AN543: Debugging Nios II Software Using the Lauterbach Debugger

© April 2009 AN-543-1.0

Introduction

This application note presents methods of debugging a Nios® II application with the Lauterbach TRACE32 Logic Development System.

The TRACE32 system, including Lauterbach PowerTrace hardware and the TRACE32 PowerView integrated development environment (IDE), provides complete visibility into the operation of a Nios II system. In combination with the Nios II EDS, SOPC Builder, and the Quartus® II software, the TRACE32 system enables you to analyze Nios II system failures or anomalous design behavior. The Lauterbach TRACE32 system gives you a degree of control unmatched by other Nios II debugging environments.

This application note includes tutorial steps for debugging an example design on the Altera® Cyclone® III 3C120 development board, and general guidelines for debugging a custom design.

This application note assumes that you are running the Lauterbach TRACE32 PowerView IDE on a Windows platform. However, the techniques presented are independent of the platform.

For detailed instructions for the Lauterbach TRACE32 Logic Development System, refer to www.lauterbach.com.

The Diagnostic Power of the Lauterbach Tools

The Lauterbach TRACE32 hardware and software tools provide an extremely powerful set of features for diagnosing and solving some of the toughest embedded systems development problems, including those problems involving the time domain. The Lauterbach tools enable you to accomplish the following typical debugging tasks:

- Set breakpoints and watchpoints
- Step though code
- Examine variable values

You can also employ more powerful techniques, such as:

- Watch for writes to a particular variable
- Capture and review a complete program execution trace

Execution Trace

By including a MICTOR connection to the Nios II processor in your Quartus II hardware design, you can non-intrusively capture large quantities of off-chip execution trace data through the Lauterbach PowerTrace hardware, allowing a thorough analysis of the program execution history. You can combine this feature with trigger conditions to capture Nios II execution activity leading up to and immediately following a system failure.

To examine execution trace data, you must have the Lauterbach LA-3801 Nios II trace module. Without the LA-3801, you can still use other features of the TRACE32 system, such as breakpoints and watchpoints.

Prerequisite Knowledge

This document is intended for advanced systems developers with a basic understanding of the following topics:

- Nios II application development
- The Quartus II software
- The Lauterbach TRACE32 PowerView IDE

To gain the minimum prerequisite knowledge, refer to the following documents:

- *Nios II Development Kit Getting Started User Guide*
- *Nios II Hardware Development Tutorial*
- The Nios II Software Development Tutorial, available in the Nios II IDE by clicking **Tutorials** on the Welcome page
- TRACE32 documentation installed with the TRACE32 PowerView IDE. *Nios II Debugger and Trace* (**debugger_nios.pdf**) is of particular interest.

  Documentation for the TRACE32 Logic Development System is also available at www.lauterbach.com.

Software Requirements

The following software components are required:

- Altera Complete Design Suite version 9.0 or later
- Lauterbach TRACE32 PowerView IDE for the Nios II processor v. 9.0 or later.

  After you install the TRACE32 IDE, Altera recommends that you upgrade to the latest software distribution at www.lauterbach.com.

- The **lauterbach.zip** file, available on Literature: Nios II Processor page of the Altera website, accompanying this document. The contents of **lauterbach.zip** are shown in Table 1.

  Unzip **lauterbach.zip** into a working directory. Be sure to not include any spaces in the working directory path name. The remainder of this application note refers to your working directory as <*working_directory*>.

Table 1. Contents of lauterbach.zip

Directory Name	Contents
mictor	The Nios II hardware example design for the Altera Cyclone III 3C120 development board and the Altera THDB-SUM adapter board. This hardware design includes the Trace Output Buffer component.
mictor\software_examples\app	The nested loops application in **nested_loops.c**
schematics	■ Schematics for the Altera Cyclone III 3C120 development board and the Altera THDB-SUM adapter board ■ **Mictor_Trace_Connections_HSMA_3c120.xls**, which documents the trace data pin mapping in the example design
board_config	The following board reconfiguration files: ■ **3C120_flash_option_bits.jbc** ■ **m2_prod_1338.pof** For details, refer to "Special Hardware Techniques" on page 19.
troubleshooting	Files to assist in troubleshooting communication problems between the TRACE32 system and the Nios II OCI core: ■ **lauterbach_system_up_debug_script.cmm** ■ **good_debug.log** ■ **good_debug.lst** ■ **failed_debug.log**

Hardware Requirements

This section describes the hardware requirements for debugging with the Lauterbach TRACE32 Logic Development System.

Lauterbach PowerTrace Hardware

The Lauterbach PowerTrace hardware consists of the following components:

- Lauterbach LA-7707 PowerTrace Ethernet 256 MB Universal NEXUS/debug controller
- Lauterbach LA-7837 Debugger for NIOS-II (ICD)
- Lauterbach LA-3801 Preprocessor for NIOS-II Flex Cable

If you are not using trace features, the LA-3801 is optional.

To protect the Lauterbach circuitry from damage, it is critical that you apply power to the target board and the Lauterbach PowerTrace hardware in the correct order. Do not connect or power up the Lauterbach PowerTrace hardware until you have carefully studied "Starting the TRACE32 Logic Development System" on page 10.

Target Hardware

To support the Lauterbach TRACE32 debugging system, your target board must meet the following requirements:

- The FPGA device supports the Nios II processor.

- The board provides a 10-pin JTAG header.
- The board provides a MICTOR socket.

If you are not using trace features, the MICTOR socket is unnecessary.

Tutorial Target Hardware

The tutorial steps in this application note use the following hardware components:

- Altera Cyclone III 3C120 development board

For detailed information about the Cyclone III development board, refer to the *Cyclone III 3C120 Development Board Reference Manual*.

You might need to perform the hardware configuration steps described in "Updating the MAX II Design on the Cyclone III Development Board" on page 21 and "Updating the Flash Option Bits on the Cyclone III Development Board" on page 22.

- Altera THDB-SUM adapter board. The THDB-SUM connects to high-speed mezzanine connector (HSMC) Port A on the Altera Cyclone III 3C120 development board, to provide a MICTOR socket for trace data.

For detailed information about the THDB-SUM board, refer to the *Santa Cruz, USB, Mictor, SD Card HSMC Reference Manual*.

If you are not using trace features, you do not need the THDB-SUM board.

FPGA Design Requirements

Your hardware design must connect the Nios II processor to the MICTOR socket as described in "Hardware Design Preparation" on page 6.

If you are not using trace features, the MICTOR connection is unnecessary.

