

この資料は英語版を翻訳したもので、内容に相違が生じる場合には原文を優先します。こちらの日本語版は参考用としてご利用ください。設計の際には、最新の英語版で内容をご確認ください。

2009 年 4 月

AN-543-1.0

## 概要

このアプリケーション・ノートでは、Lauterbach Trace32 ロジック開発システムによって Nios II アプリケーションをデバッグする方法について説明します。

Lauterbach PowerTrace ハードウェアおよび TRACE32 PowerView 統合開発環境 (IDE) を含む TRACE32 システムは、Nios II システムの動作に完全な可視性を提供します。Nios II EDS、SOPC Builder、および Quartus® II ソフトウェアと併用することで、TRACE32 システムは Nios II システム障害または異常なデザイン動作を解析することを可能にします。Lauterbach TRACE32 システムは、ほかの Nios II デバッグ環境で比類のないコントロール性を提供しています。

このアプリケーション・ノートでは、Altera® Cyclone® III 3C120 開発ボード上のデザイン例をデバッグするチュートリアル・ステップ、およびカスタム・デザインのデバッグに関する一般ガイドラインについて説明します。

このアプリケーション・ノートでは、Windows プラットフォームで動作する Lauterbach TRACE32 PowerView IDE が使用されていることを前提としています。ただし、ここで述べた手法はプラットフォームから独立しています。



Lauterbach TRACE32 ロジック開発システムに関する詳細な説明については、[www.lauterbach.com](http://www.lauterbach.com) を参照してください。

## Lauterbach ツールの診断能力

Lauterbach TRACE32 ハードウェアおよびソフトウェア・ツールは、エンベデッド・システム開発に関する最も困難な問題 (タイム・ドメインに関わる問題を含む) を診断および解決する非常に強力な機能を提供します。Lauterbach ツールにより、次の一般的なデバッグ・タスクが実行できます。


- ブレークポイントおよびウォッチポイントを設定する
- コードをステップする
- 変数の値を検査する

また、下記のようなより強力な手法を実行することもできます。

- 特定の変数への書き込みを監視する
- 完全なプログラムの実行トレースをキャプチャおよびレビューする

## 実行トレース


Quartus II ハードウェア・デザイン内の Nios II プロセッサに MICTOR 接続を含めることで、押し付けることなく Lauterbach PowerTrace ハードウェアを通して大量のオフチップ実行トレース・データをキャプチャして、プログラム実行履歴に対する徹底的な解析を可能にします。この機能をトリガ条件と組み合わせて、システム障害が起きるまでと起こった直後の Nios II 実行アクティビティをキャプチャすることができます。

 実行トレース・データを検査するためには、Lauterbach LA-3801 Nios II トレース・モジュールが必要です。LA-3801 がなくても、ブレークポイントやウォッチポイントなど TRACE32 システムのほかの機能を使用することができます。


## 予備知識

本資料は、以下のトピックに対する基本的な知識を持っている上級システム開発者向けに書かれています。

- Nios II アプリケーション開発
- Quartus II ソフトウェア
- Lauterbach TRACE32 PowerView IDE

 最低限の予備知識を得るためには、次の資料を参照してください。


- [Nios II Development Kit Getting Started User Guide](#)
- [Nios II Hardware Development Tutorial](#)
- Nios II ソフトウェア開発チュートリアル。このチュートリアルにアクセスするには、Nios II IDE のウェルカム・ページで **Tutorials** をクリックします。
- TRACE32 PowerView IDE と共にインストールされている資料。「*Nios II Debugger and Trace*」([debugger\\_nios.pdf](#)) は特に重要です。

 TRACE32 ロジック開発システムに関する資料は [www.lauterbach.com](http://www.lauterbach.com) から入手できます。

## ソフトウェア要件

次のソフトウェア・コンポーネントが必要です。

- Altera Complete Design Suite v9.0 またはそれ以降
- Nios II プロセッサ用 Lauterbach TRACE32 PowerView IDE v9.0 またはそれ以降

 TRACE32 IDE をインストールした後に、[www.lauterbach.com](http://www.lauterbach.com) で、ソフトウェアを最新リリースに更新することが推奨されています。

- **lauterbach.zip** ファイル。このファイルは、アルテラ・ウェブサイトの [Literature: Nios II Processor](#) ページで、本資料の下にあります。表 1 に、**lauterbach.zip** の内容を示します。

**lauterbach.zip** を作業ディレクトリに解凍します。作業ディレクトリのパス名にスペースを含めないように注意してください。作業ディレクトリを以降、`<working_directory>` と表記します。

表 1. lauterbach.zip の内容

ディレクトリ名	内容
mictor	アルテラ Cyclone III 3C120 開発ボードおよびアルテラ THDB-SUM アダプタボードのための Nios II ハードウェア・デザイン例。 このハードウェア・デザインには、Trace Output Buffer コンポーネントが含まれています。
mictor\software_examples\app	<b>nested_loops.c</b> におけるネステッド・ループのアプリケーション。
schematics	<ul style="list-style-type: none"> <li>■ アルテラ Cyclone III 3C120 開発ボードおよびアルテラ THDB-SUM アダプタボードの回路図。</li> <li>■ デザイン例におけるトレース・データ・ピンのマッピングを記載する <b>Mictor_Trace_Connections_HSMA_3c120.xls</b></li> </ul>
board_config	次のボード・リコンフィギュレーション・ファイル。 <ul style="list-style-type: none"> <li>■ <b>3C120_flash_option_bits.jbc</b></li> <li>■ <b>m2_prod_1338.pof</b></li> </ul> <p>詳細については、<a href="#">ページ 21</a> の「特別なハードウェア手法」を参照してください。</p>
troubleshooting	TRACE32 システムおよび Nios II OCI コア間の通信問題のトラブルシューティングに役立つファイル。 <ul style="list-style-type: none"> <li>■ <b>lauterbach_system_up_debug_script.cmm</b></li> <li>■ <b>good_debug.log</b></li> <li>■ <b>good_debug.lst</b></li> <li>■ <b>failed_debug.log</b></li> </ul>

## ハードウェア要件

このセクションでは、Lauterbach TRACE32 ロジック開発システムによるデバッグ作業のハードウェア要件について説明します。

### Lauterbach PowerTrace ハードウェア

Lauterbach PowerTrace ハードウェアは以下のコンポーネントから構成されています。

- Lauterbach LA-7707 PowerTrace Ethernet 256 MB Universal NEXUS/ デバッグ・コントローラ
- NIOS-II (ICD) 用 Lauterbach LA-7837 デバッガ
- NIOS-II Flex Cable 用 Lauterbach LA-3801 プリプロセッサ



トレース機能を使用しない場合、LA-3801 はオプションです。




Lauterbach 回路を損傷から保護するためには、正しい順序でターゲット・ボードおよび Lauterbach PowerTrace ハードウェアに電源を提供することが重要です。[ページ 10](#) の「TRACE32 ロジック開発システムの起動」をご覧くださいになる前に、Lauterbach PowerTrace ハードウェアを接続または電源投入してはいけません。

## ターゲット・ハードウェア

Lauterbach TRACE32 デバッグ・システムをサポートするために、ターゲット・ボードは次の要件を満たす必要があります。


- FPGA デバイスが Nios II プロセッサをサポートする
- ボードが 10 ピンの JTAG ヘッダを備える
- ボードが MICTOR ソケットを備える


 トレース機能を使用しない場合、MICTOR ソケットは不必要です。

### チュートリアルターゲット・デバイス


本資料でのチュートリアル・ステップでは、次のハードウェア・コンポーネントが使用されます。


- アルテラ Cyclone III 3C120 開発ボード

 Cyclone III 開発ボードについて詳しくは、[「Cyclone III 3C120 Development Board Reference Manual」](#)を参照してください。

 ページ 22 の「Cyclone III 開発ボードにおける MAX II デザインの更新」およびページ 24 の「Cyclone III 開発ボード上のフラッシュ・オプション・ビットの更新」で述べたハードウェア・コンフィギュレーション・ステップを実行する必要がある場合があります。


- アルテラ THDB-SUM アダプタ・ボード。トレース・データ用の MICTOR ソケットを提供するためには、THDB-SUM ボードをアルテラ Cyclone III 3C120 開発ボード上の高速メザニン・コネクタ (HSMC) のポート A に接続します。

 THDB-SUM ボードについて詳しくは、[「Santa Cruz, USB, Mictor, SD Card HSMC Reference Manual」](#)を参照してください。

 トレース機能を使用しない場合、THDB-SUM ボードは不必要です。

## FPGA デザイン要件

ハードウェア・デザインは、ページ 6 の「ハードウェア・デザインの準備」に示すように Nios II プロセッサを MICTOR ソケットに接続しなければなりません。

