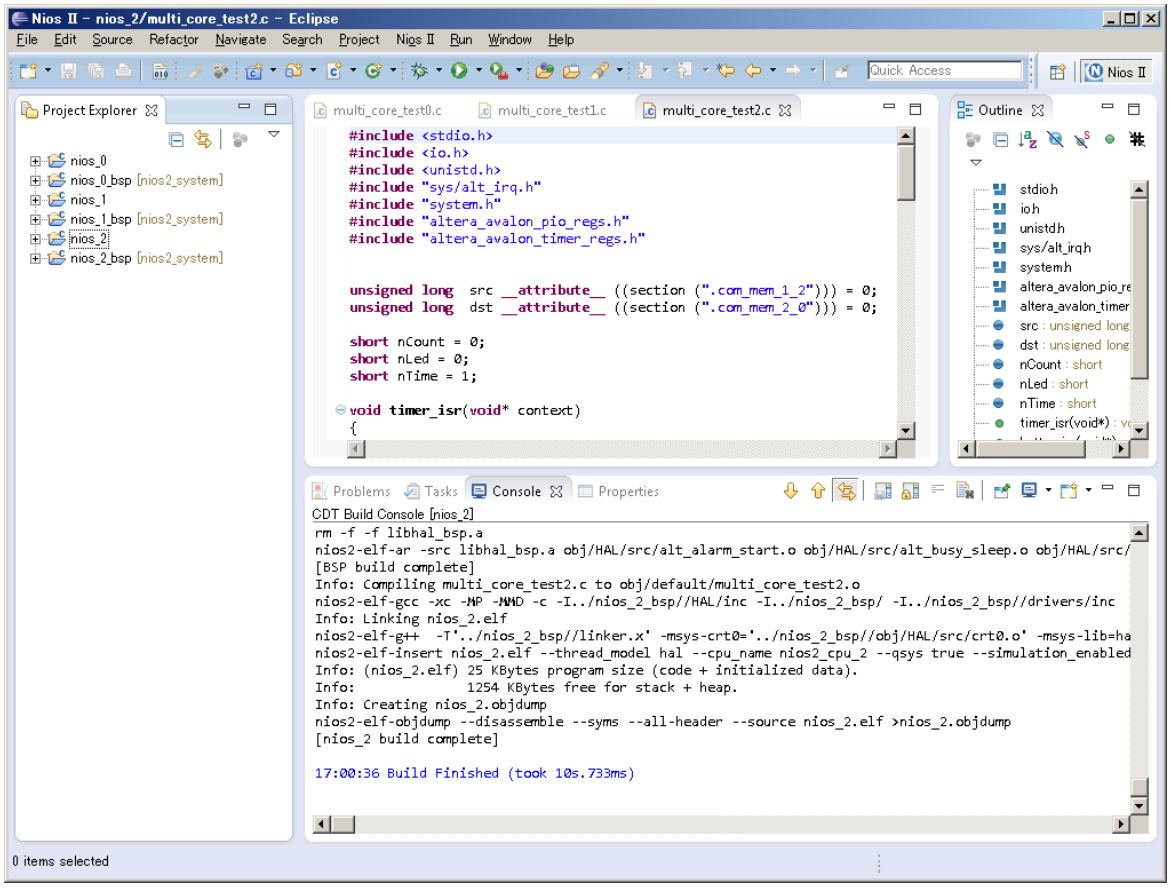

図 4-2-10

4-3. ソフトウェアの実行

① Nios II SBT の Nios II メニューから Quartus II Programmer を選択します。Quartus II Programmer が起動したら Auto Detect をクリックしデバイスを認識させます。続いて File 欄の <none> をダブルクリックしてファイルを選択し、Program/Configure にチェックを入れて Start をクリックし FPGA のコンフィグレーションを行います。

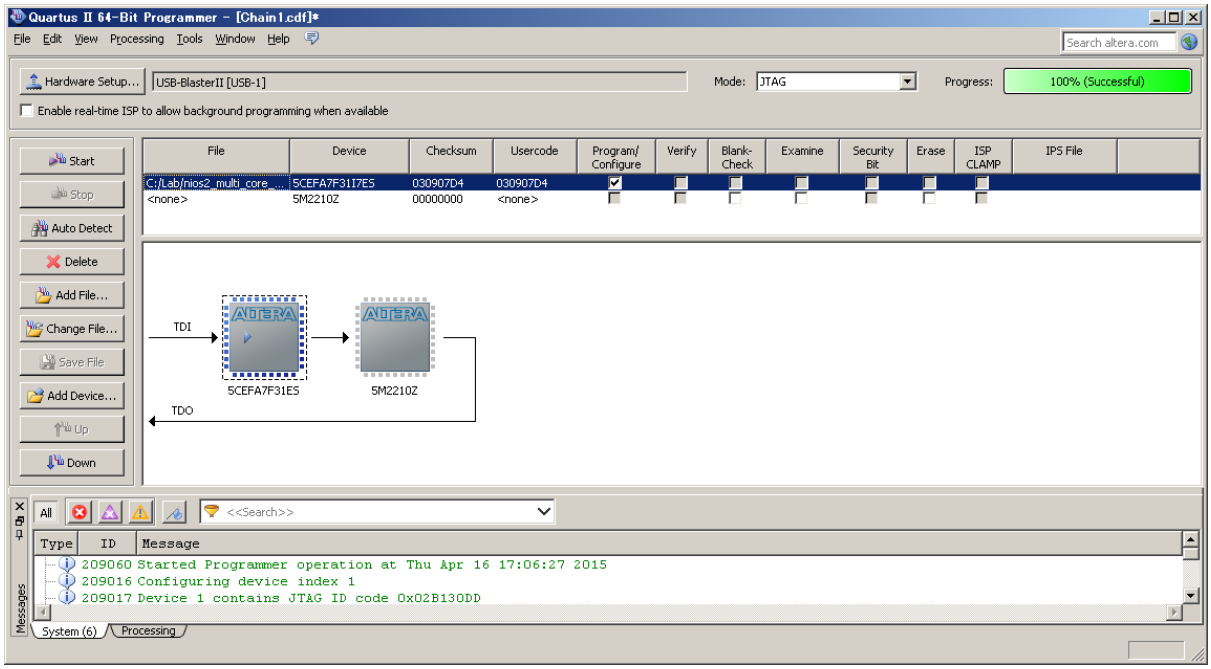

図 4-3-1

② Nios II SBT に戻り、Run メニューから Run Configuration... を選択し、Run Configurations を起動し、Nios II Hardware を選択した状態で右クリック、New を選択します。