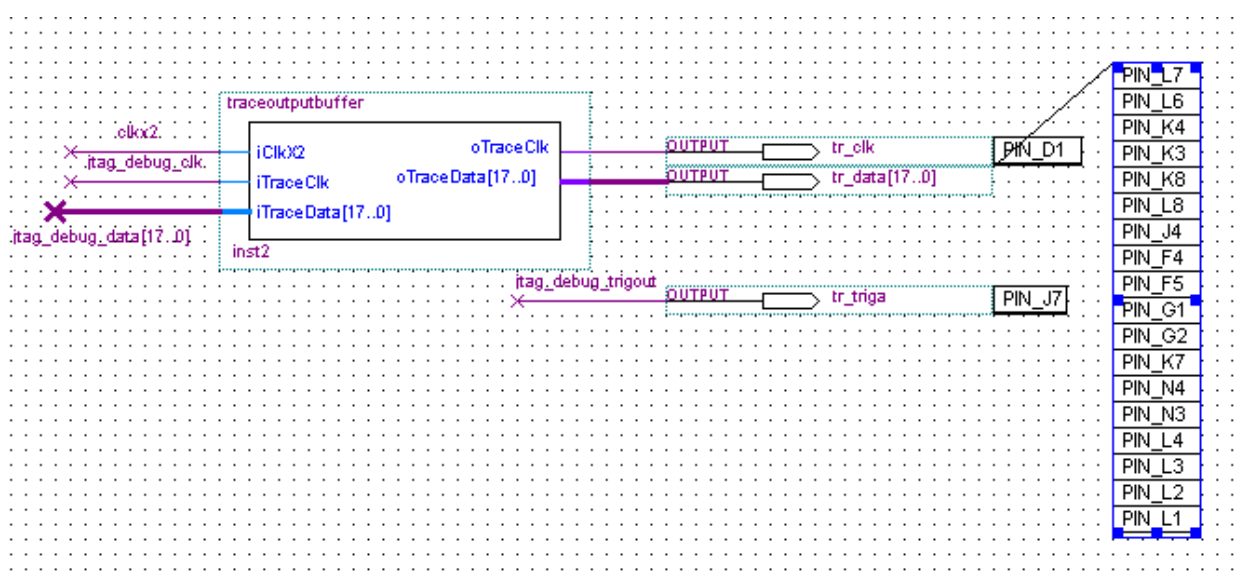
The example design uses the Cyclone III 3C120 development board with a THDB-SUM adapter board connected to the HSMC connector labeled HSMA. The THDB-SUM board provides a MICTOR socket for connection with the Lauterbach trace hardware.

The <*working_directory*>**schematics** directory contains a file named **Mictor_Trace_Connections_HSMA_3c120.xls**. This spreadsheet documents the trace data pin mapping in the example design. The spreadsheet associates the trace data pins in the FPGA design, listed in the **FPGA Logic** column, with the physical FPGA pins connected to the MICTOR, listed in the **FPGA Pin** column.

This mapping is derived from the Cyclone III development board and THDB-SUM adapter board schematics, found in the <*working_directory*>**schematics** directory. The pin mappings are shown in Figure 1.

In your custom design, determine the correct trace data pin mappings from your schematics. You implement the mapping in the Quartus II pin planner.

Figure 1. Trace Output Buffer and Pin Assignments



Preparing the Cyclone III Development Board for Lauterbach Debugging

The Altera Cyclone III 3C120 development board is highly configurable. You must configure several critical hardware settings to enable the Lauterbach trace module to access the FPGA through the MICTOR connection, and to enable the Lauterbach debug cable to access the FPGA through the 10-pin JTAG header. This section describes the required settings.

Configure the Cyclone III development board as follows:

- Connect the THDB-SUM adapter board directly to the HSMC connector labeled HSMA on the Cyclone III development board.

  If you are not using trace features, do not connect the THDB-SUM.

- Configure the small dual in-line package (DIP) switch labeled SW3. On SW3, set switch 4 to 1. (Switch 4 is also labeled MAX_ENABLE.) This configuration enables an external device, such as the Lauterbach LA-7837 Debugger, to control the JTAG chain through the 10-pin JTAG header.
- Set switch 5 on SW1, labeled MAX0, to 1 (OPEN).
- Remove jumper J6.
- Set the PCM CONFIG SELECT rotary switch to 0.

For details about configuring the Cyclone III development board, refer to the *Cyclone III 3C120 Development Board Reference Manual*.

Hardware Design Preparation

To debug your Nios II design with the Lauterbach TRACE32 system, you must configure the Nios II processor's JTAG debug module. You set the JTAG debug module to level 4, to enable hardware breakpoints, data triggers, and (if needed) off-chip trace.

In addition, to use the hardware trace features of the TRACE32 Logic Development System, you must connect the Nios II processor to a MICTOR socket on the target board.

You use the Quartus II software and SOPC Builder to make these preparations.

☞ **lauterbach.zip** includes a Quartus II project for the Cyclone III 3C120 development board in which these preparation steps are already done. The project is in the **mictor** directory, and the Quartus II project file is named **niosII_cycloneIII_3c120_host_board_top.qpf**.

Setting the JTAG Debug Module to Level 4

☞ If you are using the example design, the steps in this section are already complete. To ensure a complete understanding of the steps, you can read them while examining the example design in SOPC Builder.

To work with the Lauterbach TRACE32 Logic Development System, you must set the JTAG debug module in the Nios II processor to Level 4, as follows:

1. Open your Quartus II project, and launch SOPC Builder.
2. In SOPC Builder, double-click the Nios II processor to open the Nios II processor MegaWizard™ interface.
3. Click the **JTAG Debug Module** page, and then select **Level 4**.
4. If the hardware design targets a Stratix® Series device, under **Advanced Debug Settings**, turn off **Automatically generate internal 2X clock signal** to disable generation of a second phase-locked loop (PLL).

   ☞ This step is unnecessary with Cyclone Series devices, because extra PLLs are not available.

5. Click **Generate** to regenerate the SOPC Builder system.
6. In the Quartus II software, display the hardware design schematic.
7. Right-click the symbol for the SOPC Builder module in the schematic, and click **Update Symbol or Block**.
8. Click **OK**.

Adding a MICTOR Connection to the Hardware Design

If you are using the example design, the steps in this section are already complete. To ensure a complete understanding of the steps, you can read them while examining the example design in the Quartus II software.

To transmit trace data, your design must have a MICTOR connection. To add this connection, you can insert the Trace Output Buffer component, included in **lauterbach.zip** and shown in Figure 1 on page 5, between your SOPC Builder module and the MICTOR. This section describes how to connect your design to the MICTOR through the Trace Output Buffer.

The Trace Output Buffer data centering component buffers the trace data and shifts the data signal by adding a half-clock cycle delay. This shift ensures that the trace data is not sampled too close to a trace clock signal edge.

Table 2 lists top-level signals in the SOPC Builder module that are relevant to trace data capture. These signals are exported to the top level of the module by the JTAG debug module, which you enable in "Setting the JTAG Debug Module to Level 4".

