



# Debugging Nios II Systems with the SignalTap II Embedded Logic Analyzer

June 2008, ver. 1.2

Application Note 446

## Introduction

As FPGA system designs become more complex and system focused—with increasing numbers of processors, peripherals, buses, and bridges—designers need better and more sophisticated system-level debug tools. Altera's SignalTap® II Embedded Logic Analyzer provides real-time hardware debugging capabilities by embedding a logic analyzer in the system. The Nios® II plug-in for the SignalTap II Embedded Logic Analyzer extends the SignalTap II system debugging capabilities by enabling the capture of a Nios II processor's program execution.

This application note teaches you to debug your system design using dynamic information provided during software execution by the Nios II processor. It examines the use of the Nios II plug-in for the SignalTap II Logic Analyzer, and presents the capabilities, configuration options, and use model for the plug-in. A short tutorial demonstrates how to use the Nios II plug-in, SignalTap II Logic Analyzer, and Nios II Integrated Development Environment (IDE) to trigger on, capture, and trace the receipt of a character from an RS-232 UART interface.

The Nios II plug-in extends the capabilities of the SignalTap II Embedded Logic Analyzer, enabling you to easily trigger on and capture instruction trace data being executed by the Nios II processor core. You can specify an instruction-trace trigger, which triggers the SignalTap II Logic Analyzer when the processor reaches a specific address, by entering a symbol name from your program, or specify your own SignalTap II trigger condition. The Nios II plug-in automatically correlates the processor trace with a specified software image, providing you with a symbol name plus offset view of the trace, along with decoded Nios II machine language operation codes (op-codes).

## Prerequisites

The goal of this application note is to teach you how to use the Nios II plug-in in the SignalTap II framework. The application note assumes you are familiar with certain Altera® software tools, including the basic use of the SignalTap II Embedded Logic Analyzer. In addition, the tutorial assumes you have access to certain Altera tools and intellectual property (IP).

This application note assumes that you are familiar with the use of the Quartus® II software, SOPC Builder, and Nios II embedded processor, and the basic capabilities and operation of the SignalTap II Embedded

Logic Analyzer. Because the Nios II plug-in is an extension of the SignalTap II Logic Analyzer, the basic use of the plug-in is nearly identical to that of the SignalTap II Logic Analyzer.



For more information about the SignalTap II configuration options and use modes, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

### Tool Requirements

Using the Nios II plug-in requires the following tools:

- Quartus II software version 8.0 or later
- Nios II Embedded Development Suite 8.0 or later

Completing the tutorial at the end of this application note requires the following additional resources:

- Nios II Development Kit
- RS-232 serial cable (optional)
- JTAG ByteBlaster™ or USB-Blaster™ cable
- The source file **signal\_tap\_test.c** (available as a downloadable file with this application note)

For more detailed information about the hardware and software required for the tutorial, refer to “[Tutorial: Using the Nios II Plug-In](#)” on page 11.

## The Nios II Plug-In

The Nios II plug-in is a debugging extension to the SignalTap II Embedded Logic Analyzer. It enables you to capture the op-codes executed by a Nios II embedded processor. The Nios II plug-in operates by instantiating debug nodes inside the Nios II processor's ALU. The Nios II plug-in supports all variants of the Nios II processor core.

Each Nios II plug-in instantiation is associated with a specific Nios II processor, and can operate with other Nios II plug-in instantiations and with other SignalTap II instances.

You can specify the following types of trigger conditions:

- Instruction address
- Symbol name (function) present in the Nios II Executable and Loadable Format (**.elf**) file
- Symbol name plus an offset

The plug-in decodes all the instructions it captures to a human-readable format. Instruction addresses are translated to symbol name plus address offset (if possible), and instruction op-codes are translated to their equivalent assembly language mnemonics.

## Quartus II Project and SignalTap II Logic Analyzer Set-Up

You must create and configure a SignalTap II file in your Quartus II project for use with the Nios II plug-in. To add the SignalTap II file to your system, perform the following steps:

1. On the Quartus II File menu, click **New**.
2. In the **New** dialog box, in the **Verification/Debugging Files** category, click **SignalTap II Logic Analyzer File**.
3. Click **OK**.

Alternatively, you can open a new or existing SignalTap II file by performing the following steps:

- ✓ On the Quartus II Tools menu, click **SignalTap II Logic Analyzer**.

After you perform these steps, a SignalTap II window appears.