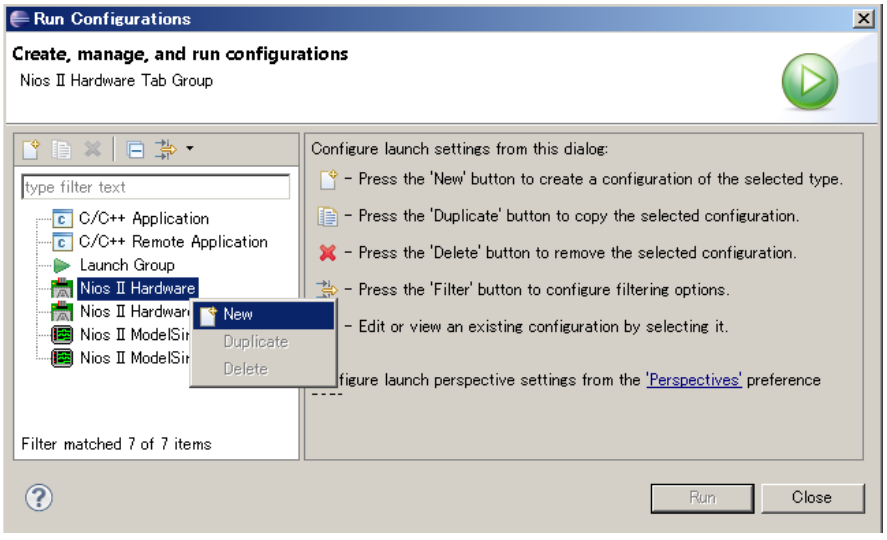

図 4-3-2

③ New Configuration が生成されたら Name に任意の名前を設定します。続いて Project タブの Project name を選択すると、自動的に ELF file name に .elf ファイルのパスが設定されます。

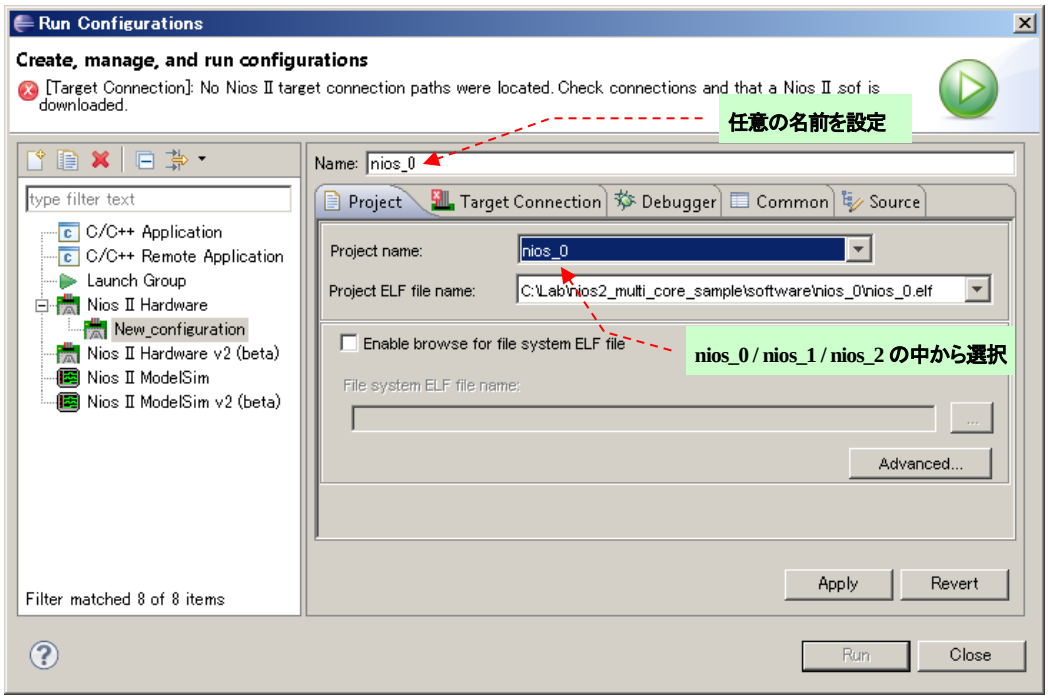

図 4-3-3

④ Target Connection タブを開いて、Processors をターゲット・プロセッサに、Byte Stream Devices をターゲット・プロセッサに接続されている JTAG-UART を選択します。それ以外はデフォルトのままとします。Processors や Byte Stream Devices が表示されていない場合は、Refresh Connections をクリックします。