 トレース機能を使用しない場合、MICTOR 接続は不必要です。

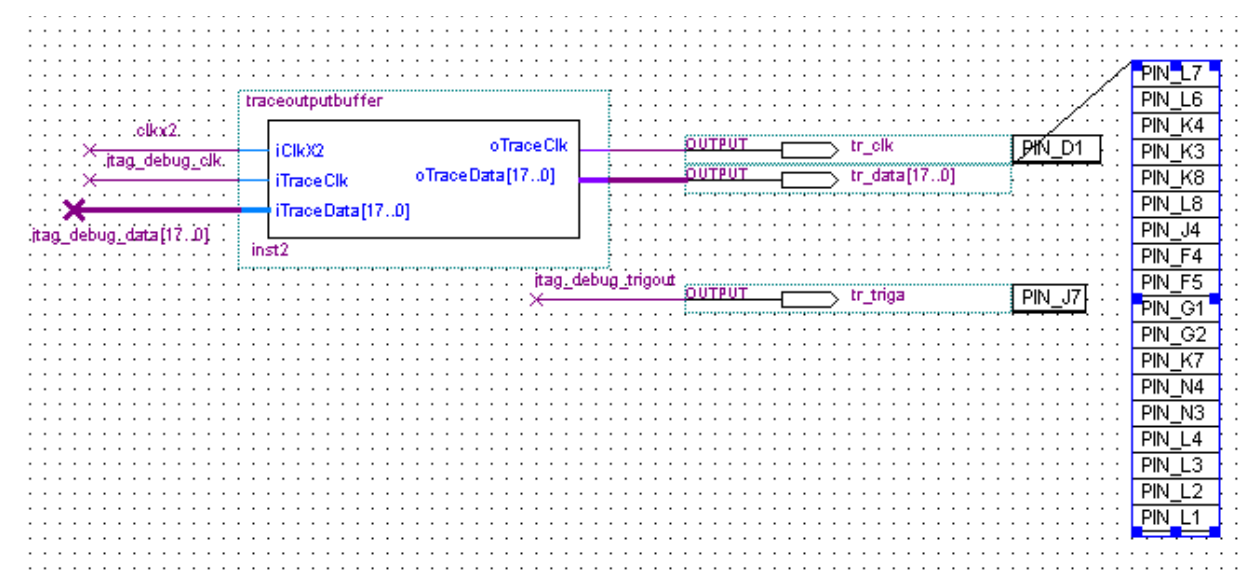
デザイン例では、HSMA というラベルの付いた HSMC コネクタに接続された THDB-SUM アダプタ・ボードを持つ Cyclone III 3C120 開発ボードが使用されています。THDB-SUM ボードは、Lauterbach トレース・ハードウェアとの接続用の MICTOR ソケットを備えています。

<working\_directory>\schematics ディレクトリには、**Mictor\_Trace\_Connections\_HSMA\_3c120.xls** というファイルがあります。このスプレッドシートでは、デザイン例におけるトレース・データ・ピンのマッピングを記載しています。このスプレッドシートには、FPGA デザインにおけるトレース・データ・ピン (**FPGA Logic** のカラム)、および MICTOR に接続した FPGA の物理ピン (**FPGA Pin** のカラム) が含まれています。

このマッピングは、<working\_directory>\schematics ディレクトリにおける Cyclone III 開発ボードおよび THDB-SUM アダプタ・ボードの回路図から得られたものです。図 1 に、これらのピン・マッピングを示します。

カスタム・デザインでは、回路図から正しいトレース・データ・ピンを決定します。マッピングは Quartus II pin planner で実装します。

図 1. トレース出力バッファおよびピン・アサインメント



## Lauterbach によるデバッグ作業のために Cyclone III 開発ボードを準備

アルテラの Cyclone III 3C120 開発ボードは高度にコンフィギュレーション可能です。Lauterbach トレース・モジュールを MICTOR 接続を通して FPGA にアクセスすることを可能にするため、および Lauterbach デバッグ・ケーブルを 10 ピン JTAG ヘッダを通して FPGA にアクセスすることを可能にするためには、いくつかのクリティカルなハードウェア設定をコンフィギュレーションしなければなりません。このセクションでは、必要な設定について説明します。

下記のように Cyclone III 開発ボードをコンフィギュレーションします。

- THDB-SUM アダプタ・ボードを、HSCA というラベルの付いた Cyclone III 開発ボード上の HSMC コネクタに直接接続します。



トレース機能を使用しない場合、THDB-SUM を接続しないでください。

- SW3 というラベルの付いた小型の DIP スイッチをコンフィギュレーションします。SW3 で、スイッチ 4 を 1 に設定します。(スイッチ 4 にも MAX\_ENABLE というラベルが付いている。) このコンフィギュレーションにより、Lauterbach LA-7837 デバッガなどの外部デバイスは 10 ピン JTAG ヘッダを通して JTAG チェインをコントロールすることができます。
- SW1 で、MAX0 というラベルの付いたスイッチ 5 を 1 (OPEN) に設定します。
- ジャンパ J6 を取り外します。
- PCM CONFIG SELECT ロータリー・スイッチを 0 に設定します。


 Cyclone III 開発ボードのコンフィギュレーションについて詳しくは、「[Cyclone III 3C120 Development Board Reference Manual](#)」を参照してください。

## ハードウェア・デザインの準備


Lauterbach TRACE32 システムによって Nios II デザインをデバッグするには、Nios II プロセッサの JTAG デバッグ・モジュールをコンフィギュレーションしなければなりません。ハードウェア・ブレイクポイント、データ・トリガ、および (必要な場合に) オフチップ・トレースをイネーブルするためには、JTAG デバッグ・モジュールをレベル 4 に設定します。

更に、TRACE32 ロジック開発システムを使用するには、Nios II プロセッサをターゲット・ボード上の MICTOR ソケットに接続しなければなりません。

Quartus II ソフトウェアおよび SOPC Builder を使用してこれらの準備をします。


 **lauterbach.zip** には、これらの準備ステップの完了した Cyclone III 3C120 開発ボード用の Quartus II プロジェクトが含まれています。そのプロジェクトは **mictor** ディレクトリにあり、ファイル名は **niosII\_cycloneIII\_3c120\_host\_board\_top.qpf** です。

## JTAG デバッグ・モジュールをレベル 4 に設定

 デザイン例を使用する場合、このセクションで述べたステップはすでに完了しました。これらのステップを完全に理解するためには、SOPC Builder でデザイン例を検証しながらこれらのステップを読むことができます。


Lauterbach TRACE32 ロジック開発システムで作業するには、下記のように、Nios II プロセッサ内の JTAG デバッグ・モジュールをレベル 4 に設定しなければなりません。

1. Quartus II プロジェクトを開いて、SOPC Builder を起動します。
2. SOPC Builder で、Nios II プロセッサをダブル・クリックして、Nios II processor MegaWizard™ インタフェースを開きます。
3. **JTAG Debug Module** ページをクリックして、**Level 4** を選択します。
4. ハードウェア・デザインが Stratix® シリーズのデバイスをターゲットにする場合、**Advanced Debug Settings** で、**Automatically generate internal 2X clock signal** をオフにして、2 番目の PLL の生成をディセーブルします。

 追加の PLL がないため、このステップは Cyclone シリーズにとって不必要です。

5. **Generate** をクリックして SOPC Builder システムを再生成します。
6. Quartus II ソフトウェアで、ハードウェア・デザインの回路図を表示します。
7. 回路図における SOPC Builder モジュールのシンボルを右クリックし、**Update Symbol or Block** をクリックします。
8. **OK** をクリックします。

## ハードウェア・デザインに MICTOR 接続を追加

 デザイン例を使用する場合、このセクションで述べたステップはすでに完了しました。これらのステップを完全に理解するためには、Quartus II ソフトウェアでデザイン例を検証しながらこれらのステップを読むことができます。

トレース・データを送信するために、デザインに MICTOR 接続が必要です。この接続を追加するには、**lauterbach.zip** に含まれる Trace Output Buffer コンポーネント (5 ページのステップ図 1 を参照) を SOPC Builder モジュールおよび MICTOR の間に挿入することができます。このセクションでは、デザインを Trace Output Buffer を通して MICTOR に接続する方法について説明します。

Trace Output Buffer データ中心のコンポーネントはトレース・データをバッファして、半クロック・サイクルを追加することでデータ信号をシフトします。このシフトによって、トレース・データがトレース・クロック信号エッジの近くでサンプルされないことを確保します。

表 2 に、トレース・データのキャプチャに関連する SOPC Builder モジュールのトップレベル信号を示します。これらの信号は、「JTAG デバッグ・モジュールをレベル 4 に設定」でイネーブルにした JTAG デバッグ・モジュールによってモジュールのトップレベルにエクスポートされます。

表 2. トレース出力信号

SOPC Builder モジュールにおける信号名	traceoutputbuffer における信号名
jtag_debug_offchip_trace_clk_from_the_cpu	iTraceClk
jtag_debug_offchip_trace_data_from_the_cpu[17..0]	iTraceData [17..0]
jtag_debug_trigout_from_the_cpu	(1)
clkx2_to_the_cpu	iClkX2