Table 2. Trace Output Signals

Signal Name in SOPC Builder Module	Signal Name in traceoutputbuffer
`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`

Note to Table 2:

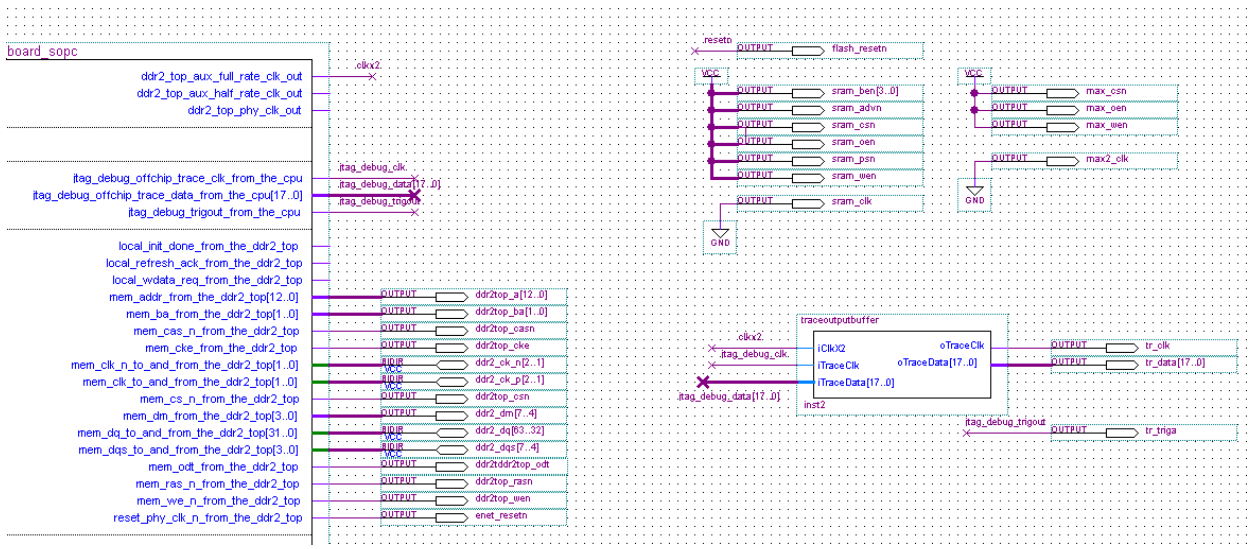
(1) `jtag_debug_trigout_from_the_cpu` connects directly to a top-level output pin on the FPGA.

(2) `clkx2_to_the_cpu` runs at double the Nios II processor clock speed.

The Trace Output Buffer component must be connected to the SOPC Builder block as shown in Figure 2. The Trace Output Buffer component exports the debugging signals shown in Table 2 to the top level.

Figure 2 is an excerpt from the hardware example design schematic, showing the portion that connects the SOPC Builder symbol to the Trace Output Buffer component.

Figure 2. JTAG Debug Nodes



To add the Trace Output Buffer component to your own hardware design, perform the following steps:

1. Copy the following two files from the <*working_directory*>/**mictor** directory to your Quartus II project directory:
   - **traceoutputbuffer.bsf**
   - **traceoutputbuffer.vhd**
2. In the Quartus II software, open the schematic in your hardware design.
3. To add the Trace Output Buffer component, right-click the schematic, point to **Insert**, and click **Symbol**.
4. Select the **traceoutputbuffer** component in the **Project** folder.
5. Make the connections in your schematic between the Trace Output Buffer component signals and the signals on the SOPC Builder symbol, as shown in Table 2 and Figure 2.
6. The `jtag_debug_trigout_from_the_cpu` pin is not routed to the Trace Output Buffer component. Route the `jtag_debug_trigout_from_the_cpu` signal to an FPGA output pin. In the hardware example design, `jtag_debug_output` is routed to a pin called `tr_triga`.
7. Compile the Quartus II project.

Creating a Raw Binary Format File of the Hardware Design

To load the hardware design into the FPGA using the TRACE32 PowerView IDE, you must generate a Raw Binary Format (**.rbf**) File. The TRACE32 IDE does not support the SRAM Object File (**.sof**) format. This section describes two ways to generate a **.rbf**.

Directly Generating a .rbf

If your design is not already compiled, the simplest way to create a **.rbf** is to generate it directly with the Quartus II software. To configure the Quartus II software to generate a **.rbf** as a result of the compilation process, perform the following steps:

1. In the Quartus II software, on the Assignments menu, click **Device** to open the **Settings** dialog box.
2. On the **Device** page, click **Device and Pin Options**.
3. Click the **Programming Files** tab.
4. Turn on **Raw Binary File**.
5. Click **OK**.
6. Click **OK** again.
7. Compile the design.

Converting a .sof to a .rbf

If you have already generated a **.sof**, you can convert it to **.rbf** format with the following steps:

1. In the Quartus II software, on the File menu, click **Convert Programming Files**.
2. Under **Output programming file**, select **Raw Binary File (.rbf)** for **Programming file type**.
3. In the **File name** box, enter a file path and file name.
4. Under **Input files to convert**, select **SOF Data.**
5. Click **Add File** and browse to the **.sof** to be converted.
6. Select the **.sof** name and click **Properties**.
7. Verify that **Compression** is turned off, and click **OK**.
8. Click **Generate** to generate the **.rbf**.

The **niosII_cycloneIII_3c120_host_board_top.sof** file provided with the hardware example design is already converted for you in a file named **mictor_01.rbf**.

Preparing the Software Example

In this application note's tutorial steps, you run the nested loops software example to demonstration the features of the Lauterbach TRACE32 system. This section describes how to build the nested loops example using the Nios II software build tools.

To build the nested loops software example, perform the following steps:

1. To open a Nios II command shell, on the Start menu, point to **Programs** > **Altera** > **Nios II EDS** <*version*>, and click **Nios II** <*version*> **Command Shell**.

2. To change to the directory where your software design application example is installed, type the following command:

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

   The remainder of this application note refers to your software application directory as <*app_directory*>.

3. Type the following command:

   `./create-this-app↵`

Starting the TRACE32 Logic Development System

This section describes how to correctly apply power to the Lauterbach PowerTrace hardware and start the TRACE32 PowerView IDE.

CAUTION

Always apply power first to the Lauterbach hardware before applying power to the target board or connecting it to the Lauterbach hardware. When powering down, always disconnect power to the Lauterbach hardware last.

Before you start, ensure that your hardware is connected as follows:

- The PowerTrace hardware is disconnected from the target board.
- The PowerTrace hardware is connected to the host.
- The LA-7837 module is connected to the LA-7707 PowerTrace module.
- The LA-3801 module is connected to the LA-7707 PowerTrace module.

  If you are not using trace features, do not connect the LA-3801.

You must perform a precise sequence of actions to correctly connect the PowerTrace hardware and attach the TRACE32 IDE. If you are debugging custom hardware, perform the tutorial instructions in this section, substituting your design file names for the example design file names.

CAUTION

Failure to follow the steps in the correct order can result in damage to the Lauterbach PowerTrace hardware, your target hardware, or both. Do not apply power to the target board until step 8.

To bring up the TRACE32 system, perform the following steps:

1. Apply power to the Lauterbach hardware.
2. Launch the TRACE32 IDE.

   Ensure that you apply power to the Lauterbach hardware before launching the TRACE32 IDE. The TRACE32 IDE expects the Lauterbach hardware to be powered up.

3. On the CPU menu, click **System Settings** to open the **B::SYStem** dialog box, shown in Figure 3.
4. In the **B::SYStem** dialog box, under **Mode**, select **NoDebug**.