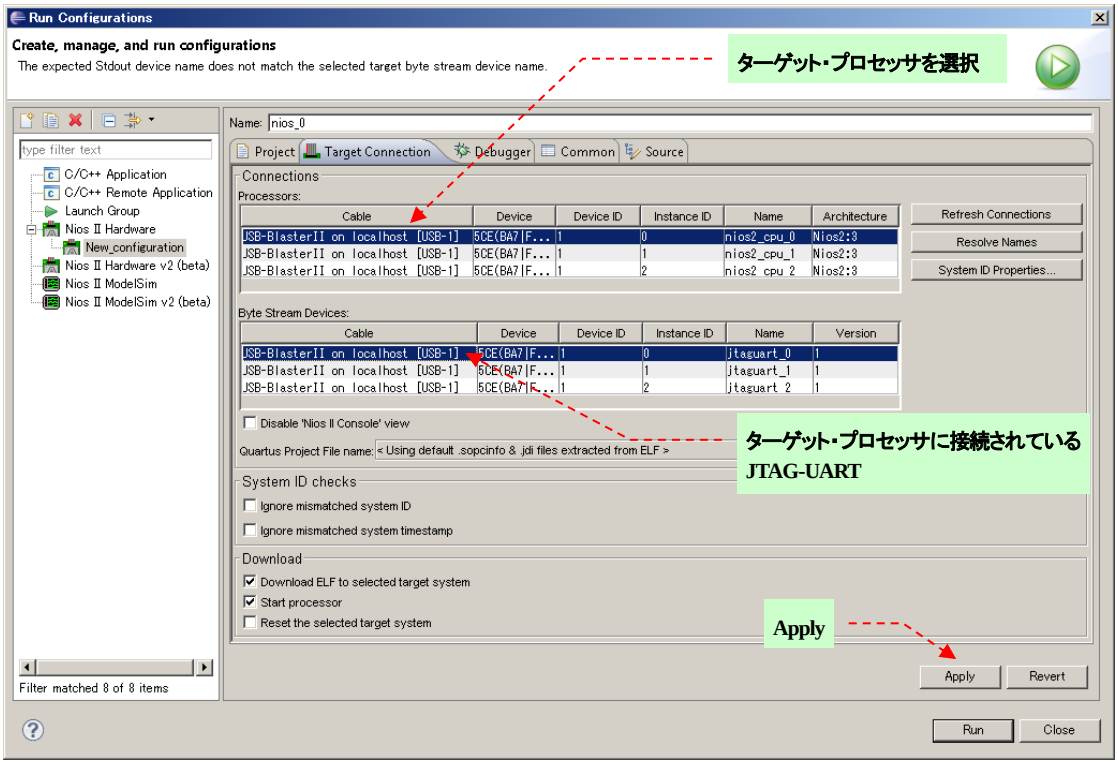

図 4-3-4

⑤ Apply をクリックして設定を確定します。

⑥ その他の Nios II の Run Configuration も、上記の操作で同様に設定したら Close をクリックしてウィンドウを閉じます。

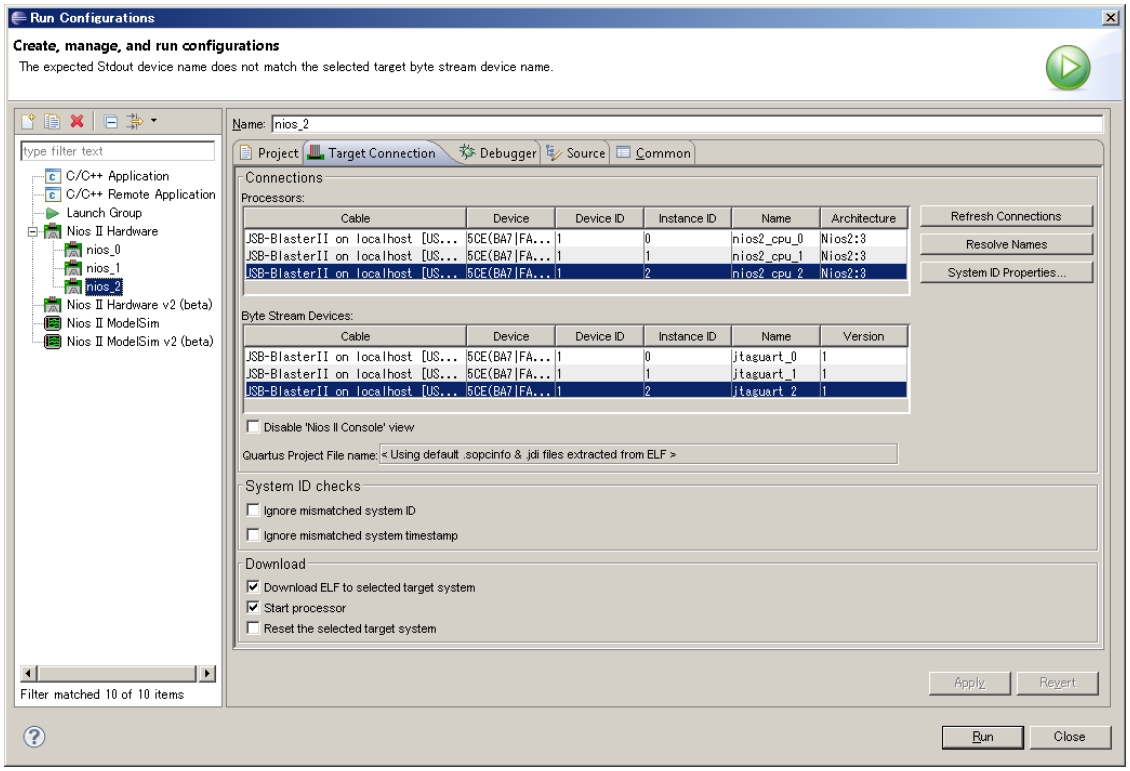

図 4-3-5

⑦ Nios II SBT に戻り、Project Explorer の各プロジェクトを選択した状態で右クリック、Run As ⇒ 3 Nios II Hardware を選択します。

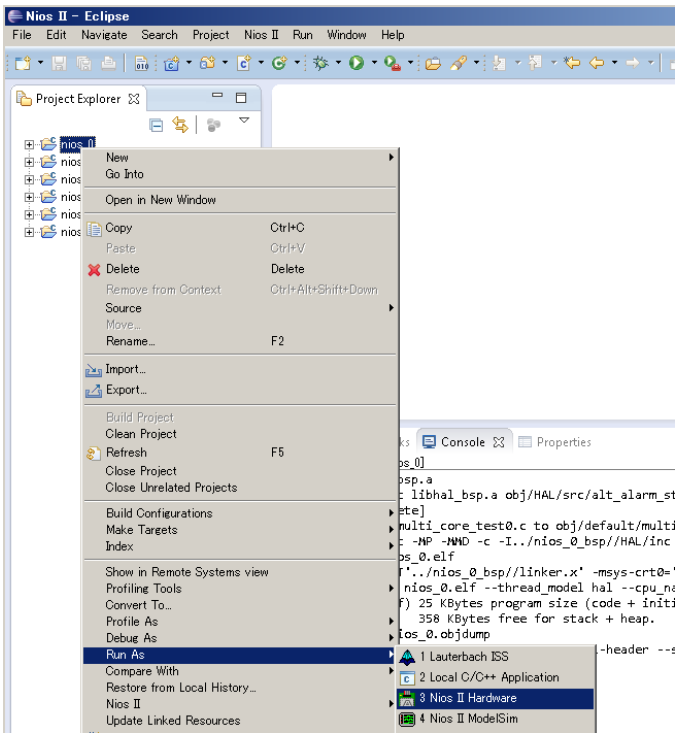

図 4-3-6

⑧　全てのプロセッサの Run を行うと、全ての LED が点滅を始めます。

LED0 から LED2 は各プロセッサが点滅周期を制御しており、BUTTON0 から BUTTON2 の押下で点滅周期を切り替えます。また、BUTTON3 の押下により、全てのプロセッサに対して点滅周期の初期化を行います。