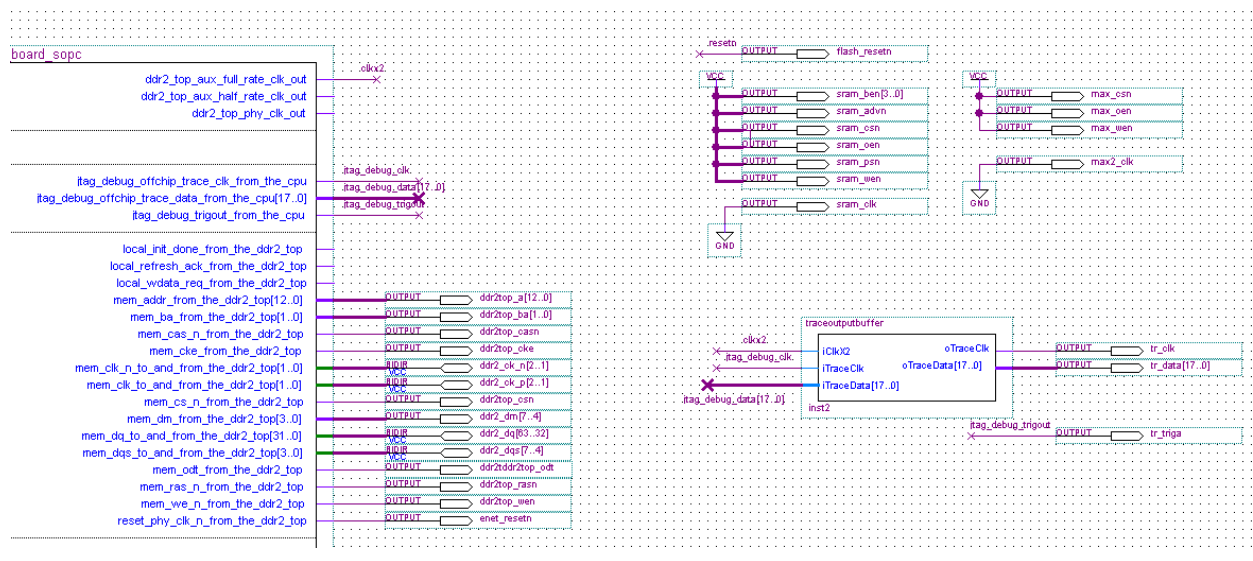
表 2 の注：

- (1) jtag\_debug\_trigout\_from\_the\_cpu は FPGA 上のトップレベル出力ピンに直接接続します。
- (2) clkx2\_to\_the\_cpu は、Nios II プロセッサのクロック速度の 2 倍で動作します。

Trace Output Buffer コンポーネントは、図 2 に示すように SOPC Builder ブロックに接続しなければなりません。Trace Output Buffer コンポーネントは、表 2 に示すデバッグ信号をトップレベルにエクスポートします。

図 2 はハードウェア・デザイン例の回路図からの抜粋であり、SOPC Builder シンボルを Trace Output Buffer コンポーネントに接続する部分を示します。

図 2. JTAG デバッグ・ノード



Trace Output Buffer コンポーネントをハードウェア・デザインに追加するには、次の手順に従います。

1. `<working_directory>/mictor` ディレクトリから次の 2 つのファイルを Quartus II プロジェクト・ディレクトリにコピーします。
  - `traceoutputbuffer.bsf`
  - `traceoutputbuffer.vhd`
2. Quartus II ソフトウェアで、ハードウェア・デザインにおける回路図を開きます。
3. Trace Output Buffer コンポーネントを追加するには、回路図を右クリックし、**Insert** をポイントして、**Symbol** をクリックします。
4. **Project** フォルダ内の `traceoutputbuffer` コンポーネントを選択します。
5. 表 2 および図 2 に示すように、回路図で Trace Output Buffer コンポーネント信号と SOPC Builder シンボル上の信号を接続します。
6. `tag_debug_trigout_from_the_cpu` ピンは Trace Output Buffer コンポーネントに接続されません。`jtag_debug_trigout_from_the_cpu` 信号を FPGA 出力ピンに接続します。このハードウェア・デザイン例では、`jtag_debug_output` は `tr_triga` と呼ばれるピンに接続されます。
7. Quartus II プロジェクトをコンパイルします。

## ハードウェア・デザインに RBF ファイルを作成

TRACE32 PowerView IDE を使用してハードウェア・デザインを FPGA に実装するには、`.rbf` (Raw Binary Format) ファイルを生成する必要があります。TRACE32 IDE は `.sof` フォーマットをサポートしません。このセクションでは、`.rbf` ファイルを生成する 2 つの方法について説明します。

### .rbf ファイルを直接生成

デザインがまだコンパイルされていない場合、.rbf ファイルを作成する最も簡単な方法は Quartus II ソフトウェアによって直接生成することです。.rbf ファイルをコンパイル・プロセスの結果として生成するように Quartus II ソフトウェアをコンフィギュレーションするには、次のステップを実行します。

1. Quartus II ソフトウェアで、Assignments メニューから **Device** をクリックして **Settings** ダイアログ・ボックスを開きます。
2. **Device** ページで、**Device and Pin Options** をクリックします。
3. **Programming Files** タブをクリックします。
4. **Raw Binary File** をオンにします。
5. **OK** をクリックします。
6. もう一度 **OK** をクリックします。
7. デザインをコンパイルします。

### .sof ファイルを .rbf ファイルに変換

すでに .sof ファイルを生成した場合、次の手順で .sof ファイルを .rbf フォーマットに変換することができます。

1. Quartus II ソフトウェアで、File メニューから **Convert Programming Files** をクリックします。
2. **Output programming file** で、**Programming file type** に **Raw Binary File (.rbf)** を選択します。
3. **File name** ボックスで、ファイル・パスおよびファイル名を入力します。
4. **Input files to convert** で、**SOF Data** を選択します。
5. **Add File** をクリックし、変換しようとする .sof ファイルを選択します。
6. その .sof 名を選択して **Properties** をクリックします。
7. **Compression** がオフになっていることを確認し、**OK** をクリックします。
8. **Generate** をクリックして、.rbf ファイルを生成します。



ハードウェア・デザイン例と共に提供される

`niosII_cycloneIII_3c120_host_board_top.sof` ファイルは、すでに `mictor_01.rbf` というファイルに変換されています。T

## ソフトウェア例の準備


このアプリケーション・ノートの特約チュートリアル・ステップでは、Lauterbach TRACE32 システムの機能を実証するためにネステッド・ループのソフトウェア例を実行します。このセクションでは、Nios II ソフトウェア・ビルド・ツールによってネステッド・ループの例を構築する方法について説明します。

ネステッド・ループのソフトウェア例を構築するには、次のステップを実行します。

1. Nios II コマンド・シェルを起動するには、**Start** メニューで、**Programs > Altera > Nios II EDS <version>** をポイントして、**Nios II <version> Command Shell** をクリックします。

2. ソフトウェアのデザイン・アプリケーション例のインストール先に移動するには、次のコマンドを入力します。

```
cd <working_directory>\mictor\software_examples\app\  
nested_loops↵
```

 ソフトウェア・アプリケーションのディレクトリを以降、<app\_directory>と表記します。

3. 次のコマンドを入力します。

```
./create-this-app↵
```

## TRACE32 ロジック開発システムの起動


このセクションでは、正確に Lauterbach PowerTrace に電源を投入する方法、および TRACE32 PowerView IDE の起動方法について説明します。



ターゲット・ボードに電源を投入する前に、またはターゲット・ボードを Lauterbach ハードウェアに接続する前には、常に最初に Lauterbach ハードウェアに電源を投入してください。パワーダウンの時は、常に Lauterbach ハードウェアへの電源を最後に切るようにしてください。

開始する前に、ハードウェアが次のように接続されていることを確認してください。

- PowerTrace ハードウェアがターゲット・ボードに接続されていない
- PowerTrace ハードウェアがホストに接続されている
- LA-7837 モジュールが LA-7707 PowerTrace モジュールに接続されている
- LA-3801 モジュールが LA-7707 PowerTrace モジュールに接続されている

 トレース機能を使用しない場合、LA-3801 を接続しないでください。


PowerTrace ハードウェアの接続および TRACE32 IDE の取り付けを正しい順序で実行しなければなりません。カスタム・ハードウェアをデバッグする際、デザイン例のファイル名をデザインのファイル名に置き換えた上で、このセクションでのチュートリアルを指示を実行します。




これらのステップが正しい順序で実行されていないと、Lauterbach PowerTrace ハードウェア、ターゲットのハードウェア、あるいはその両方に損傷を与える可能性があります。ステップ 8 まではターゲット・ボードに電源を入れてはいけません。

TRACE32 システムを立ち上げるには、次のステップを実行します。

1. Lauterbach ハードウェアに電源を投入します。
2. TRACE32 IDE を起動します。

 TRACE32 IDE を起動する前に、Lauterbach ハードウェアに電源を入れたことを確認してください。TRACE32 IDE は、Lauterbach ハードウェアの電源が入れたことを予期します。

3. 図 3 に示すように、CPU メニューで、**System Settings** をクリックして **B::SYStem** ダイアログ・ボックスを開きます。
4. **B::SYStem** ダイアログ・ボックスでの **Mode** の下から、**NoDebug** を選択します。
5. ターゲット・ボードが電源から切断されていることを確認します。
6. Lauterbach PowerTrace LA-7837 モジュールの 10 ピン JTAG ヘッダをターゲット・ボードに接続し、ケーブルのピン 1 がターゲット・ボード上のピン 1 に対応することを確認します。
7. Lauterbach MICTOR ケーブルを THDB-SUM ボードの MICTOR ソケットに挿入します。