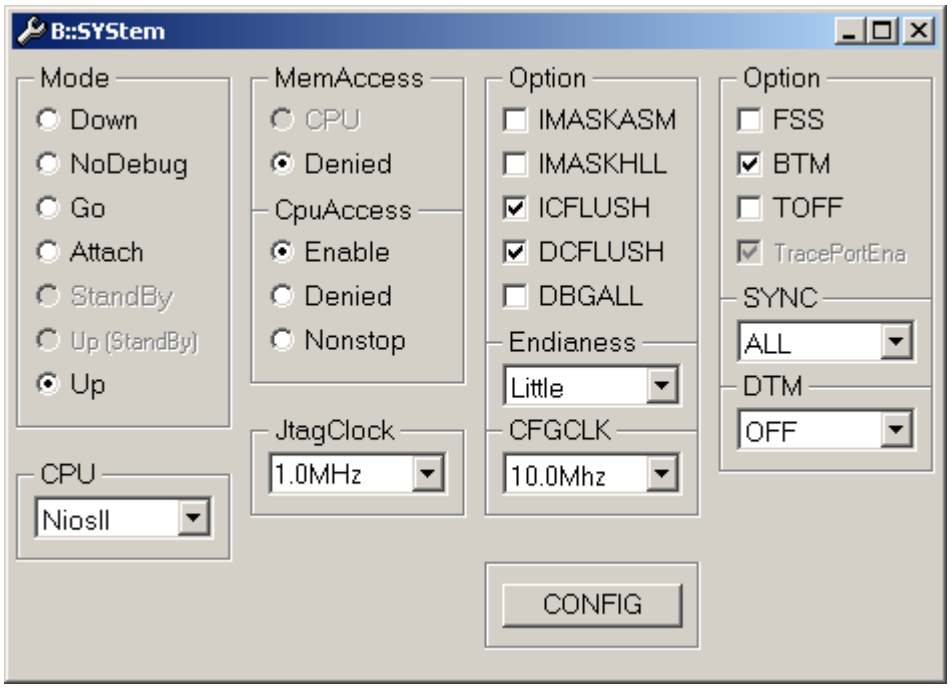
5. Ensure that the target board is disconnected from its power source.
6. Connect the Lauterbach PowerTrace LA-7837 module's 10-pin JTAG header to the target board, ensuring that pin 1 on the cable corresponds to pin 1 on the target board.
7. Plug the Lauterbach MICTOR cable into the THDB-SUM board's MICTOR socket.

   If you are not using trace features, do not connect the MICTOR cable.

CAUTION Always apply power first to the Lauterbach hardware before applying power to the target board or connecting it to the Lauterbach hardware.

8. Power on the target board. The message `running` appears in the state line at the bottom of the TRACE32 GUI.
9. Select **Attach** under **Mode** in the **B::SYStem** dialog box. The Lauterbach TRACE32 IDE establishes communications with the PowerTrace hardware, and the radio button changes to **Up**, as shown in Figure 3.

Figure 3. B::System Dialog Box



10. Ensure that the settings in the **B::SYStem** dialog box are configured as shown in Table 3.

    If **CpuAccess** is set to **Denied** (the default value), trace data is not collected. To enable this setting whenever the TRACE32 IDE launches, add the following line to the **t32.cmm** script file, located in the Lauterbach installation directory:

    ```
    system.cpuaccess enable
    ```

Table 3. Settings in the B::SYStem Dialog Box

Area	Setting Name	Setting Value
CpuAccess	**Enable**	Selected
Option	**ICFLUSH**	On
	DCFLUSH	On
	FSS	Off
	BTM	On
	TOFF	Off

11. At the bottom of the TRACE32 screen, there is a command line, identified by the prompt `B::`. Type `area` at this prompt. This command opens the B::area window, which displays the text output from subsequent commands.

    If you see the following message when you run the `area` command, you have not configured your target board correctly:

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

    If this message appears, check your board configuration, as described in "Preparing the Cyclone III Development Board for Lauterbach Debugging" on page 5.

12. Program the **.rbf** in the FPGA with the following command:

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

    You see the following messages:

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

    If the message `target processor in reset` appears, communications is not established between the TRACE32 IDE and the PowerTrace hardware. Select **Attach** under **Mode** in the **B::SYStem** dialog box, and wait for the radio button to change to **Up**, as shown in Figure 3.

    If the message `access timeout processor running` appears, the target is not stopped. Click Break , and type the `jtag.loadrbf` command again.

13. In the tool bar, click Break to stop the target, in preparation for loading the software. The `running` message in the state line is replaced by `stopped`.

14. Use the `data.load.elf` command to load **nested_loops.elf**, as follows:

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

This example of the `data.load.elf` command demonstrates the following command-line arguments:

- `/PLUSVM`—loads the code into virtual memory on the host and the target. For a full trace listing, the code must be loaded on the host.
- `/StripPART 3`—makes source code path names compatible with the Lauterbach TRACE32 IDE. The GNU tools build the executable and linking format (**.elf**) file with Cygwin-style path names. However, the TRACE32 IDE expects to access C source code through a Windows-style path. The `/StripPART` switch converts the path names in the **.elf** to Windows-compatible path names so the TRACE32 IDE can locate source code referenced in the **.elf**.
- `/PATH C:\`—defines the root path where the TRACE32 IDE is to begin searching for source code. You can specify multiple search paths with the `/PATH` argument. Alternatively, to search additional paths, on the View menu, point to **Symbol** and click **Source Search Paths**.
- `/stabs`—specifies that the debugging information is in STABS format.

15. After the **.elf** is successfully loaded, the following message appears in the B::area window:

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

16. Click **Go**, then **Break**. The Nios II program starts execution, runs through the initialization code, and breaks in the body of the **nested_loops.c** example application.

To view the source code for **nested_loops.c**, refer to "Displaying Processor Usage Data" on page 17.

Viewing System State in the TRACE32 PowerView IDE

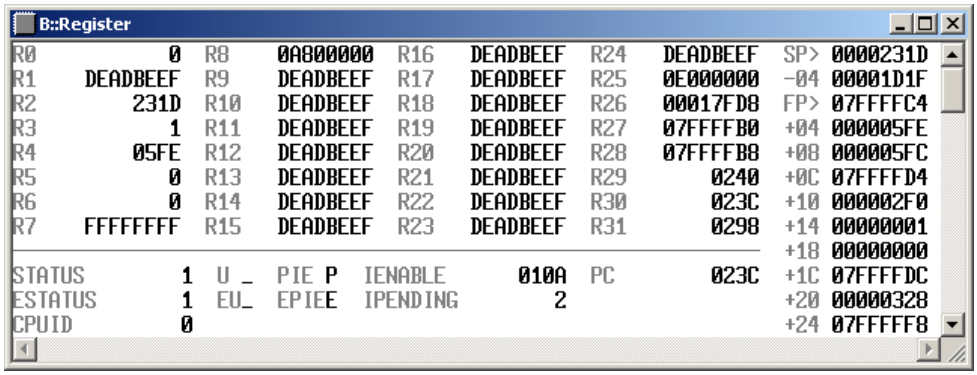
After you load the **.elf**, the TRACE32 PowerView IDE can examine the current state of the Nios II system. The following sections describe some ways to examine the system state.

You must click **Break** before you can view the system state.

Viewing Nios II Registers

To view the Nios II registers, on the View menu, click **Registers**. Figure 4 shows the Nios II registers as they are set up in the example design.

Figure 4. Register Window



Viewing Nios II Source Code

To examine the source code, on the View menu, click **List Source**. The B::Data.List window appears, displaying your application source code.