LED3 は共有メモリを使ったプロセッサ間通信が正しく動作しているかどうかを示します。動作の仕組みに関しては図 4-3-7 を参照してください。

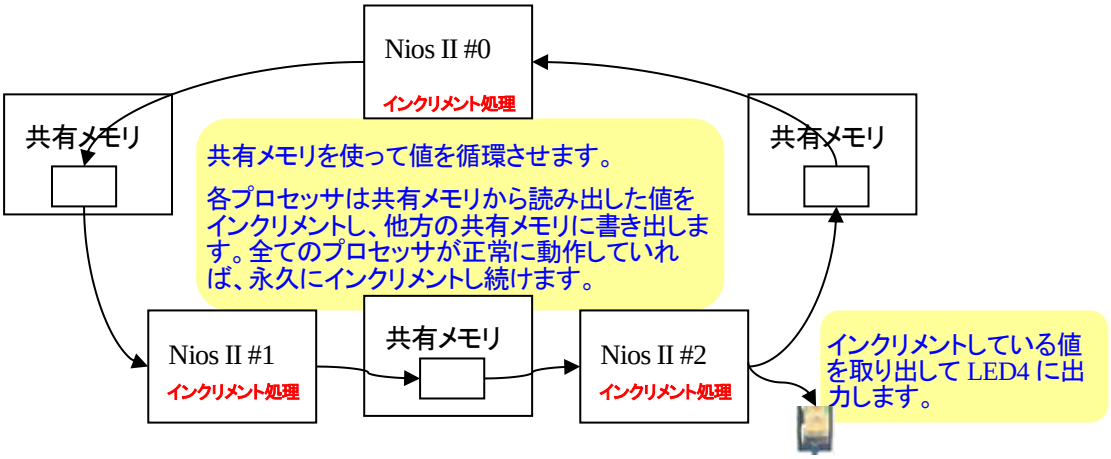

図 4-3-7

4-4. ハードウェア・イメージとソフトウェアの Flash ROM への書き込み

① デフォルトでは CFI Flash ROM の初期化処理が全ての Nios II ソフトウェアに含まれることになり、Flash ROM から起動させた場合、処理が競合してしまい正常に動作しませんので、Nios II #0 にのみ残し、Nios II #1 と Nios II #2 の初期化処理をコメント・アウトします。Nios II SBT の Nios II #1 プロジェクトと Nios II #2 プロジェクトの BSP フォルダの alt_sys_init.c ファイルを開き、下記の箇所を修正します。

```
        :
void alt_sys_init( void )
{
    ALTERA_AVALON_TIMER_INIT ( TIMER_1, timer_1);
//    ALTERA_AVALON_CFI_FLASH_INIT ( EXT_FLASH, ext_flash);
    ALTERA_AVALON_JTAG_UART_INIT ( JTAG_UART_1, jtag_uart_1);
}
        :
```

コメント・アウト

<< Nios II #1 の修正コード >>

```
        :
void alt_sys_init( void )
{
    ALTERA_AVALON_TIMER_INIT ( TIMER_2, timer_2);
//    ALTERA_AVALON_CFI_FLASH_INIT ( EXT_FLASH, ext_flash);
    ALTERA_AVALON_JTAG_UART_INIT ( JTAG_UART_2, jtag_uart_2);
}
        :
```

コメント・アウト

<< Nios II #2 の修正コード >>

修正が終了したら、Nios II #1 と Nios II #2 のプロジェクトを再ビルドします。

※詳細については、5 章. 複数の Nios II を構成する際の注意事項を参照ください。

② Nios II SBT の Nios II メニューから Flash Programmer を選択し、Nios II Flash Programmer を起動します。

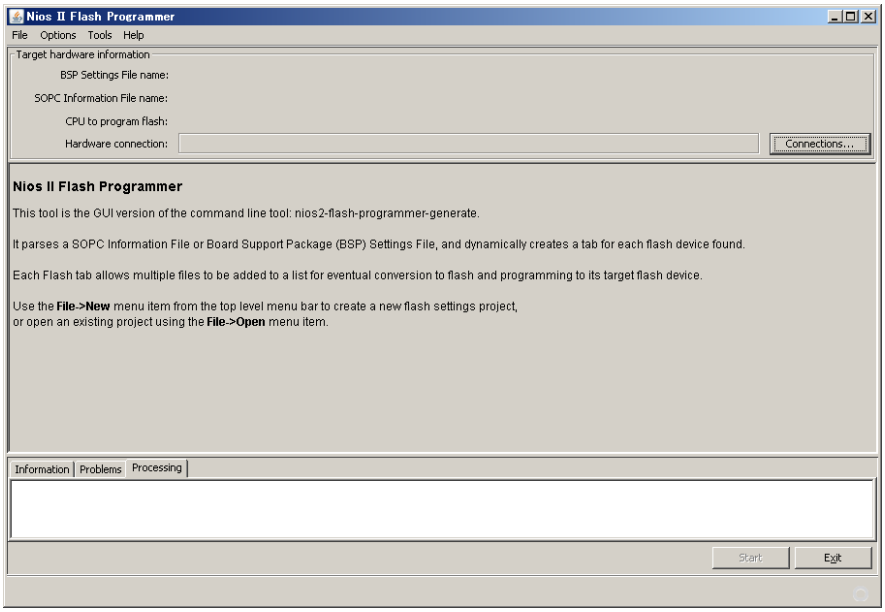

図 4-4-1

※ Flash Programmer を使って Flash ROM に書き込む際は、直前に本検証で作成したコンフィグレーション・データ(.sof)を書き込んでから行ってください。

③ Nios II Flash Programmer の File メニューから New を選択し、New Flash Programmer Setting File が起動したら、Get flash programmer system details from BSP Settings File または Get flash programmer system details from SOPC information File にチェックを入れ、各項目に BSP セッティング・ファイル※1、または、.sopcinfo ファイルを設定し、OK をクリックします。なお、選択する Nios II はどれでも結構です。

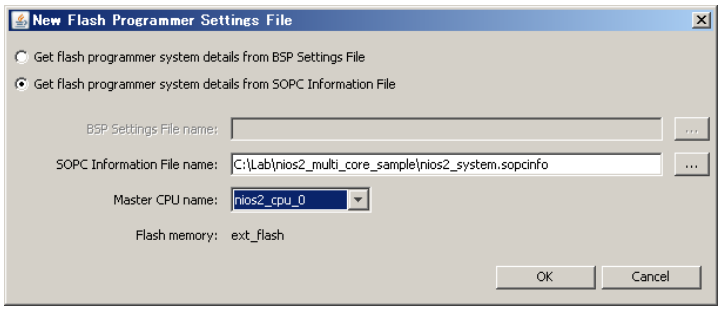

図 4-4-2-1

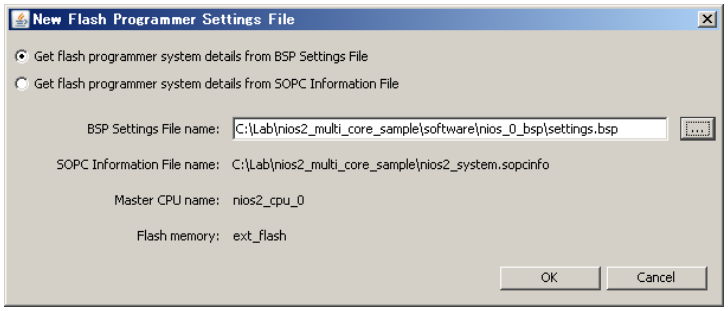

図 4-4-2-2

※ 1. BSP セッティング・ファイルは、デフォルトでは Nios II ソフトウェア・プロジェクト（software フォルダ）の BSP フォルダの中に settings.bsp と言うファイル名で存在します。