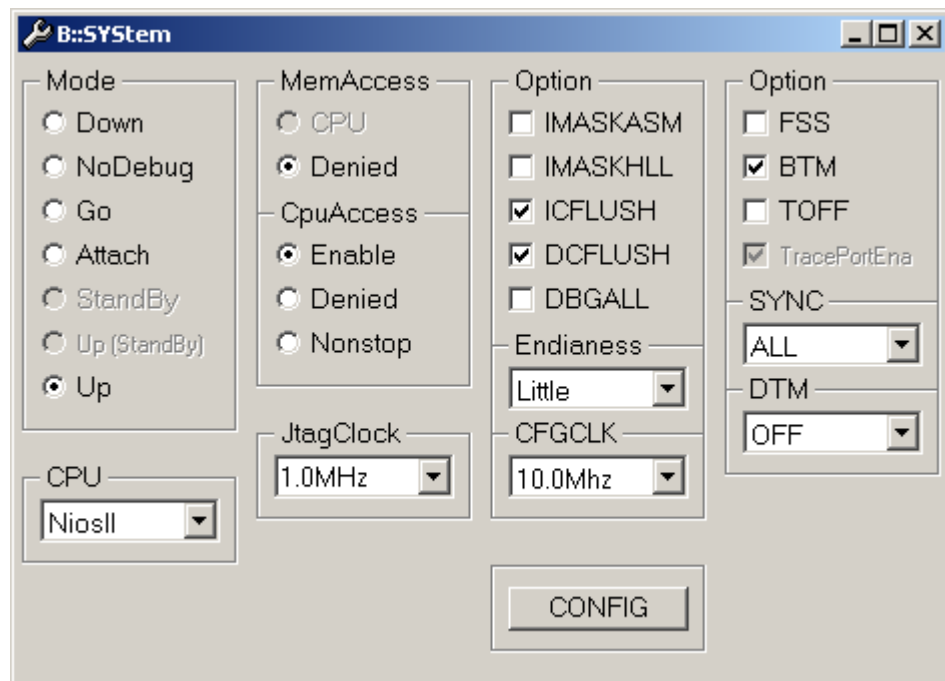
 トレース機能を使用しない場合、MICTOR ケーブルを接続しないでください。




ターゲット・ボードに電源を投入する前、またはターゲット・ボードを Lauterbach ハードウェアに接続する前には、常に最初に Lauterbach ハードウェアに電源を投入するようにしてください。

8. ターゲット・ボードの電源をオンにします。running というメッセージが TRACE32 GUI の下側に現れます。
9. **B::SYStem** ダイアログ・ボックスでの **Mode** の下から、**Attach** を選択します。Lauterbach TRACE32 IDE は PowerTrace ハードウェアとの通信が確立され、そしてラジオ・ボタンが **Up** に変更します (図 3 を参照)。

図 3. B::System ダイアログ・ボックス



10. **B::SYStem** ダイアログ・ボックスでの設定が表 3 に示すようにコンフィギュレーションされることを確認します。


 **CpuAccess** が **Denied** (デフォルト値) に設定されると、トレース・データは収集されません。この設定を TRACE32 IDE が起動するたびにイネーブルにする場合は、Lauterbach のインストール先ディレクトリにある **t32.cmm** スクリプト・ファイルに次のラインを追加します。

```
system.cpuaccess enable
```

表 3. **B::SYStem** ダイアログ・ボックスにおける設定

エリア	設定名	設定値
<b>CpuAccess</b>	<b>Enable</b>	選択される
<b>Option</b>	<b>ICFLUSH</b>	オン
	<b>DCFLUSH</b>	オン
	<b>FSS</b>	オフ
	<b>BTM</b>	オン
	<b>TOFF</b>	オフ

11. TRACE32 画面の下側には、**B::** として表示されるコマンドラインがあります。このプロンプトに **area** を入力します。このコマンドにより、**B::Area** ウィンドウが開かれます。このウィンドウでは後続のコマンドによるテキスト出力が表示されます。

 **area** コマンドを実行する時に次のメッセージが表示される場合は、ターゲット・ボードが正しくコンフィギュレーションされていないことを表します。

```
no debugging device attached
**** Unknown SLD HUB. Can't find Nios II nodes!
**** Are you sure that the FPGA is configured correctly?
```


このメッセージが表示された場合、ページ 5 の「Lauterbach によるデバッグ作業のために Cyclone III 開発ボードを準備」で述べたとおりに、ボード・コンフィギュレーションをチェックします。


12. 次のコマンドによって、FPGA 内の **.rbf** ファイルをプログラムします。

```
jtag.loadrbf <working_directory>\mictor\mictor_01.rbf←
```

次のメッセージが表示されます。

```
FPGA configuration start
FPGA is from the Cyclone III family
FPGA configuration finished
```

 target processor in reset というメッセージが表示される場合は、TRACE32 IDE と PowerTrace ハードウェアの間の通信が確立されていないことを示します。**B::SYStem** ダイアログ・ボックスでの **Mode** の下から、**Attach** を選択し、ラジオ・ボタンが **Up** になるまで待ちます (図 3 を参照)。

access timeout processor running というメッセージが表示される場合は、ターゲットが停止されないことを表します。**Break**  をクリックして、jtag.loadrbf コマンドをもう一度入力します。

13. ソフトウェアの読み込みを準備するために、ツール・バーで、**Break** をクリックしてターゲットを停止させます。メッセージ・ラインでの running メッセージは stopped メッセージに置き換えられます。

14. 下記のとおり、data.load.elf を使用して、**nested\_loops.elf** をロードします。

```
data.load.elf <app_directory>\nested_loops.elf /PLUSVM /StripPART 3 /PATH C:\ /stabs
```


この data.load.elf コマンドの例では、次のコマンドライン引数を使用されません。

- /PLUSVM— コードをホスト上およびターゲット上の仮想メモリにロードします。完全なトレース・リスティングに対しては、コードをホスト上にロードしなければなりません。
- /StripPART 3— ソース・コードのパス名を、Lauterbach TRACE32 IDE に互換性があるようにします。GNU ツールは、Cygwin スタイルのパス名によって、.elf (executable and linking format) ファイルを構築します。ただし、TRACE32 IDE は、Windows スタイルのパスを通して C ソース・コードにアクセスすることを予期します。TRACE32 IDE が .elf ファイルのソース・コードを特定できるように、/StripPART スイッチは .elf ファイルのパス名を Windows 互換のパス名に変換します。
- /PATH C:\—TRACE32 IDE のルートパスを定義して、ソース・コードの検索を開始します。/PATH 引数を使用して複数の検索パスを指定することができます。また、追加のパスを検索するには、View メニューで、**Symbol** をポイントして、**Source Search Paths** をクリックします。
- /stabs— デバッグ情報が STABS フォーマットであることを指定します。

15. .elf ファイルが成功にロードされた後、以下のメッセージが B::area ウィンドウで表示されます。

```
file '<app_directory>\nested_loops.elf' (ELF) loaded
```

16. **Go** をクリックし、そして **Break** をクリックします。Nios II プログラムは実行を開始し、初期化コードを実行し、そして **nested\_loops.c** アプリケーション例の本文でブレークします。

 **nested\_loops.c** のソース・コードを表示するには、ページ 18 の「プロセッサ使用状況データの表示」を参照してください。

## TRACE32 PowerView IDE によるシステム・ステートの表示

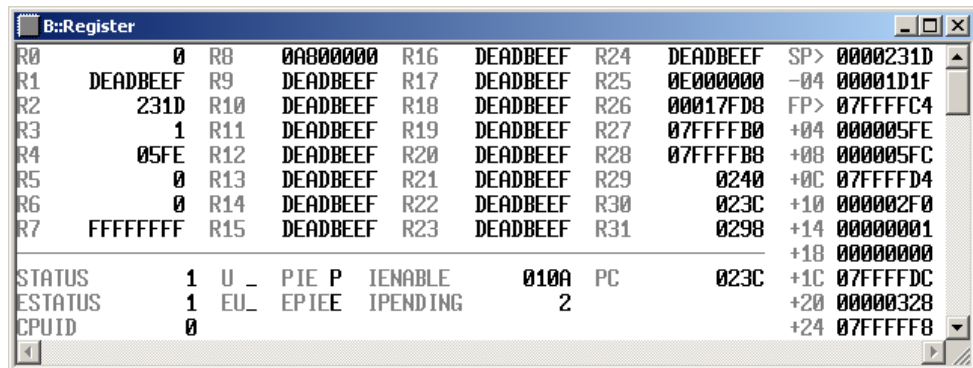
.elf ファイルをロードした後、TRACE32 PowerView IDE は Nios II システムの現在の状態を検証することができます。次のセクションでは、システム状態を検証するいくつかの方法について説明します。

 システム状態を表示する前に、**Break** をクリックしなければなりません。

### Nios II レジスタの表示

Nios II レジスタを表示するには、View メニューで、**Resiters** をクリックします。図 4 に、デザイン例でセットアップされている Nios II レジスタを示します。