The Nios II plug-in currently does not support Quartus II projects in which the incremental compilation option is set. Therefore, you must ensure that incremental compilation is disabled in your Quartus II project. To disable incremental compilation, perform the following steps:

1. On the Assignments menu, click **Settings**.
2. In the **Settings** dialog box, expand **Compilation Process Settings** and click **Incremental Compilation**.
3. Under **Incremental Compilation**, select **Off**.
4. Click **OK**.

## Adding the Nios II Plug-In

When you add the Nios II plug-in to your system, you must specify the Nios II processor you wish to monitor and, optionally, the processor's software image, an **.elf** file. The Nios II plug-in processes the **.elf** file to extract symbol information, which is used during configuration to specify trigger conditions.

To add the Nios II plug-in to your system, perform the following operations:

1. In the Quartus II window, on the Processing menu, point to **Start**, and then click **Start Analysis & Elaboration**.
2. In the SignalTap II window, right-click in the SignalTap II node list. Point to **Add Nodes with Plug-In** and click **Nios II**. The **Select Hierarchy Level** dialog box appears.
3. In the **Select Hierarchy Level** dialog box, select the Nios II processor instance you want to monitor with the plug-in.
4. Click **OK**. The **Plug-In Options** dialog box appears.
5. In the **Plug-In Options** dialog box, optionally specify the location of the **.elf** file.



Two Nios II plug-in instances cannot monitor the same Nios II processor. If you attempt to add more than one instance of the Nios II plug-in per processor, an error message appears.

6. Click **OK**.

You can change the **.elf** file used by the Nios II plug-in at any time by performing the following steps:

1. In the SignalTap II window, click the **Setup** tab.
2. In the SignalTap II node list, right-click on the Nios II plug-in instance you want to modify and click **Plug-In Options**. The configuration options for the Nios II plug-in appear.



The **.elf** file is generated by the Nios II IDE during the software build process. In most cases, the **.elf** file for your software project is located in the **Debug** directory or the **Release** directory created by the Nios II IDE. You can also create an **.elf** file in the Nios II software build tools flow.

### Specifying Trigger Conditions

Unlike standard SignalTap II Logic Analyzer trigger conditions, which are described as hardware or logic events, the Nios II plug-in's trigger conditions are specified as instruction addresses. The Nios II plug-in is triggered when the Nios II processor reaches the specified instruction address during program execution.

## Basic Triggering

In basic triggering mode, the Nios II plug-in uses a processor-visible system address as the trigger to begin trace capture. To set the trigger, click the **Trigger Conditions** column in the Nios II plug-in and type an instruction address.

You can type any of the following supported trigger conditions:

- Hexadecimal integer: `0x<32-bit number>` (for example, `0x20000000`)
- **.elf** file symbol: `<string>` —An alphanumeric C/C++ function name that appears as a symbol in the **.elf** file (for example, `f00`)
- **.elf** file symbol + offset: `<string>+<hexadecimal number>` (for example, `f00+0x80`)

To specify a trigger condition with a symbol name option you must configure the Nios II plug-in with a Nios II processor **.elf** file. If the **.elf** file does not contain the specified symbol name, an error message appears and the trigger condition field is set to an undefined value. When you type a value in the trigger condition field, the Nios II plug-in examines the contents of the reference **.elf** file for any changes. This check guarantees that the trigger conditions remain synchronized with the **.elf** file contents, even if the **.elf** file is changing frequently, such as during software debugging.

To determine the symbols in the **.elf** file, use one of the following methods to generate an **objdump** file:

- Turn on the `CREATE_OBJDUMP` option in the makefile, and then build the project.
- After you create the **.elf** file, in the Nios II command shell, type the following command:

```
nios2-elf-objdump -s <filename>.elf ↵
```

For more information refer to [“Building the Nios II Software” on page 13](#).



You can specify your standard SignalTapII Logic Analyzer trigger conditions in the SignalTap II window by right-clicking on the **Trigger Conditions** column and selecting a trigger pattern. However, you should enter your Nios II plug-in trigger conditions manually for clarity.

Not specifying an address trigger condition for the Nios II plug-in has the same effect as not specifying a trigger condition in a normal SignalTap II instance. If you specify no trigger condition, the logic analyzer triggers immediately and the displayed data is generally not useful.

### *Multiple Triggers*

The Nios II plug-in supports the SignalTap II multiple trigger conditions feature, and can be incorporated as part of a more complex capture sequence. You can include the Nios II plug-in trigger pattern as a part of any SignalTap II trigger condition.