④ Nios II Flash Programmer の Connections... ボタンをクリックします。Hardware Connections ウィンドウが表示されたら、Processors: のリストから③で選択した Nios II を選択し、Close をクリックします。

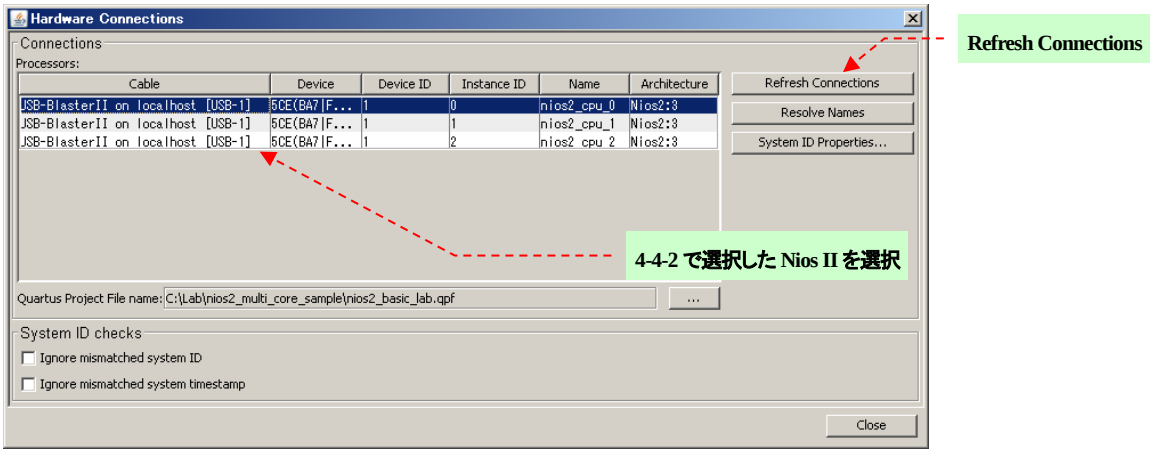

図 4-4-3

※ Processors: にリストが正しく表示されない場合は、Refresh Connections をクリックしてください。

※ System ID ミスマッチのエラーが出た場合には、System ID checks の 2 項目をチェックしてください。

⑤ Nios II Flash Programmer の Add ボタンをクリックします。ファイル選択ウィンドウが表示されたら Nios II #0 のソフトウェア・プロジェクト・フォルダに生成されている .elf ファイルを選択します。続いて Properties... ボタンをクリックして Properties ウィンドウを開きます。Properties ウィンドウの CPU reset address: には 0x4000000、Flash base address: に 0x0、Flash end address: に 0x400000 が設定されていることを確認します。Boot loader: には標準のブート・コピアが設定されていますのでそのままにして、Close で閉じます。

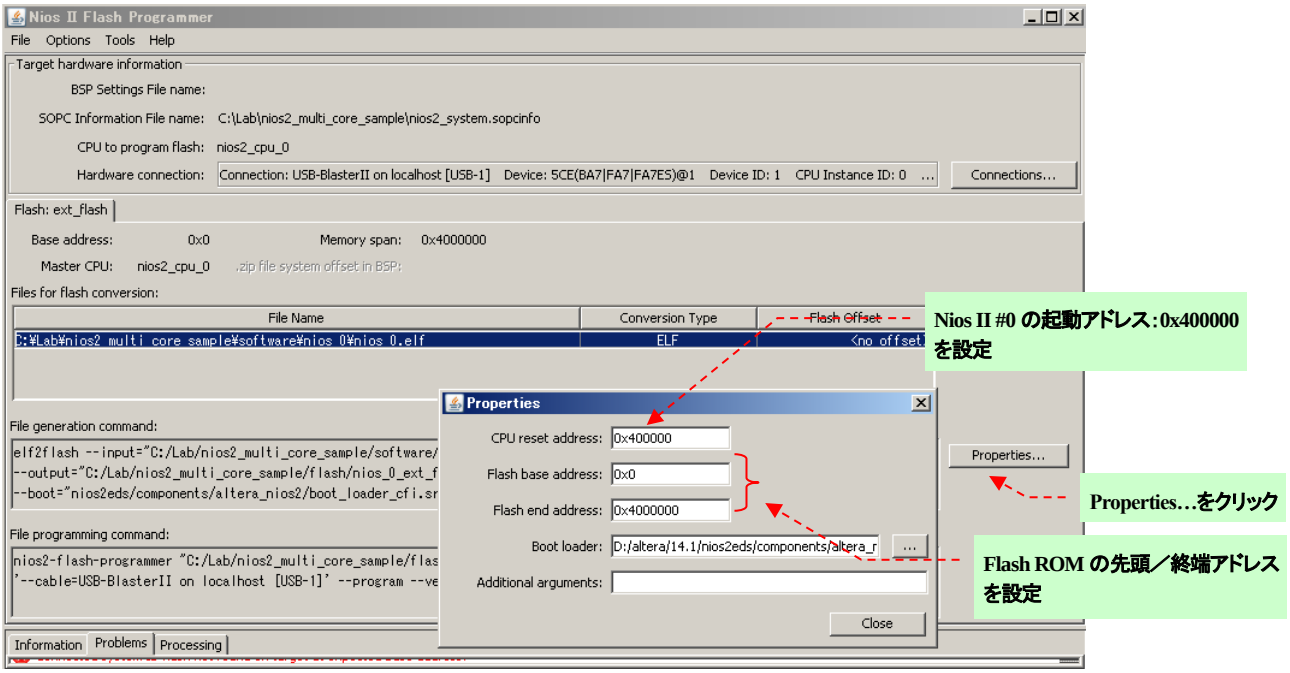

図 4-4-4

⑥ 再び Add ボタンをクリックし、ファイル選択ウィンドウが表示されたら Nios II #1 のソフトウェア・プロジェクト・フォルダに生成されている .elf ファイルを選択します。続いて Properties... ボタンをクリックして Properties ウィンドウを開きます。Properties ウィンドウの CPU reset address: には 0x460000 を設定します。Boot loader: には標準のブート・コピアが設定されていますのでそのままにして、Close で閉じます。Nios II #2 についても同様の手順で CPU reset address: には 0x4C0000 を設定します。