図 4. Register ウィンドウ



R0	0	R8	0A800000	R16	DEADBEEF	R24	DEADBEEF	SP>	0000231D
R1	DEADBEEF	R9	DEADBEEF	R17	DEADBEEF	R25	0E000000	-04	00001D1F
R2	231D	R10	DEADBEEF	R18	DEADBEEF	R26	00017FD8	FP>	07FFFC4
R3	1	R11	DEADBEEF	R19	DEADBEEF	R27	07FFFFB0	+04	00005FE
R4	05FE	R12	DEADBEEF	R20	DEADBEEF	R28	07FFFFB8	+08	00005FC
R5	0	R13	DEADBEEF	R21	DEADBEEF	R29	0240	+0C	07FFFD4
R6	0	R14	DEADBEEF	R22	DEADBEEF	R30	023C	+10	00002F0
R7	FFFFFFFF	R15	DEADBEEF	R23	DEADBEEF	R31	0298	+14	0000001
								+18	0000000
STATUS	1	U_	PIE P	IENABLE	010A	PC	023C	+1C	07FFFD4
ESTATUS	1	EU_	EPIEE	IPENDING	2			+20	0000328
CPUID	0							+24	07FFFF8

## Nios II ソース・コードの表示

ソース・コードを表示するには、View メニューで、**List Source** をクリックします。B::Data.List ウィンドウが現れ、アプリケーションのソース・コードを表示します。

 **Mode** をクリックすることで、アセンブリ言語の表示と非表示を切り替えることができます。

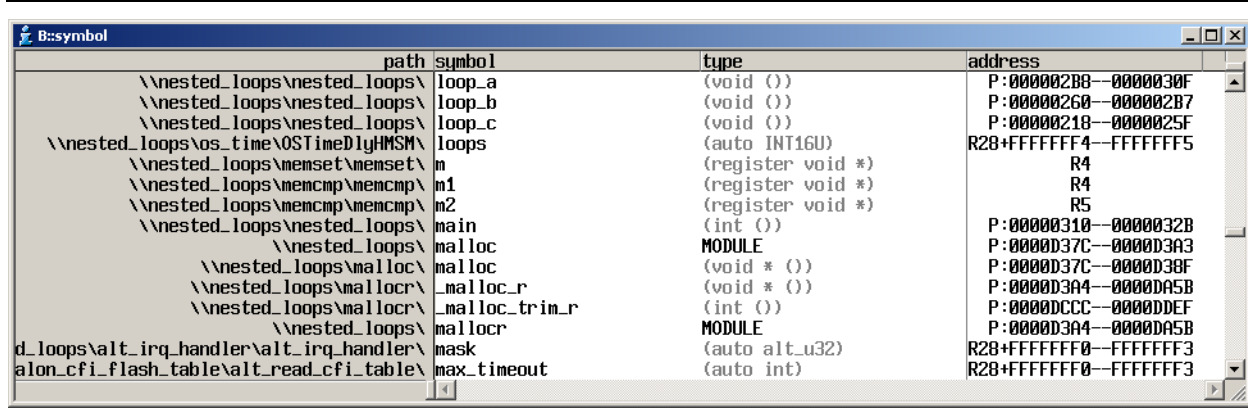
## シンボルの検索

View メニューで、**Symbols** をポイントして **Browse** をクリックすることで、プログラムで定義されているすべてのシンボルを表示することができます。シンボルを定義するソース・ファイルを開くには、そのシンボル名をクリックします。

## シンボルの表示

C ソース・コードで定義されているシンボルを表示するには、B:: プロンプトでそのシンボル・コマンドを入力します。図 5 に、**nested\_loop.c** で定義されたシンボルを示します。

図 5. シンボル定義のウィンドウ



path	symbol	type	address
\\nested_loops\\nested_loops\\	loop_a	(void ())	P:000002B8--0000030F
\\nested_loops\\nested_loops\\	loop_b	(void ())	P:00000260--000002B7
\\nested_loops\\nested_loops\\	loop_c	(void ())	P:00000218--0000025F
\\nested_loops\\os_time\\OSTimeDlyHMSM\\	loops	(auto INT16U)	R28+FFFFFFFF4--FFFFFFFF5
\\nested_loops\\memset\\memset\\	m	(register void *)	R4
\\nested_loops\\memcpy\\memcpy\\	m1	(register void *)	R4
\\nested_loops\\memcpy\\memcpy\\	m2	(register void *)	R5
\\nested_loops\\nested_loops\\	main	(int ())	P:00000310--0000032B
\\nested_loops\\	malloc	MODULE	P:0000037C--000003A3
\\nested_loops\\malloc\\	malloc	(void * ())	P:0000037C--0000038F
\\nested_loops\\mallocr\\	_malloc_r	(void * ())	P:000003A4--00000A5B
\\nested_loops\\mallocr\\	_malloc_trim_r	(int ())	P:00000DCC--00000DEF
\\nested_loops\\	mallocr	MODULE	P:000003A4--00000A5B
d_loops\\alt_irq_handler\\alt_irq_handler\\	mask	(auto alt_u32)	R28+FFFFFFFF0--FFFFFFFF3
alon_cfi_flash_table\\alt_read_cfi_table\\	max_timeout	(auto int)	R28+FFFFFFFF0--FFFFFFFF3

## ブレークポイントおよびウォッチポイントの設定

B::Data.List ウィンドウでは、ブレークポイントおよびウォッチポイントの設定ができます。例えば、変数 `count_c` への書き込みを監視するには、次のステップを実行します。

1. 「Nios II ソース・コードの表示」で述べたとおり、B::Data.List ウィンドウを開きます。
2. `loop_c()` 関数の最上部にスクロールします。
3. 変数 `count_c` をクリックして、選択します。
4. 図 6 に示すように、`count_c` を右クリックし、**Breakpoints** をポイントし、**Write** をクリックします。