You can click **Mode** to show or hide assembly language.

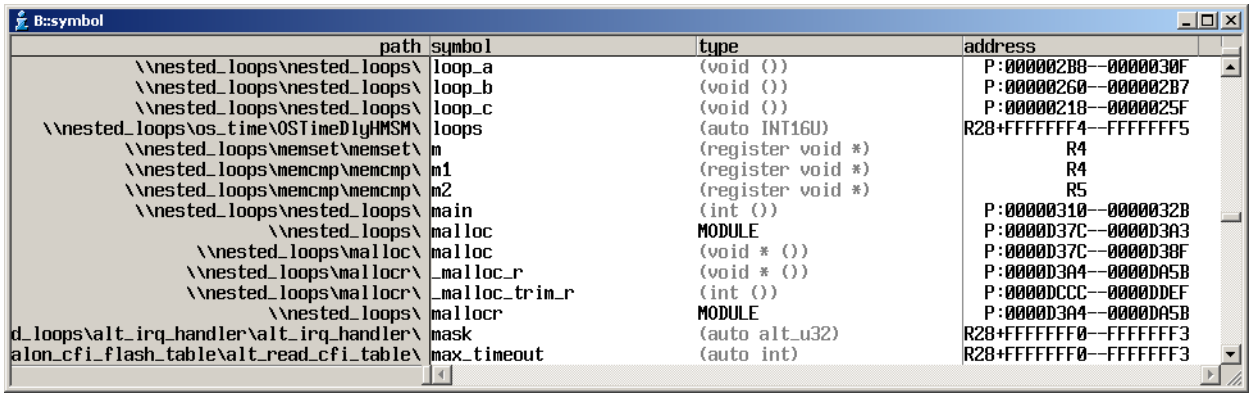
Finding Symbols

On the View menu, you can point to **Symbols** and click **Browse** to display all symbols defined in the program. Click on a symbol name to open the source file that defines the symbol.

Displaying Symbols

At the `B::` prompt, type the `symbol` command to display the symbols defined in your C source code. Figure 5 shows the symbols defined in **nested_loop.c**.

Figure 5. Symbol Definition Window



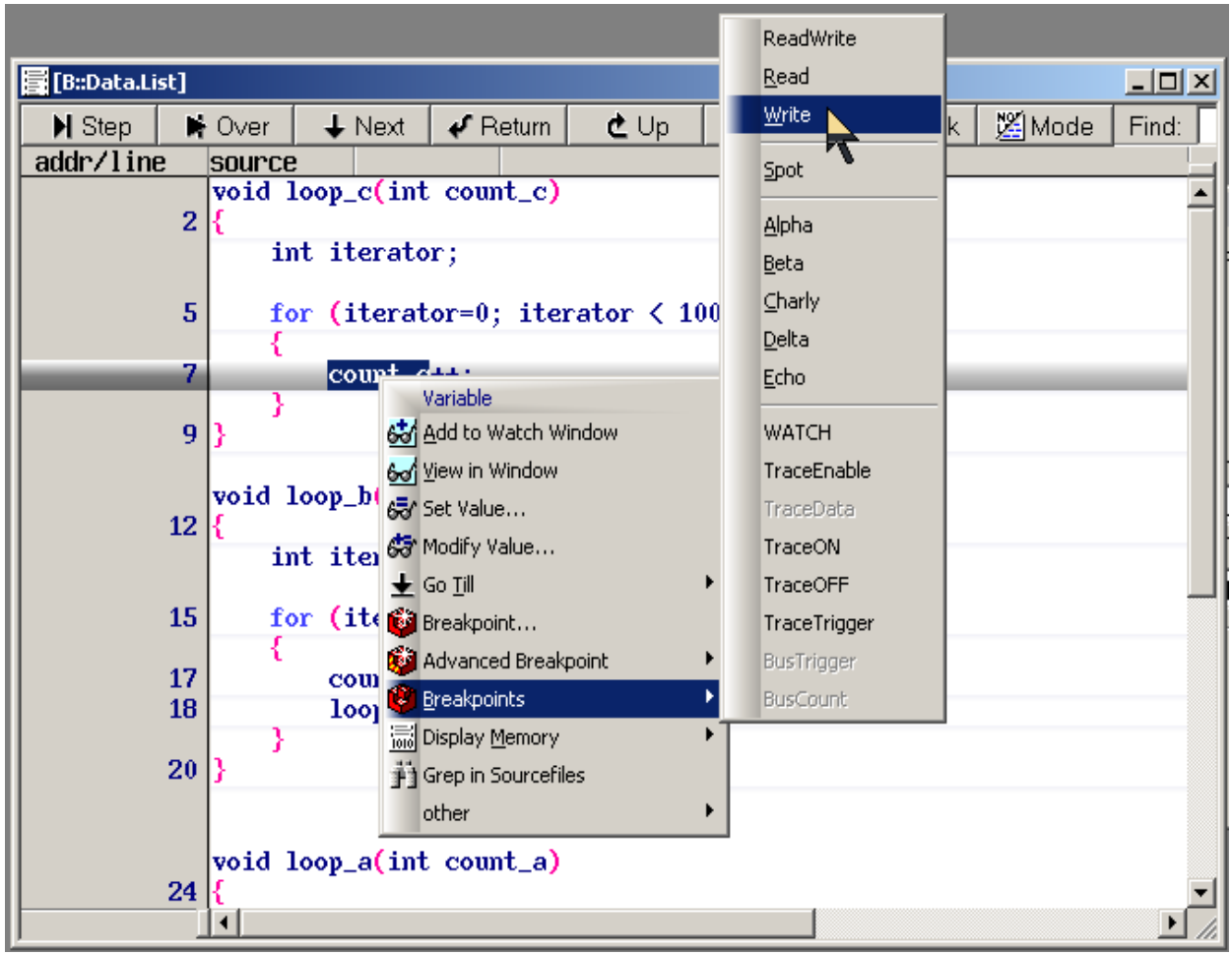
Setting Breakpoints and Watchpoints

In the B::Data.List window, you can set breakpoints and watchpoints. For example, to watch for writes to variable `count_c`, perform the following steps:

1. Open the B::Data.List window as described in "Viewing Nios II Source Code".
2. Scroll to the top of the `loop_c()` function.
3. Click the variable name `count_c` to select.
4. Right-click `count_c`, point to **Breakpoints**, and click **Write**, as shown in Figure 6.

☞ On the Break menu, you can click **List** to view a list of breakpoints.