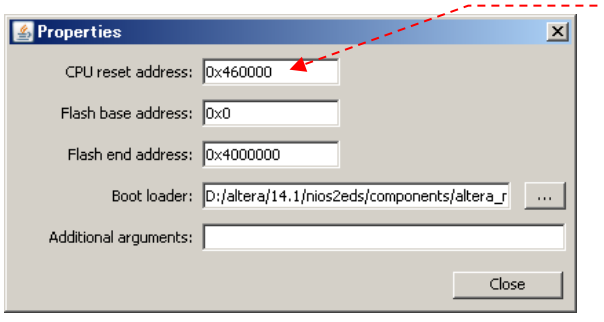

図 4-4-5

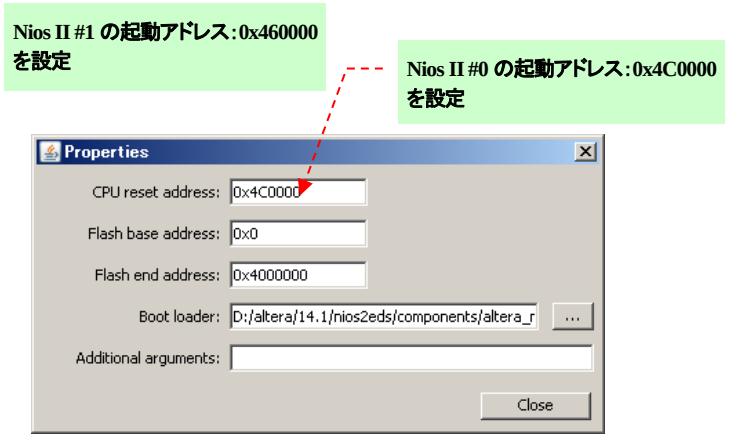

図 4-4-6

⑦ 再び Add ボタンをクリックし、ファイル選択ウィンドウが表示されたら本検証で作成した .sof ファイルを選択します。Flash Offset にはファスト・パッシブ・パラレル・モードでの User Hardware 1 コンフィグレーション・アドレスである 0x800000 を設定します。続いて Properties… ボタンをクリックして Properties ウィンドウを開きます。Properties ウィンドウの Additional arguments: に --pfl --optionbit=0x18000 --programmingmode=FPP と設定してください。

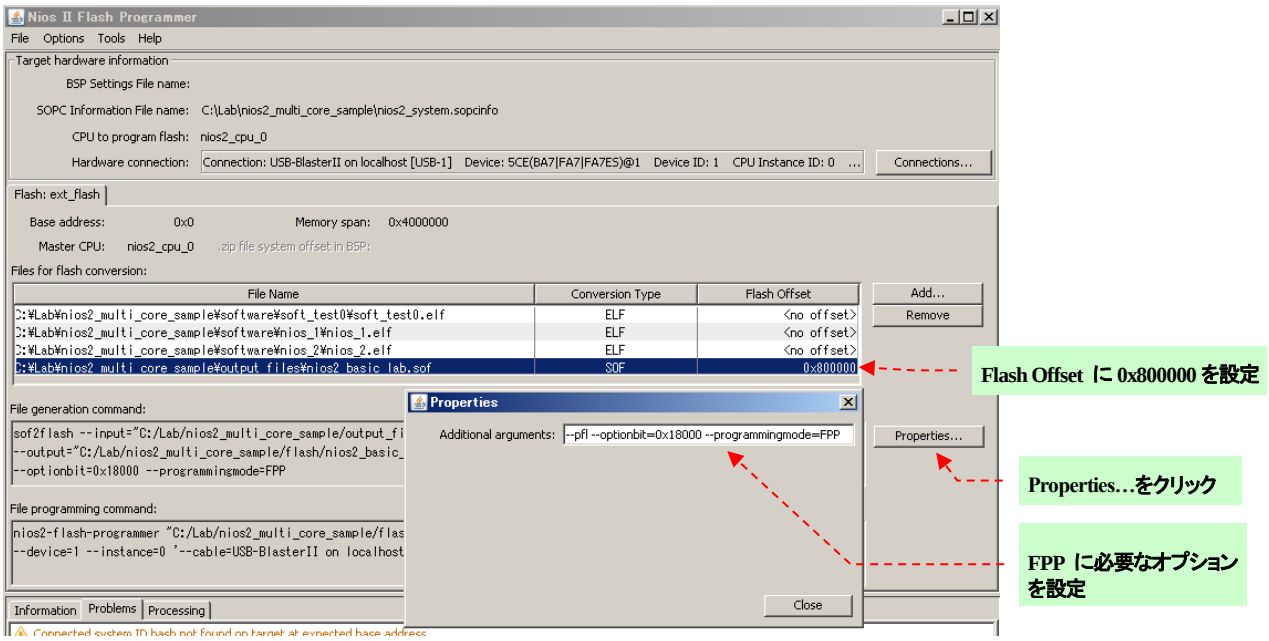

図 4-4-7

※User Hardware 1 コンフィグレーション・アドレスである 0x800000 や optionbit で指定している 0x18000 は、アルテラ社評価用ボードの設定となります。詳しくは、評価用ボードのマニュアルを参照ください。

⑧ start ボタンをクリックして Flash ROM にソフトウェア・イメージとコンフィグレーション・データを書き込みます。Processing にエラー・メッセージが表示されなければ終了です。ボードの PGM_RECONFIG ボタンを押下して、正しく動作するか確認してください。

※上記の設定では、評価用ボードの設定が、FPP（ファスト・パッシブ・パラレル）に設定されており、User Hardware1 のイメージがデフォルトでロードされる設定になっていることを確認してください。詳しくは、評価用ボードのマニュアルを参照ください。

5. 複数の Nios II を構成する際の注意事項

Nios II SBT で、ソフトウェアをビルドすると、Qsys の GUI 上で、Nios II と接続されているペリフェラル用の初期化コードを自動で生成します。この各ペリフェラルに対応した初期化コードで、ペリフェラルを制御するためにアルテラ社から提供された HAL を利用するための準備や各ペリフェラルの一般的な理想と考えられる初期状態のレジスタ設定等を行います。しかし、この自動で生成される各ペリフェラルの初期化コードは、複数の Nios II が構成されたシステムで、且つ複数の Nios II が同一のペリフェラルを共有させたシステムに対応したものでありません。