### *Advanced Triggers*

Complex triggers can be created using the SignalTap II Advanced Trigger option. However, the Nios II plug-in loses much of its benefit when this option is used. The signal groupings contained in the Nios II plug-in appear in the Advanced Trigger Configuration Editor, where they are treated as normal signals. The Advanced Trigger Configuration Editor lacks the Nios II plug-in's capability to trigger by address or symbol.

### *Power-Up Triggers*

You can use the Nios II plug-in with the SignalTap II power-up trigger feature. Because power-up triggers are enabled before the SignalTap II Logic Analyzer is started manually, they are useful for monitoring systems in which the Nios II processor operates in self-booting mode, that is, immediately after the FPGA is configured. In these cases, the Nios II processor begins software execution directly from system memory without the aid of a debugger to start, stop, and load the processor's run-time memory.

## **Assigning the Acquisition Clock**

You must specify a clock signal to control the acquisition of samples. Specify the clock signal in the Signal Configuration pane of the SignalTap II window. Altera recommends that you select the clock signal used by the Nios II processor as the SignalTap II acquisition clock. Using the Nios II processor clock ensures that the captured instruction trace data accurately corresponds to the instruction execution of the Nios II processor.

## **Selecting Sample Depth, Memory Type, and Buffer Acquisition Mode**

For the Nios II plug-in, just as for the SignalTap II Logic Analyzer, you must configure the sample depth, memory type, and buffer acquisition mode for the capture session. These configuration options behave as they would in a normal SignalTap II Logic Analyzer capture session, and are accessible through the Signal Configuration pane.

Exercise care in selecting the **Sample Depth** size. The Nios II plug-in requires many signals to be captured for every sample taken, quickly depleting available memory resources. Use the SignalTap II built-in resource estimator to gain a better understanding of how adjusting the sample depth parameter impacts your design.

## Design Compilation and Programming the Target Device

The Nios II plug-in is compiled in the Quartus II design with the SignalTap II Logic Analyzer. However, because the Quartus II incremental compilation option is not supported by the Nios II plug-in, you must perform a full compilation of the Quartus II project after adding the plug-in.

After compilation, you can program the FPGA target device with the SRAM Object File (.sof) from the SignalTap II window just as you would if you were not using a Nios II plug-in.

## Running a Capture Session

You perform data acquisition with the Nios II plug-in the same way that you gather data with the SignalTap II Logic Analyzer. First, program the FPGA with the .sof file generated by the Quartus II software. Next, run SignalTap II analysis, either manually through the SignalTap II Instance Manager, or automatically when the FPGA is programmed and power-up triggering is selected. If the system meets the trigger conditions, the SignalTap II Logic Analyzer displays the acquired data in the SignalTap II results window.

You can use the Nios II plug-in in two different types of data capture sessions, one with the Nios II IDE and the other in stand-alone mode.

## Performing Data Capture with the Nios II IDE

To use the Nios II plug-in with the Nios II IDE, you must manually download a Nios II software image and control the operation of the processor through the debugger. This type of capture session is usually conducted when you are developing and debugging a Nios II software application.

To run a SignalTap II capture session with the Nios II processor controlled by the Nios II IDE, perform the following steps:

1. In the SignalTap II window, program the FPGA target device with the .sof file generated by the Quartus II software, by performing the following steps:

- a. On the Hardware menu, select the programming cable that is connected to your Nios II development board.
  - b. In the **SOF Manager** field, click the browse button.
  - c. In the **Select Programming File** dialog box, select the **.sof** file generated for your project.
  - d. Click **Open**. The **Program Device** button is now available.
  - e. Click the **Program Device** button to download the **.sof** file to the FPGA. 
2. In the SignalTap II window, in the Instance Manager pane, click the **Run Analysis** button to start the logic analyzer capture session. 
  3. In the Nios II IDE, right-click on the name of the software project you want to run on the Nios II processor and click **Debug As**. This action starts the debugger, downloading the **.elf** file into system memory and halting the processor on the entry point to `main()`.
  4. On the **Debug** tab, click the **Resume** button to start the Nios II processor execution. 

The SignalTap II Logic Analyzer continues running until the trigger condition specified in the Nios II plug-in is reached. While the SignalTap II Logic Analyzer is running, you can use all of the Nios II IDE debug operations safely (for example, you can set breakpoints and stop the processor).

When you launch the debugger, the Nios II plug-in triggers if the processor advances to the trigger address. If the start-up breakpoint location occurs after the trigger address location specified for the Nios II plug-in, it may cause a false hit by the debugger. To change the breakpoint start-up location for the debugger, perform the following steps in the Nios II IDE:

1. On the **Run** menu, click **Debug**. The Debug window appears.
2. In the Debug window, click the **Debugger** tab.
3. Select the breakpoint location(s) from which you want the Nios II IDE debugger run, and click **Apply**.