Figure 6. Write Breakpoint



5. Click **Go**. The debugger pauses execution every time data is written to the `count_c` variable.

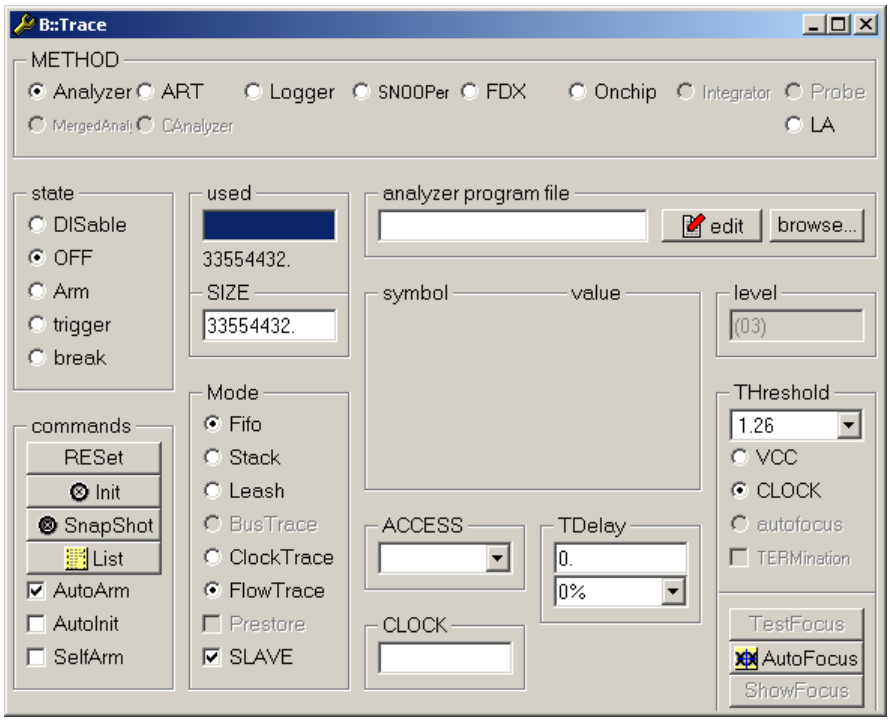
☞ On the View menu, you can click **Locals** to view the value of `count_c`.

Using Trace

Before capturing Nios II instruction execution, you must configure the TRACE32 PowerView IDE's trace options, by performing the following steps:

1. On the Trace menu, click **Configuration** to display the **B::Trace** dialog box, shown in Figure 7.

Figure 7. Trace Configuration Dialog Box



2. Verify the following settings:
   - Under **METHOD**, make sure **Analyzer** is selected.
   - Make sure **THreshold** is set to an appropriate level, depending on the target design. For the example design, 1.26V is an appropriate level. For custom hardware, the appropriate level is design-dependent.
3. Click **RESet** and **Init** to clear the trace buffer.
4. In the TRACE32 toolbar, click **Go** to begin collecting trace data. In the **Trace** configuration dialog box, the **used** bar fills.

If the **used** bar does not fill completely, check for one of the following causes:

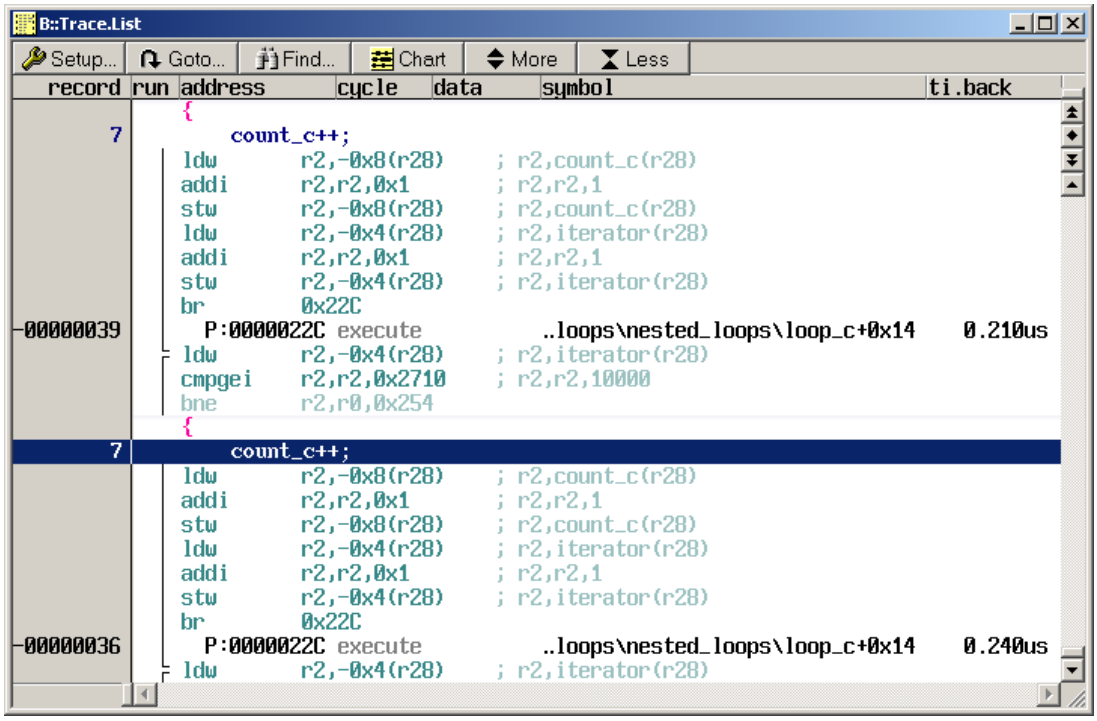
- You have set a breakpoint or a watchpoint.
- The trace clock is not properly connected in your hardware design.
- The trace clock is not sampled correctly. Ensure that a valid trace clock is generated, and that the voltage threshold for the trace data is appropriate.

You must click **Break** before you can view trace data.

Displaying a Code Trace

On the Trace menu, point to **List** and click **Default** to view the Nios II instructions captured by the trace, as shown in Figure 8. The Nios II assembler instructions are interspersed with the corresponding C source code.

Figure 8. Code Trace Listing



Displaying Processor Usage Data

To view the time spent in each function in graphical form, on the Trace menu, point to **Chart** and click **Symbols**. See Figure 9 for an example.

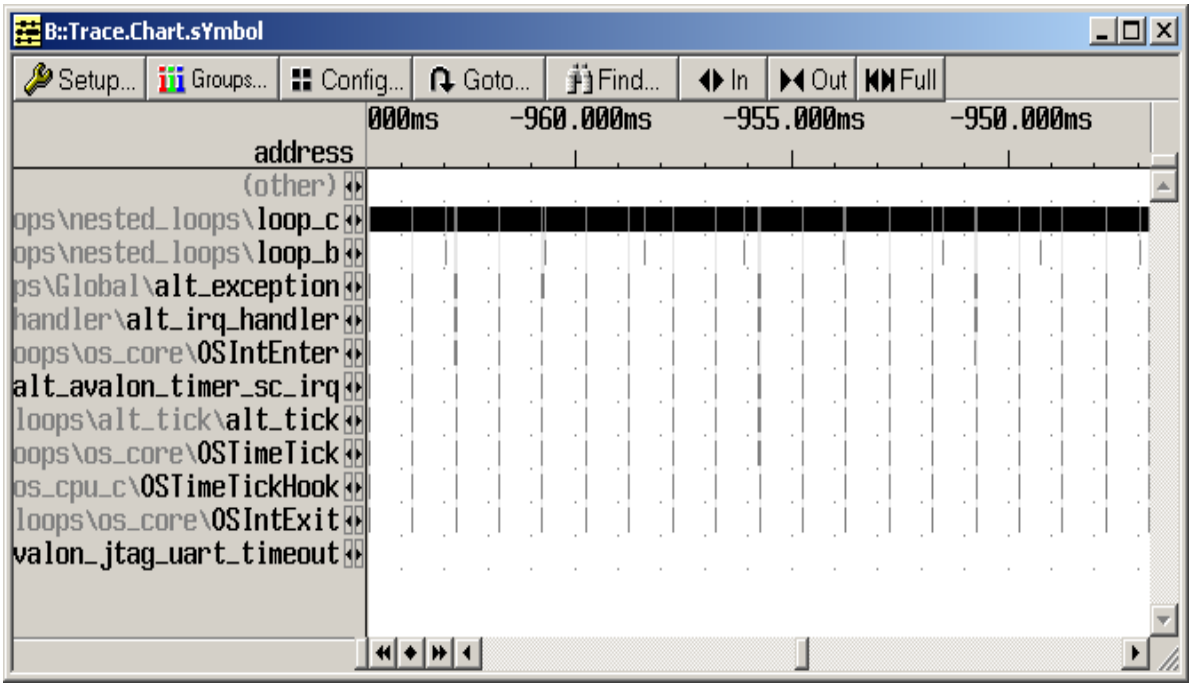
You can control the scope and detail of the data in the view as follows:

- Click **In** to zoom in for a more detailed view.
- Click **Out** to zoom out for a wider view.

The first time you display the processor usage data after capturing the data, the TRACE32 IDE takes a few moments to render the graph.

Figure 9 shows a trace data set captured from the nested loops application. Each row in the diagram represents the time spent executing one function, listed at the left side of the diagram. The x axis is the execution timeline of the application.

Figure 9. Extended Processor Usage Graph



The nested loops application consists of three loops:

- `loop_a()`, the outermost loop, calls `loop_b()` 10,000 times.
- `loop_b()` calls `loop_c()` 10,000 times.
- `loop_c()`, is the innermost loop, increments the variable `count_c` 10,000 times.

Example 1 shows the source code in the example program, **nested_loops.c**. The Nios II processor spends almost all of its time in `loop_c()`.

Example 1. nested_loops.c Source Code (Part 1 of 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... */
```

Example 1. nested_loops.c Source Code (Part 2 of 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;
}
```

Figure 9 shows the time spent in `loop_c()` as a nearly-solid bar. Interrupts and occasional returns to `loop_b()` appear as small periodic ticks in the execution timeline. `loop_b()` did not happen to return during the capture of this data set, so `loop_a()` does not appear.

Figure 10 shows a detailed view of the same trace data shown in Figure 9. The detailed view is achieved by zooming in on the data set. This view shows the time spent processing a timer interrupt and returning to the `loop_c()` function in the example application.

Special Hardware Techniques

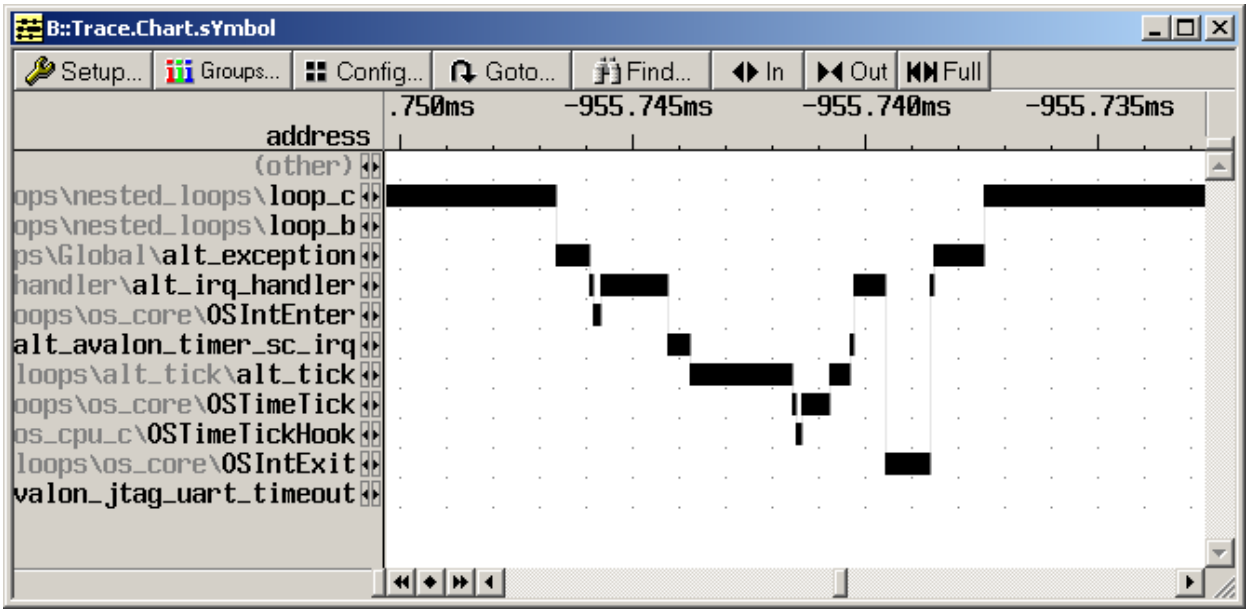
This section describes maintenance and diagnostic techniques that you might occasionally need to prepare your target hardware for debugging with the Lauterbach TRACE32 system.

Hot Swapping the Target Board's JTAG Connection

You can hot swap the Lauterbach JTAG connector with another device, such as the USB-Blaster™. However, to do so safely, you much follow the guidelines in this section.

It is important always to set the **System Settings** mode to **NoDebug** in the TRACE32 IDE before connecting the Lauterbach hardware to, or disconnecting it from, a powered-up target board. Failure to set the mode to **NoDebug** can result in hardware damage. To set the **System Settings** mode, on the CPU menu, click **System Settings** to open the **B::SYStem** dialog box, and under **Mode**, select **NoDebug**.

Figure 10. Detailed Processor Usage Graph



You might need to hot swap the Lauterbach JTAG connector with another device for one of the following reasons:

- To troubleshoot JTAG connection issues. For example, you might need to swap in another JTAG download cable in place of the Lauterbach LA-7837 debugger.
- To disconnect or reconnect the target board without powering down. For example, if you use the Quartus II Programmer with a USB-Blaster to program the FPGA with a **.sof**, you must leave the target board powered up to preserve the **.sof** image in memory.

> If you have access to the Quartus II software, it is much safer to avoid hot-swapping by converting the **.sof** to a **.rbf**. When you have your hardware design in a **.rbf**, you can use the TRACE32 IDE to program the **.rbf** in the FPGA through the Lauterbach JTAG connection. For steps to create a **.rbf**, refer to "Creating a Raw Binary Format File of the Hardware Design" on page 8.

Before you resume debugging, you must select **Attach** mode, as described in "Starting the TRACE32 Logic Development System", at step 9 on page 11.

Restoring Factory Default Board Settings

If you update the hardware design on the MAX II device, or update the flash option bits, first you need to restore the default factory settings for switches and jumpers on the Cyclone III development board. The default factory settings for switches and jumpers on the development board are as follows:

- SW3 position 1, labeled FPGA BYPASS, is set to 1.
- All other positions on SW3 are set to 0.