複数の Nios II を構成する場合においての注意事項に関して説明します。

```
#include "system.h"
#include "sys/alt_sys_init.h"

/*
* device headers
*/
#include "altera_avalon_timer.h"
#include "altera_avalon_uart.h"

/*
* Allocate the device storage
*/
ALTERA_AVALON_UART_INSTANCE( UART1, uart1 );
ALTERA_AVALON_TIMER_INSTANCE( SYSCLK, sysclk );

/*
* Initialize the devices
*/
void alt_sys_init( void )
{
    ALTERA_AVALON_UART_INIT( UART1, uart1 );
    ALTERA_AVALON_TIMER_INIT( SYSCLK, sysclk );
}
```

5-1. 排他制御に関して

周知のとおり、前提として複数のプロセッサがペリフェラルを共有する場合、排他処理が必要です。片方のプロセッサがあるペリフェラルに対して、何らかの処理をするためにアクセスしている最中に、他のプロセッサから同一ペリフェラルに対してのアクセスは避ける必要があります（フラグ情報を複数のプロセッサが共通で管理すること等で排他アクセスを実現する必要があります）。前述したように、Nios II SBT では Qsys で接続されているペリフェラル用に、初期化コードを自動で生成します（Qsys の GUI 上で、Nios II と接続されたペリフェラル用の初期化コードを自動生成します）。下の図の例のように、複数の Nios II に共通で接続されたペリフェラルが存在する場合には、それぞれの Nios II の初期化コード内に、共通で接続されたペリフェラル用の初期化コードが自動でリンクされますが、この初期化コードはこのような構成時に対応したものではありません。自動で生成されたコードを変更なく採用してしまうと、それぞれの、Nios II は同一ペリフェラルに対し排他制御をせず初期化を実行してしまいます。つまり、このような構成は、自動で生成される初期化コードをそのまま変更なく採用する場合において、許された構成ではありません。

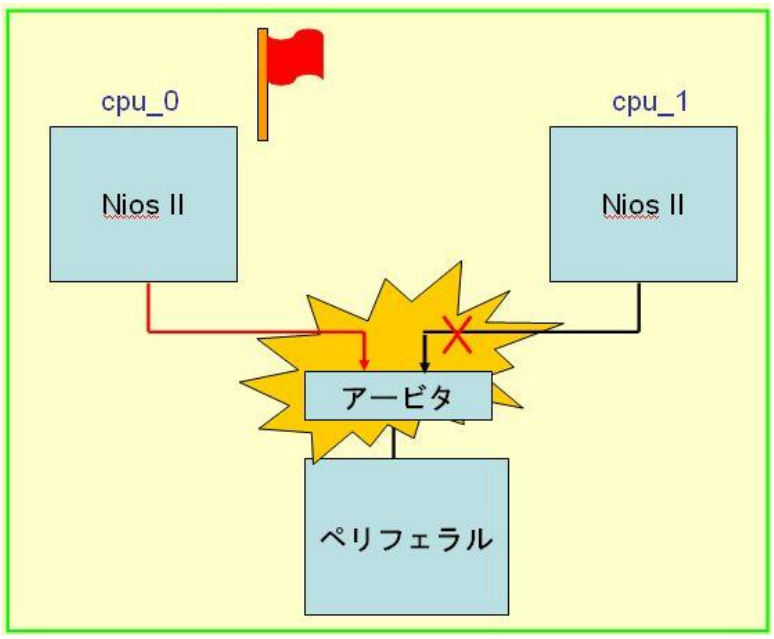

図 5-1-1

以下の例では、それぞれの Nios II から、共通のペリフェラルへアクセスができるようにハードウェアが接続されていますが、排他制御が非常に手間となり、また、割り込みがかかると目的でない方の Nios II も割り込みを認識してしまう理想的でない構成です。また、自動で生成される初期化コードでは動作保証がされない非推奨の構成となっています。

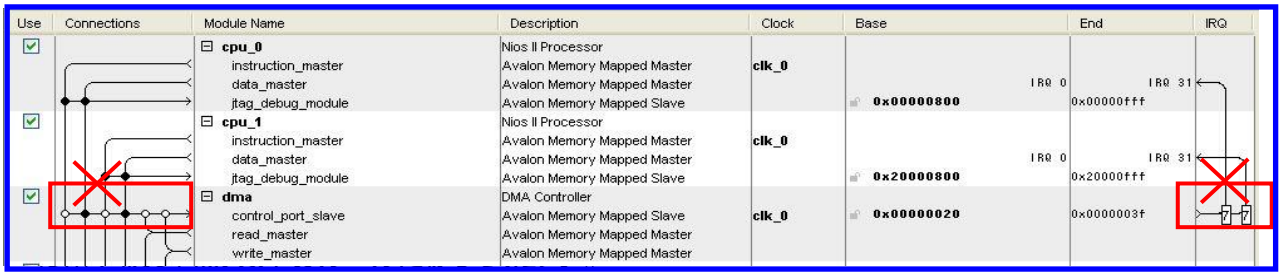

図 5-1-2

5-2. 自動で生成される初期化コードがサポートされる構成

複数の Nios II をシステムに実現する場合、それぞれの Nios II が単独で利用できるペリフェラルを複数個用意し、バスを完全に分離させた構成にします。このような構成であれば、ツールにより自動生成されたペリフェラル用の初期化コードを変更することなく利用できます。この自動生成される初期化コードは、それぞれの Nios II に接続されたペリフェラルに対してのみ用意されます。