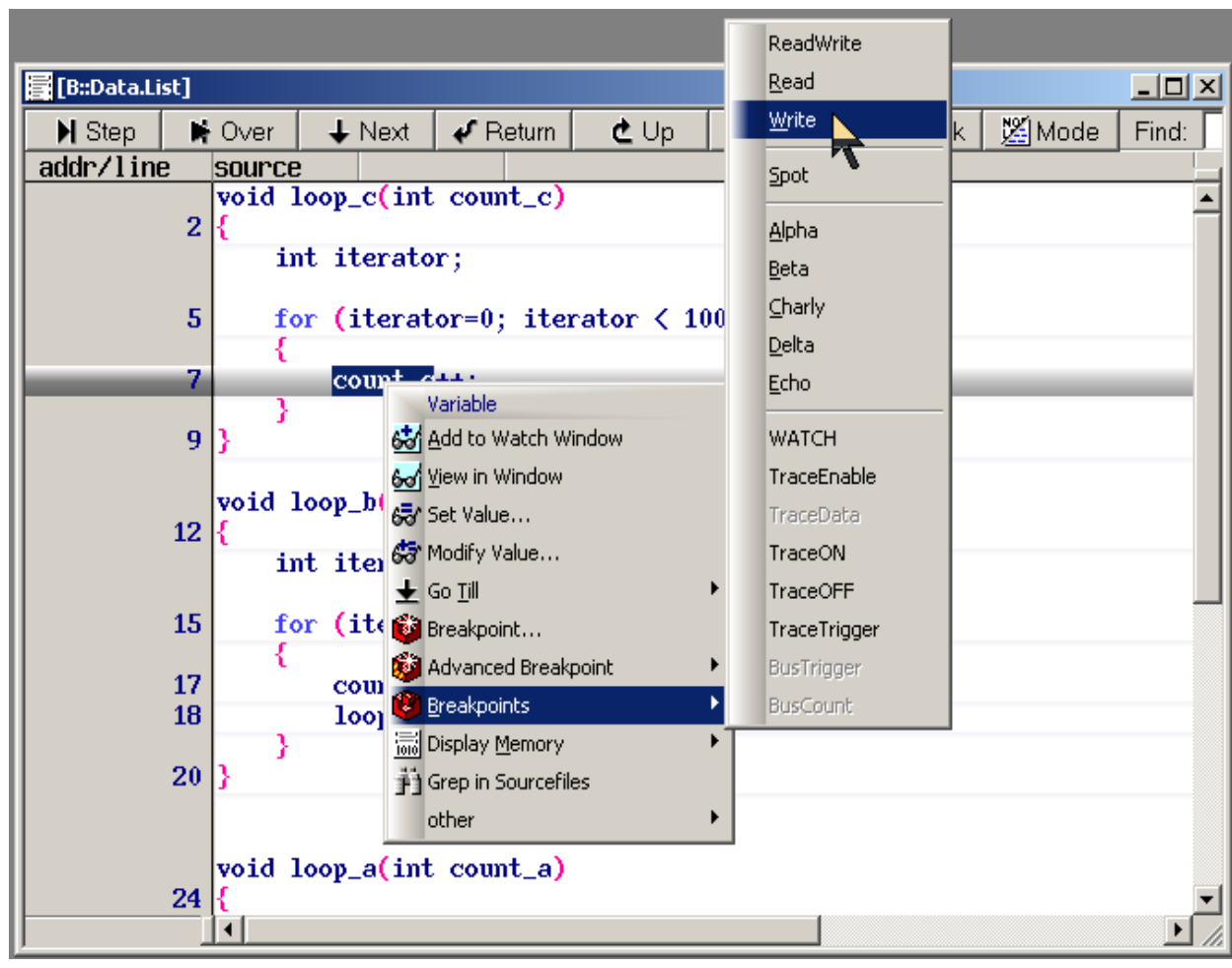

 Break メニューでは、**List** をクリックしてブレークポイントの一覧を表示することができます。

図 6. Write ブレークポイント



5. **Go** をクリックします。count\_c 変数にデータが書き込まれるたびに、デバッガは実行を一時停止します。

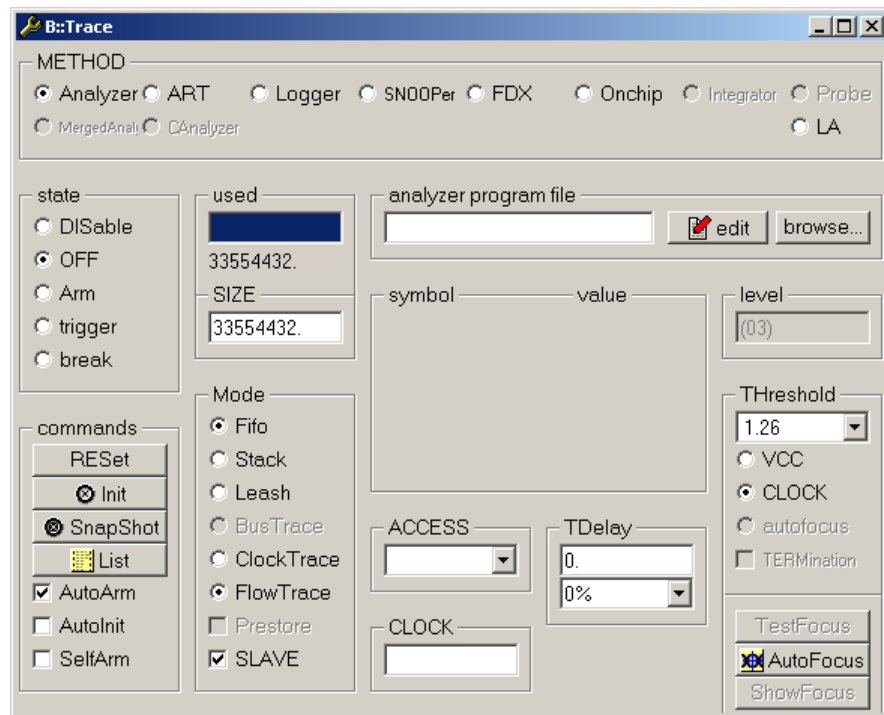
 View メニューでは、**Locals** をクリックして count\_c の値を表示することができます。

## トレースの使用

Nios II 命令の実行をキャプチャする前に、次のステップによって TRACE32 PowerView IDE のトレース・オプションをコンフィギュレーションしなければなりません。

1. Trace メニューで、**Configuration** をクリックして、**B::Trace** ダイアログ・ボックスを表示します (図 7 を参照)。


図 7. Trace コンフィギュレーション・ダイアログ・ボックス




2. 次の設定を確認します。

- **METHOD** で、**Analyzer** が選択されることを確認
- **THreshold** がターゲットに応じて適切なレベルに設定されることを確認します。このデザイン例では、**1.26V** は適切なレベルです。カスタム・デザインの場合、適切なレベルはデザインによって異なります。

3. **RESet** および **Init** をクリックして、トレース・バッファをクリアします。

4. TRACE32 ツール・バーでの **Go**  をクリックし、トレース・データの収集を開始します。**Trace** コンフィギュレーション・ダイアログ・ボックスでの **used** バーが満タンになります。

 **used** バーが満タンにならない場合は、次のいずれかの原因をチェックします。

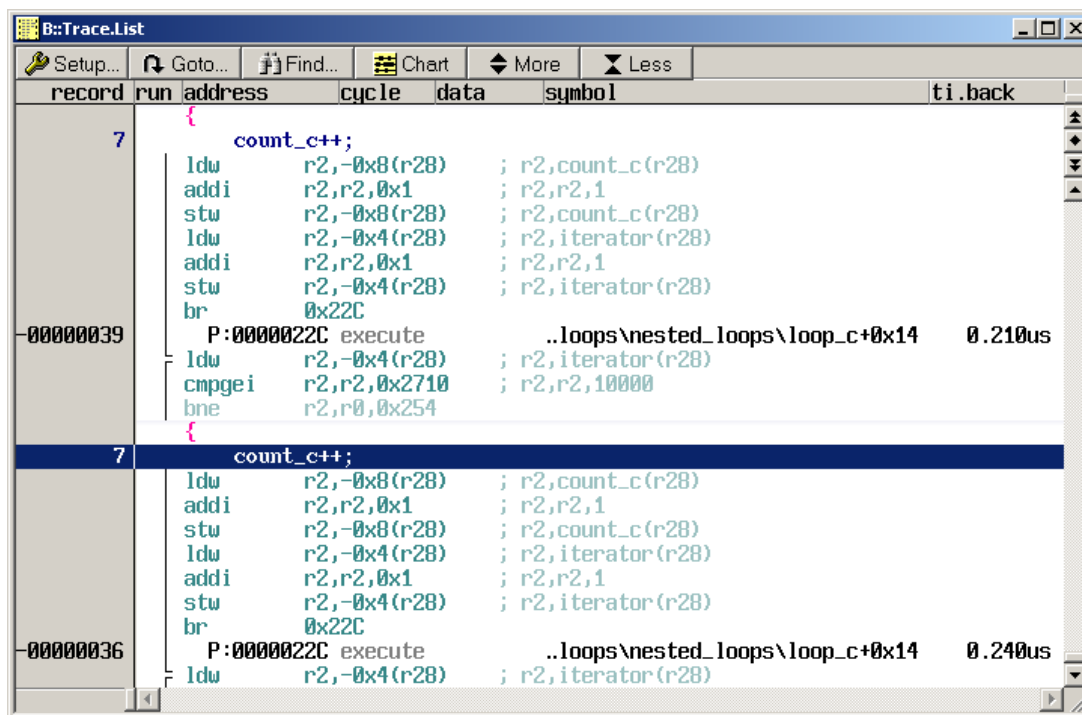
- ブレークポイントまたはウォッチポイントが設定されています。
- ハードウェア・デザインで、トレース・クロックが正しく接続されていません。
- トレース・クロックが正しくサンプルされていません。有効なトレース・クロックが生成されること、および電圧のしきい値が適切であることを確認します。

トレース・データを表示する前に、**Break** をクリックする必要があります。

## コード・トレースの表示

トレースによってキャプチャされた NiosII 命令を表示するには、[図 8](#) に示すように、Trace メニューで **List** をポイントして **Default** をクリックします。Nios II アセンブラの命令は、対応する C ソース・コードに散在します。

図 8. コード・トレースの一覧




## プロセッサ使用状況データの表示

関数ごとに使用された時間をグラフィカルな形式で表示するには、Trace メニューで、**Chart** をポイントして、**Symbols** をクリックします。例については、[図 9](#) を参照してください。

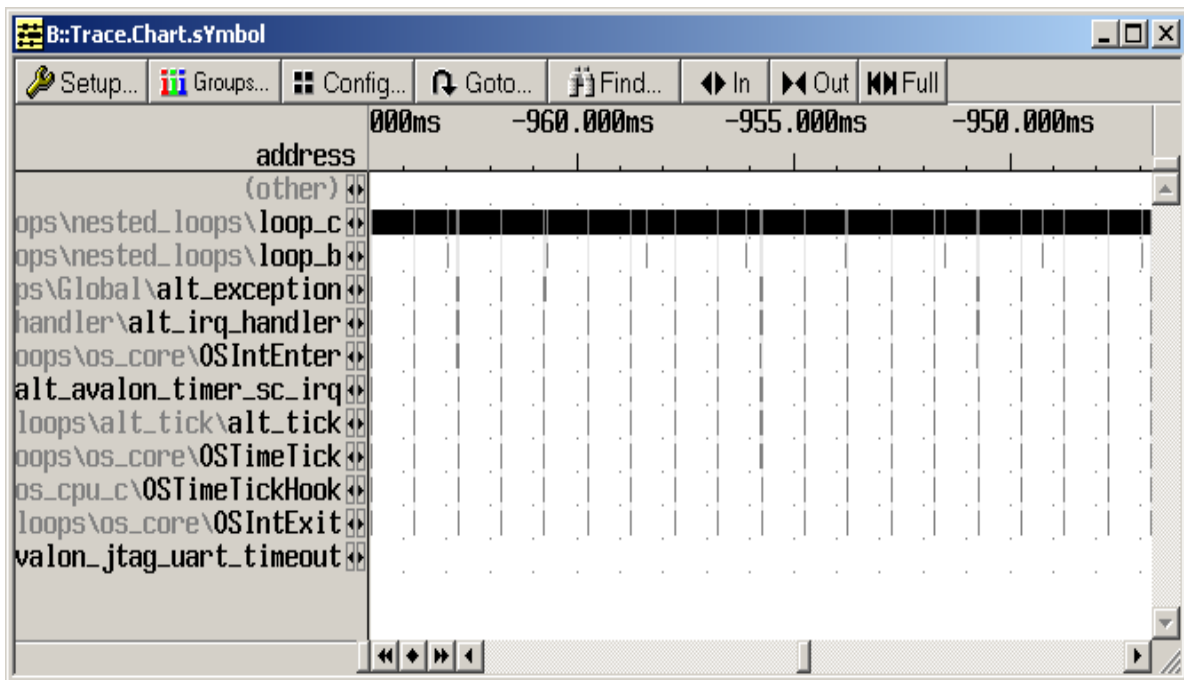
下記のようにスコープおよびデータの詳細をコントロールすることができます。

- **In** をクリックして画面を拡大して、より詳細なビューが見られます。
- **Out** をクリックして画面を縮小して、より広い範囲が見られます。

 データをキャプチャした後に初めてプロセッサ使用状況を表示する際、TRACE32 IDE はグラフをレンダリングするために、少し時間がかかります。

[図 9](#) に、ネストド・ループのアプリケーションからキャプチャされたトレース・データ・セットを示します。表内の各行は、表の左側に記載された各関数を実行するためにかかった時間を表します。x 軸は、アプリケーションの実行タイムラインです。