Alternatively, instead of using the **Debug As** option, you can use the **Run As** option. Using the **Run As** option causes the Nios II IDE to download and run the software image from system memory without invoking the debugger feature of the Nios II IDE.

## Performing Data Capture Without External Software Download

If your Nios II processor based system is configured to be self-booting, without the need for an external software download, and you have selected the SignalTap II power-up trigger feature, the SignalTap II Logic Analyzer begins running automatically when the FPGA is programmed.

In this case, the SignalTap II Logic Analyzer may already have captured data available. To determine if captured data is available, or if the logic analyzer is still running, click **Run Analysis** in the SignalTap II instance manager.



For more information about creating a self-booting Nios II processor system, refer to the *Nios II Software Developer's Handbook*.

## Analyzing Results

The Nios II plug-in allows you to view the captured Nios II processor trace data. This section describes several of the post-capture features of the Nios II plug-in.

### Viewing the Data

Captured SignalTap II data appears in the **Data** tab of the SignalTap II window. Every sample captured by the Nios II plug-in displays the following information:

- **Address**—The instruction address location in hexadecimal format. Additionally, if an **.elf** file was specified during plug-in configuration, the instruction address may be further resolved into symbol name and offset.
- **Assembly Language Mnemonic**—The Nios II assembly language equivalent of the binary op-code for the instruction.

If you specified an **.elf** file during the plug-in configuration, the software re-examines the file immediately after data acquisition completes, and converts the instruction addresses to the symbol name and offset representation. This re-examination helps to safeguard against the inadvertent use of old software images.

Using the SignalTap II tab controls, you can scroll through the program execution of the Nios II processor. If the specified acquisition clock corresponds to the Nios II processor clock, every rising clock edge corresponds to a new instruction cycle.

You may notice one or more “**empty**” instruction entries in the trace data gathered by the Nios II plug-in. These entries indicate that no instruction was executed by the Nios II processor during that particular clock cycle. This behavior is normal, and can occur for the following reasons:

- **Cache Miss**—The requested instruction address location generates a miss in the instruction cache, and additional clock cycles are required to fill the cache line and return the instruction.
- **Memory Contention or Speed**—The instruction address location is in memory that requires multiple clock cycles to access, or in memory that is currently controlled by another peripheral or processor.

You can also view the Nios II plug-in trace data in the SignalTap II list file format. In this tabular format, the trace samples are displayed chronologically in rows, decoded by sample number and the associated assembly language mnemonic. The list file format is useful because it is similar to the output format of the `nios2-elf-objdump` command, simplifying the analysis process. To create the SignalTap II list file, on the File menu, point to **Create/Update** and click **Create SignalTap II List File**.



Some Nios II processor instructions consume multiple clock cycles when they are executed. For more information about the number of clock cycles required for a particular instruction, refer to the *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*.

### Correlating Trace Data to the Processor's .elf File

You can compare the captured Nios II plug-in instruction trace to the software image executed by the Nios II processor by examining the contents of the **objdump** file.

You can use the `nios2-elf-objdump` command to create the **objdump** file. This command copies the processor's **.elf** file to the **objdump** file's human-readable format that contains C/C++ code fragments, symbolic function names, assembly instructions, and address locations.

The `nios2-elf-objdump` command-line executable is included as a part of the Nios II Embedded Design Suite. You can configure the tool with a series of command line options. For a list of conversion options for the `nios2-elf-objdump` command, in a Nios II command shell, type the following command:

```
nios2-elf-objdump --help ←
```

Although the **objdump** file contains vast amounts of information decoded from the **.elf** file, the Nios II processor's instructions appear one per line in this file, in the following format:

```
<Address>: <op-code> <Assembly Mnemonic>
```

For example, a valid instruction is:

```
200a8c0: e0800417 ldw r2,16(fp)
```

## Saving and Converting Captured Data

You can save any data captured by the Nios II plug-in with the SignalTap II data log feature. The Nios II plug-in data sets are stored with the processor's **.elf** file information. To enable data logging, turn on the **Data Log** option in the SignalTap II window.

The Nios II plug-in also supports the SignalTap II data conversion feature. To export captured data, on the File menu, click **Export** and specify the **File Name**, the **Export Format**, and the **Clock Period**.