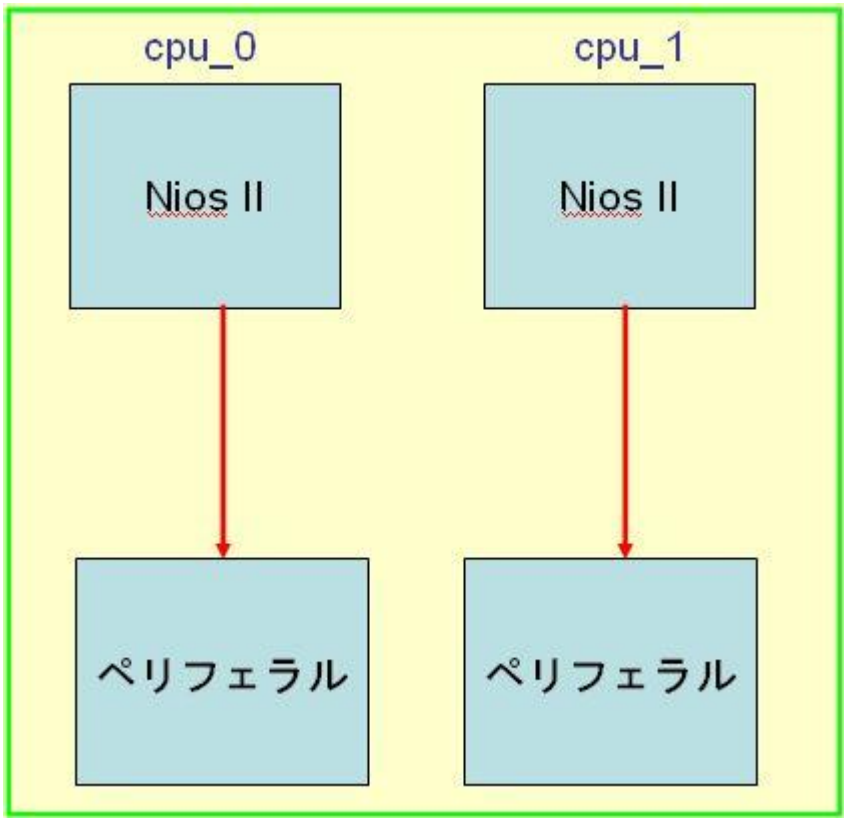

図 5-2-1

以下の例は、それぞれの、Nios II から個々のペリフェラルにのみアクセスができるハードウェア構成になっていて、割り込みポートも接続された Nios II にのみ有効になります。下の図で示されている DMA コンポーネントのみならず、UART、タイマ、SPI 等、すべてのペリフェラルは共有せず、それぞれの Nios II 用に専用で独立して接続することが推奨の構成となります。

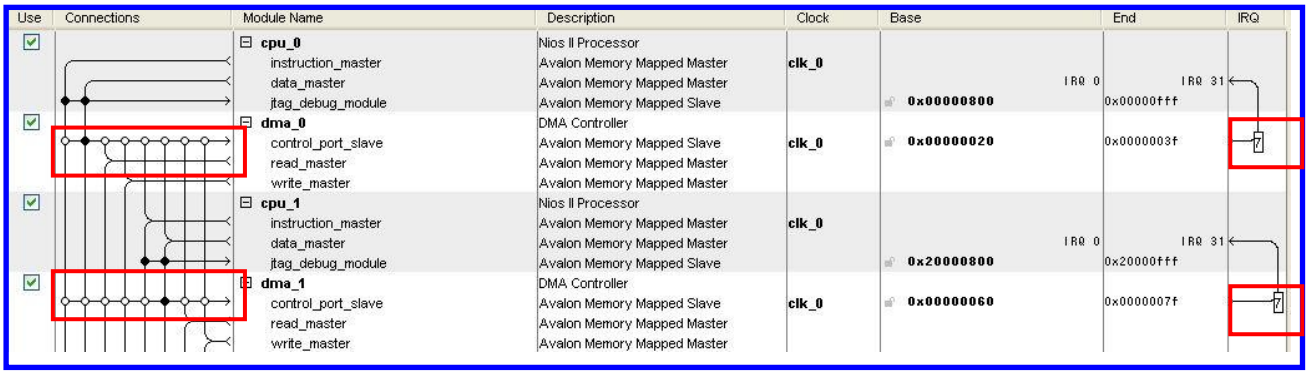

図 5-2-2

システム全体のスループットの向上、排他制御の不要等の利点より、すべてのペリフェラルは、それぞれのプロセッサごとに個別に用意し構成するのが理想的です。しかし、そのような構成がとれず、ペリフェラルを複数の Nios II で共有する場合には、前述のとおり自動で生成される初期化コードをそのまま採用すると問題を起こす可能性があります。

初期化コードを自動で生成しないペリフェラルと自動で生成するペリフェラルに関して次に示します。

<<自動で初期化コードを生成しないペリフェラル>>

- ユーザ・ペリフェラル
- Generic Tri-State Controller（CFI Flash を構成したもの）以外のメモリコンポーネント

（例: SDRAM 、DDR SDRAM コントローラ・コンポーネント、オンチップ・メモリコンポーネント等）

<<自動で初期化コードを生成するペリフェラル>>

- メモリ以外のすべてのアルテラ社から提供されるペリフェラル

詳細は、Nios II Software Developer's Handbook を確認してください。

https://www.altera.com/en_US/pdfs/literature/hb/nios2/n2sw_nii5v2.pdf

多くのペリフェラル用に初期化コードが用意されています。この自動で生成される初期化コードの処理内容を十分把握する必要があります。

例外として、System ID Core に関しては、自動で生成される初期化コード内に、マルチコア・システムにおいて、問題を引き起こす処理が含まれていなく、割り込みのポートも持たないため、複数の Nios II で共有することができます。また、もともと排他処理を目的として用意されているペリフェラル（ミューテックス・コア、メールボックス・コア）についても、複数の Nios II で共有することができます。Generic Tri-State Controller（CFI Flash を構成したもの）以外のメモリコンポーネントは、複数の Nios II でアクセスする領域が重ならないように利用する場合は共有できます。

それ以外のコンポーネントに関しては、複数の Nios II から同一のペリフェラルとして共有する場合には注意が必要です。

各初期化コード内で行われるもともとの処理内容を理解し、必要に応じてユーザ側で削除、または変更等をする必要があります。具体的な例としては、Generic Tri-State Controller（CFI Flash を構成したもの）であれば、フラッシュ・メモリをクエリモードに入れるコマンドやフラッシュ・メモリに存在するプログラムを RAM へ展開するブート動作が複数の Nios II から行われないように対処します。DMA コンポーネントであれば、初期化コード内の処理で割り込みの登録がされているので、1つの Nios II が割り込みを受け付ける状態に入った後、異なる Nios II が初期化コードを実行することで IRQ のノードが一瞬だけ成立してしまい、先に割り込みを受け付けた Nios II が例外処理を実行するが、その IRQ のパルス幅が狭いことが原因で正しい例外処理が行われず先に割り込みを受け付けた Nios II が無限ループして停止してしまうことを防ぐために、自動で生成される初期化コードは削除し、カスタムで初期化を行います。これらの例のように各ペリフェラルに応じて対処が必要です。

5-3. まとめ

Nios II でマルチ・プロセッサ・システムを構成する際、様々なペリフェラルはそれぞれの Nios II ごとに個別に用意しバスを分離することを推奨します。また、複数の Nios II から、同一のペリフェラルを共有する場合には、自動で生成されるそのペリフェラル用の初期化コードでは、システムの動作保証がされていないため、ユーザの方で問題が起きないように対処が必要となります。

改版履歴

Revision	年月	概要
1	2015 年 5 月	初版