- Jumper J6 (DEV_SEL) is installed.
- Jumper J7 (JTAG_SEL) is installed.
- SW1 position 5, labeled MAX0, is set to 1 (OPEN).

Before you resume debugging with the Lauterbach TRACE32 system, return the Cyclone III development board's jumper and switch settings to their debugging configuration, as described in "Preparing the Cyclone III Development Board for Lauterbach Debugging" on page 5.

For details about configuring the Cyclone III development board, refer to the *Cyclone III 3C120 Development Board Reference Manual*.

Updating the MAX II Design on the Cyclone III Development Board

Before you use the Cyclone III 3C120 development board with the Lauterbach PowerTrace hardware, you might need to update the hardware design programmed in the Max II device. You only need to perform this update once.

To determine which build of the Max II design is installed, apply power to the board and hold down the CPU RESET (underneath the LCD display). Look for the build number on the seven-segment display labeled POWER DISPLAY. If the displayed build number is 1338 or higher, your version of the Max II design is prepared to work with the Lauterbach PowerTrace hardware. Otherwise, you have an older version of the Max II design. You must update the design before proceeding further.

To update the Max II design you need the following items:

- The Max II design file **m2_prod_1338.pof**. This file is included in **lauterbach.zip**.
- An Altera USB-Blaster

  You cannot use the Cyclone III development board's embedded JTAG programmer to update the MAX II design. The embedded JTAG programmer circuitry is partly implemented in the Max II CPLD.

To update the Max II design, perform the following steps:

1. Disconnect power from the Cyclone III development board.
2. Ensure that the switches and jumpers are in the default factory settings, as described in "Restoring Factory Default Board Settings".
3. Connect the USB-Blaster to the board through the 10-pin JTAG port, found on the side of the board. Ensure that pin 1 on the cable corresponds to pin 1 on the target board.
4. Apply power to the Cyclone III development board.
5. On the Quartus II Tools menu, click **Programmer** to launch the Quartus II Programmer.
6. Ensure that **JTAG** is selected for **Mode**.
7. Click **Hardware Setup** and ensure that the USB-Blaster connected to your Cyclone III board is selected.
8. Click **Auto Detect**. The Quartus II Programmer finds and lists the **EPM2210**.

9. Double click **<none>** for the EPM2210 device.
10. Browse to and select *<working_directory>***/board_config/m2_prod_1338.pof**, and click **Open**.
11. Turn on the **Program / Configure** option for the EPM2210 device.
12. Click **Start**.
13. When programming is complete, confirm the presence of the correct Max II design. Hold down the CPU RESET button. The POWER DISPLAY shows `1338`.

Before you resume debugging with the Lauterbach TRACE32 system, return the Cyclone III development board's jumper and switch settings to their debugging configuration, as described in "Preparing the Cyclone III Development Board for Lauterbach Debugging" on page 5.

Updating the Flash Option Bits on the Cyclone III Development Board

Before you use the Cyclone III 3C120 development board with the Lauterbach PowerTrace hardware, you might need to update the flash option bits to enable parallel flash programming implemented in the MAX II. You only need to perform this update once.

To update the flash option bits you need the following items:

- The Jam Byte-Code file **3C120_flash_option_bits.jbc**. This file is included in **lauterbach.zip**.
- An Altera USB-Blaster

To update the flash option bits, perform the following steps:

1. Disconnect power from the Cyclone III development board.
2. Ensure that the switches and jumpers are in the factory default settings, as described in "Restoring Factory Default Board Settings".
3. Connect the USB-Blaster to the board through the 10-pin JTAG port, found on the side of the board. Ensure that pin 1 on the cable corresponds to pin 1 on the target board.
4. Apply power to the Cyclone III development board.
5. To open a Nios II command shell, on the Start menu, point to **Programs** > **Altera** > **Nios II EDS** *<version>*, and click **Nios II** *<version>* **Command Shell**.
6. Type the following commands to program the flash options bits with the Quartus II JBI Player:

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

Wait for the Quartus II JBI Player to finish programming flash. This process can take several minutes.

Before you resume debugging with the Lauterbach TRACE32 system, return the Cyclone III development board's jumper and switch settings to their debugging configuration, as described in "Preparing the Cyclone III Development Board for Lauterbach Debugging" on page 5.

Troubleshooting OCI Core Communications

If there are problems with the Nios II On-Chip Instrumentation (OCI) core, the Lauterbach TRACE32 **System.Attach** (or **System.Up**) command might fail with the message `monitor not responding`. This message indicates one of several communication problems between the TRACE32 system and the Nios II OCI core. You can use the debug script **lauterbach_system_up_debug_script.cmm** to help identify the cause of the problem.

The **lauterbach_system_up_debug_script.cmm** script is included in **lauterbach.zip**, and you can find it in the *<working_directory>***troubleshooting** directory. To use this script, execute the following instructions:

1. Copy **lauterbach_system_up_debug_script.cmm** to the Lauterbach installation directory.
2. If the TRACE32 IDE is not already running, launch it.
3. On the File menu, click **Run Batchfile**, then double-click the **lauterbach_system_up_debug_script.cmm** script.

The script generates the following files, which provide diagnostic information enabling you to analyze the failure:

- **debug.log**—The results of each command executed in the debug script.
- **debug.lst**—Produced by the trace portion of the debug script. If the system is functioning properly, **debug.lst** contains raw data and an up-counting timestamp.

In *<working_directory>***troubleshooting** you can find the following examples of **debug.log** and **debug.lst**:

- **good_debug.log** and **good_debug.lst**—Generated from a successful invocation of **lauterbach_system_up_debug_script.cmm** on a Cyclone III Nios II Embedded Evaluation Kit (NEEK) board with MICTOR daughter card.
- **failed_debug.log**—Generated on a failing Lauterbach connection to a custom board.

Compare your **debug.log** and **debug.lst** with the examples to identify the cause of the communication error.

For assistance with interpreting the test results in **debug.log** and **debug.lst**, contact Lauterbach Technical Support through www.lauterbach.com.

Referenced Documents

This application note references the following documents:

- *Nios II Development Kit Getting Started User Guide*
- *Nios II Hardware Development Tutorial*
- The Nios II Software Development Tutorial, available in the Nios II IDE by clicking **Tutorials** on the Welcome page

- TRACE32 documentation installed with the TRACE32 PowerView IDE. Documentation for the TRACE32 Logic Development System is also available at www.lauterbach.com. The following documents are of particular interest:
  - *Nios II Debugger and Trace* (**debugger_nios.pdf**)
  - *Nios II Instantiating the Off-chip Trace Logic* (**app_nios.pdf**)—information on multiplexing trace outputs from multiple Nios II processor cores

  ☞ In *Nios II Instantiating the Off-chip Trace Logic*, disregard the steps under "Disable automatic PLL instantiation". Step 4 on page 6 accomplishes the same task.

- *Santa Cruz, USB, Mictor, SD Card HSMC Reference Manual*.
- *Cyclone III 3C120 Development Board Reference Manual*.

Document Revision History

Table 4 shows the revision history for this application note.

Table 4. Revision History

Date and Revision	Changes Made	Summary of Changes
April 2009 version 1.0	Initial Release	—