図 9. 拡張されたプロセッサ使用状況グラフ



ネステッド・ループ・アプリケーションは 3 つのループから構成されています。

- `loop_a()`、最も外側のループであり、`loop_b()` を 10,000 回呼び出す
- `loop_b()`、`loop_c()` を 10,000 回呼び出す
- `loop_c()`、最も内側のループであり、変数 `count_c` を 10,000 回増やす

例 1 に、このプログラム例 (`nested_loops.c`) のソース・コードを示します。Nios II プロセッサは、ほとんどの時間を `loop_c()` に費やしました。

#### 例 1. `nested_loops.c` のソース・コード ( 1 / 2 )

```

/* Source File: nested_loops.c
   Description:  Creates a simple illustration of processor usage
                 data for display with the Lauterbach TRACE32 IDE. */

void loop_c(int count_c)
{
    int iterator;
    for (iterator=0; iterator < 10000; iterator++)
    {
        count_c++;
    }
}

/* Continued... */

```

**例 1. nested\_loops.c のソース・コード ( 2 / 2 )**

```
void loop_b(int count_b)
{
    int iterator;
    for (iterator=0; iterator < 10000; iterator++)
    {
        count_b++;
        loop_c(count_b);
    }
}

void loop_a(int count_a)
{
    int iterator;
    for (iterator=0; iterator < 10000; iterator++)
    {
        count_a++;
        loop_b(count_a);
    }
}

int main()
{
    static int count_main = 0;
    loop_a(count_main);
    while(1);
    return 0;
}
```

図 9 で、`loop_c()` に費やした時間はほぼ満タンのバーとして表示されています。割り込みや `loop_b()` への時折の帰還は、実行タイムライン内の周期的な細かいラインとして表示されています。このデータ・セットをキャプチャするときに `loop_b()` が帰還していなかったため、`loop_a()` が表示されていません。

図 10 は、図 9 と同様なトレース・データの詳細ビューです。この詳細ビューは、データ・セットを拡大表示することで実現されます。このビューに、アプリケーション例におけるタイマー割り込みおよび `loop_c()` への帰還を処理するのに使用された時間を示します。

## 特別なハードウェア手法

このセクションでは、Lauterbach TRACE32 システム によるデバッグ作業を準備するために時々必要とされるメンテナンス手法および診断手法について説明します。

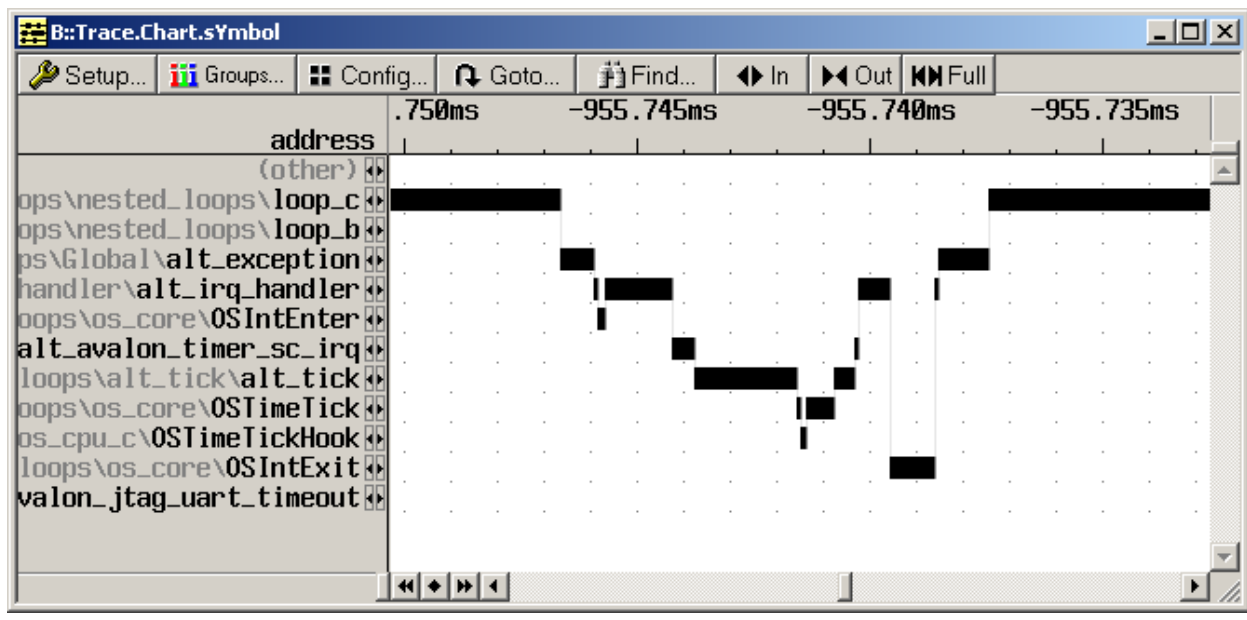
### ターゲット・ボードの JTAG 接続をホット・スワップ

JTAG コネクタを、USB-Blaster™ などのほかのデバイスにホット・スワップすることが可能です。ただし、これを安全に実行するには、このセクションで述べたガイドラインに従う必要があります。




Lauterbach ハードウェアを電源の入れたターゲット・ボードに接続する前、または接続を切る前には、常に TRACE32 IDE で **System Settings** モードを **NoDebug** に設定することが重要です。モードを **NoDebug** に設定しなければ、ハードウェアに損傷を与える恐れがあります。**System Settings** モードを設定するには、CPU メニューで、**System Settings** をクリックして **B::SYStem** ダイアログ・ボックスを開いて、そして **Mode** の下から **NoDebug** を選択します。

図 10. 詳細なプロセッサ使用状況グラフ



下記のいずれかの理由により、Lauterbach JTAG コネクタをほかのデバイスにホット・スワップする必要がある可能性があります。

- JTAG 接続の問題のトラブルシューティング。例えば、Lauterbach LA-7837 デバッガをほかの JTAG ダウンロード・ケーブルに交換する必要がある場合があります。
- 電源を切断せずにターゲット・ボードを外したり、または再接続する場合。例えば、Quartus II Programmer を USB-Blaster と一緒にして、.sof ファイルによって FPGA をプログラムする場合、.sof イメージをメモリに維持するためには、ターゲット・ボードの電源を入れたままにする必要があります。


 Quartus II ソフトウェアに接続している場合は、.sof ファイルを .rbf ファイルに変換することで、ホット・スワップを避けるほうが安全です。ハードウェア・デザインが .rbf ファイルである場合、TRACE32 IDE を使用して、.rbf ファイルを Lauterbach JTAG 接続を通して FPGA にプログラムすることができます。.rbf ファイルを作成する手順については、ページ 8 の「ハードウェア・デザインに RBF ファイルを作成」を参照してください。

デバッグを再開する前に、11 ページでの「TRACE32 ロジック開発システムの起動」のステップ 9 で説明したとおり **Attach** モードを選択する必要があります。

## ボードのファクトリ・デフォルト設定の復元

MAX II デバイスでハードウェア・デザインを更新する場合、あるいはフラッシュ・オプション・ビットを更新する場合は、まずは Cyclone III 開発ボード上のスイッチとジャンパを工場出荷時のデフォルト設定に戻す必要があります。開発ボード上のスイッチとジャンパの工場出荷時デフォルト設定は以下のとおりです。

- FPGA BYPASS というラベルの付いた、SW3 の位置 1 が 1 に設定される
- SW3 上のほかの位置がすべて 0 に設定される
- ジャンパ J6 (DEV\_SEL) が装着されている
- ジャンパ J7 (JTAG\_SEL) が装着されている
- MAX0 というラベルの付いた、SW1 の位置 5 が 1 (OPEN) に設定される

 Lauterbach TRACE32 システムによってデバッグを再開する前には、Cyclone III 開発ボードのジャンパおよびスイッチを、ページ 5 の「Lauterbach によるデバッグ作業のために Cyclone III 開発ボードを準備」で説明したとおりデバッグ用のコンフィギュレーションに戻します。

 Cyclone III 開発ボードのコンフィギュレーションについて詳しくは、「Cyclone III 3C120 Development Board Reference Manual」を参照してください。

## Cyclone III 開発ボードにおける MAX II デザインの更新

Cyclone III 3C120 開発ボードを Lauterbach PowerTrace ハードウェアに使用前に、Max II デバイスにプログラムされたデザインを更新する必要がある可能性があります。この更新は 1 回だけで十分です。

Max II デザインのどのビルドがインストールされているのを確認するには、ボードに電源を入れて CPU RESET (LCD ディスプレイの下にある) を押し続けます。POWER DISPLAY というラベルの付いた 7 セグメント・ディスプレイに、ビルド番号が表示されます。表示されたビルド番号が 1338 以上であれば、Max II デザインのバージョンは Lauterbach PowerTrace ハードウェアに対応します。そうでない場合、それは古いバージョンの Max II デザインです。次に進む前に、デザインを更新しなければなりません。

Max II デザインを更新するには、次のものがが必要です。


- Max II デザイン・ファイルの **m2\_prod\_1338.pof**。このファイルは **lauterbach.zip** に含まれています。
- アルテラの USB-Blaster



Cyclone III 開発ボードに内蔵された JTAG プログラマによって MAX II デザインを更新してはいけません。内蔵された JTAG プログラマ回路の一部は Max II CPLD に実装されています。

Max II デザインを更新するには、次のステップを実行します。

1. Cyclone III 開発ボードから電源を切断します。
2. スイッチおよびジャンパが「**ボードのファクトリ・デフォルト設定の復元**」に示したように、工場出荷時のデフォルト設定に設定されていることを確認します。
3. USB-Blaster を、ボードの側面にある 10 ピン JTAG ポートによってボードに接続します。ケーブルのピン 1 がターゲット・ボードのピン 1 に対応することを確認します。
4. Cyclone III 開発ボードに電源を投入します。
5. Quartus II Tools メニューで、**Programmer** をクリックして Quartus II プログラマを実行します。
6. **Mode** で **JTAG** が選択されていることを確認します。
7. **Hardware Setup** をクリックして、そして Cyclone III ボードに接続されている USB-Blaster が選択されたことを確認します。
8. **Auto Detect** をクリックします。Quartus II プログラマは **EPM2210** を検出して、リストします。
9. EPM2210 デバイスに対して、**<none>** をダブルクリックします。
10. **<working\_directory>/board\_config/m2\_prod\_1338.pof** に移動し、そのファイルを選択し、そして **Open** をクリックします。
11. EPM2210 デバイスに対して、**Program / Configure** オプションをオンにします。
12. **Start** をクリックします。
13. プログラミングが完了したら、Max II デザインが正しいかどうかをチェックします。CPU RESET ボタンを押し続けます。POWER DISPLAY に、1338 が表示されています。

 Lauterbach TRACE32 システムによってデバッグを再開する前には、Cyclone III 開発ボードのジャンパおよびスイッチを、ページ 5 の「Lauterbach によるデバッグ作業のために Cyclone III 開発ボードを準備」で説明したとおりデバッグ用のコンフィギュレーションに戻します。

## Cyclone III 開発ボード上のフラッシュ・オプション・ビットの更新

Cyclone III 3C120 開発ボードを Lauterbach PowerTrace ハードウェアに使用する前に、フラッシュ・オプション・ビットを更新して、MAX II に実装されたパラレル・フラッシュ・プログラミングをイネーブルにする必要がある可能性があります。この更新は 1 回だけで十分です。

フラッシュ・オプション・ビットを更新するには、次のものがが必要です。


- Jam Byte-Code ファイルの **3C120\_flash\_option\_bits.jbc**。このファイルは **lauterbach.zip** に含まれています。
- アルテラの USB-Blaster

フラッシュ・オプション・ビットを更新するには、次のステップを実行します。

1. Cyclone III 開発ボードから電源を切断します。
2. スイッチとジャンパが、「ボードのファクトリ・デフォルト設定の復元」に示すように工場出荷時の設定に設定されていることを確認します。
3. USB-Blaster を、ボードの側面にある 10 ピン JTAG ポートによってボードに接続します。ケーブルのピン 1 がターゲット・ボードのピン 1 に対応することを確認します。
4. Cyclone III 開発ボードに電源を投入します。
5. Nios II コマンド・シェルを起動するには、Start メニューで、**Programs > Altera > Nios II EDS <version>** をポイントして、**Nios II <version> Command Shell** をクリックします。
6. 次のコマンドを入力して、Quartus II JBI Player でフラッシュ・オプション・ビットをプログラムします。

```
quartus-jli <working_directory>/board_config/flash_option_bits.jbc -a CONFIGURE
quartus-jli <working_directory>/board_config/flash_option_bits.jbc -a PROGRAM
```

Quartus II JBI Player によるフラッシュのプログラミングが完了するまで待ちます。このプロセスは数分かかることがあります。

 Lauterbach TRACE32 システムによってデバッグを再開する前には、Cyclone III 開発ボードのジャンパおよびスイッチを、ページ 5 の「Lauterbach によるデバッグ作業のために Cyclone III 開発ボードを準備」で説明したとおりデバッグ用のコンフィギュレーションに戻します。

## OCI コア通信のトラブルシューティング

Nios II OCI (On-Chip Instrumentation) コアに問題が発生すると、Lauterbach TRACE32 の **System.Attach** (または **System.Up**) コマンドは `monitor not responding` というメッセージで失敗することがあります。このメッセージは、TRACE32 システムと Nios II OCI 間の通信問題のいずれかを表します。デバッグ・スクリプト `lauterbach_system_up_debug_script.cmm` は問題の原因を確認するのに役立ちます。

`lauterbach_system_up_debug_script.cmm` スクリプトは、`<working_directory>\troubleshooting` にある `lauterbach.zip` ファイルに含まれています。このスクリプトを使用するには、次の手順を実行します。

1. `lauterbach_system_up_debug_script.cmm` ファイルを Lauterbach のインストール先ディレクトリにコピーします。
2. TRACE32 IDE が実行されていない場合、それを起動します。
3. File メニューで、**Run Batchfile** をクリックし、そして `lauterbach_system_up_debug_script.cmm` スクリプトをダブルクリックします。

スクリプトが次のファイルを生成します。これらのファイルは障害の解析に有用な診断情報を提供します。

- **debug.log**— デバッグ・スクリプトに実行された各コマンドの結果です。
- **debug.lst**— デバッグ・スクリプトにおけるトレースの部分によって生成されます。システムが正常に動作している場合、**debug.lst** には生データおよびアップカウント・タイムスタンプが含まれています。

`<working_directory>\troubleshooting` には、**debug.log** および **debug.lst** の例があります。

- **good\_debug.log** および **good\_debug.lst**— MICTOR ドーター・カードが付いた Cyclone III Nios II Embedded Evaluation Kit (NEEK) ボード上における `lauterbach_system_up_debug_script.cmm` の成功の呼び出しにより生成されます。
- **failed\_debug.log**— カスタム・ボードへの失敗した Lauterbach 接続により生成されます。

**debug.log** および **debug.lst** をこれらの例に比較して、通信エラーの原因を特定します。



**debug.log** および **debug.lst** にあるテスト結果の解釈の支援については、[www.lauterbach.com](http://www.lauterbach.com) から Lauterbach 社のテクニカル・サポートにお問い合わせください。

## 参考資料

このアプリケーション・ノートでは、以下のドキュメントを参照しています。:

- [Nios II Development Kit Getting Started User Guide](#)
- [Nios II Hardware Development Tutorial](#)
- Nios II ソフトウェア開発チュートリアル。このチュートリアルにアクセスするには、Nios II IDE のウェルカム・ページで **Tutorials** をクリックします。
- TRACE32 PowerView IDE と共にインストールされる資料。TRACE32 ロジック開発システムに関する資料は [www.lauterbach.com](http://www.lauterbach.com) から入手できます。次の資料が特に重要です。
  - [Nios II Debugger and Trace \(debugger\\_nios.pdf\)](#)
  - [Nios II Instantiating the Off-chip Trace Logic \(app\\_nios.pdf\)](#)— 複数の Nios II プロセッサ・コアからのトレース出力の多重化に関する情報です。
- 👉 「[Nios II Instantiating the Off-chip Trace Logic](#)」では、「Disable automatic PLL instantiation」で説明するステップを無視にしてください。6 ページのステップ 4 は同じタスクを実行しています。
- [Santa Cruz, USB, Mictor, SD Card HSMC Reference Manual](#)
- [Cyclone III 3C120 Development Board Reference Manual](#)

## 改訂履歴

表 4 に、このアプリケーション・ノートの改訂履歴を示します。

表 4. 改訂履歴

日付およびリビジョン	変更内容	概要
2009 年 4 月 v1.0	初版	—



101 Innovation Drive  
San Jose, CA 95134  
[www.altera.com](http://www.altera.com)  
Technical Support  
[www.altera.com/support](http://www.altera.com/support)

Copyright © 2009 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before