## Tutorial: Using the Nios II Plug-In

This tutorial shows you how to use the Nios II plug-in for the SignalTap II Logic Analyzer to trace the transmission of an RS-232 character on a Nios II development board. The tutorial explains how to set up and configure the Nios II plug-in, how to use the plug-in during the Nios II software debug cycle (Nios II IDE), and how to perform an analysis of the data captured using the **.elf** file as a reference.

## Hardware and Software Requirements

To complete this tutorial you must have the following:

- Quartus II software version 8.0 or higher—Both the Quartus II Web Edition and the fully licensed version works with the example design.
- Nios II Embedded Development Suite (EDS) version 8.0 or higher

- Nios II Development Kit—You can use any Altera Nios II development kit that supports the **full\_featured** example design. The following development kits support this design:
  - Stratix® II Edition
  - Cyclone® II Edition
- RS-232 UART cable—The tutorial can be run without this cable; however, without the cable, the output from the Nios II development board does not appear on the Nios II IDE console.
- Software file—Download the file **signal\_tap\_test.c** to a folder on your hard drive. This software file can be found on the Altera literature pages with this application note, at [www.altera.com/literature/lit-nio2.jsp](http://www.altera.com/literature/lit-nio2.jsp).
- JTAG ByteBlaster or USB-Blaster cable

### Hardware Project Set-Up

You must first create the system hardware containing the Nios II processor and configure the Quartus II project for the Nios II plug-in.

#### *Setting Up the Development Board*

Make sure that your Nios II development board has power and is connected to your workstation through the JTAG download cable. Additionally, verify that your Nios II development board's RS-232 UART port is connected to a working serial port on your workstation.



While having the Nios II development board connected to your workstation with a serial cable is ideal, this tutorial can be run without it. However, without the serial cable, you cannot see the character being received by the host workstation.

#### *Configuring the Quartus II Software*

To configure the Quartus II software, perform the following steps:

1. Copy the Nios II **full\_featured** example design for your particular board to a location in which you can edit it. The example design is located in the following directory:

```
$$SOPC_KIT_NIOS2/examples/verilog/  
<your-nios2-board-type>/full_featured
```

2. Open the Quartus II software.
3. On the **File** menu, click **Open Project**.

4. Browse to the location where you copied the **full\_featured** example design. This is your project directory.
5. Select the `<your-nios2-board-type>_full_featured.qpf` project file.
6. Click **Open**.

### *Generating Nios II Hardware*

You must regenerate the **full\_featured** example design using SOPC Builder before it will properly compile in the Quartus II software.

1. On the Tools menu, click **SOPC Builder**. SOPC Builder appears.
2. In SOPC Builder, click **Generate**. This action rebuilds the hardware for the **full\_featured** example design.



This operation may take a few minutes to complete.

3. After generation completes, click **Exit**.

### **Nios II Software Set-Up**

You can now use the Nios II command shell to compile your software application and generate the **.elf** and **objdump** files. After the files are generated, you can identify the trigger location that you want to use for the Nios II plug-in.

### *Building the Nios II Software*

To build the software application and generate an **objdump** file, perform the following steps:

1. Open a Nios II command shell.
2. Change to your project directory, the new copy of the **full\_featured** example design directory.
3. Create new directories in which to place software files later, by typing the following command:

```
mkdir software software/app software/bsp ←
```

4. Copy the C source file **signal\_tap\_test.c** to the **software/app** directory.
5. Change to the new directory **software/bsp**.

6. Create the Board Support Package (BSP), by typing the following command:

```
nios2-bsp hal . ../.. ↵
```

7. Change to the new directory **software/app**.
8. Create the application project makefile by typing the following command:

```
nios2-app-generate-makefile \  
  --bsp-dir ../bsp \  
  --src-files ./signal_tap_test.c \  
  --elf-name signal_tap_test.elf \  
  --set CREATE_OBJDUMP 1 \  
  --set OBJDUMP_INCLUDE_SOURCE 1 ↵
```



When `CREATE_OBJDUMP` is set to 1, building the project generates an **objdump** file.

9. To build the project, type the following command:

```
make ↵
```

This step creates the **.elf** and **objdump** files.

### *Importing the Software Project to the Nios II IDE*

To debug your project in the Nios II IDE, you must first import it. To import your software project to the Nios II IDE, perform the following steps:

1. Open the Nios II IDE.
2. On the File menu, click **Import**. The **Import** dialog box appears.
3. Expand the **Altera Nios II** folder.
4. Click **Existing Nios II software build tools project or folder into workspace**.
5. Click **Next**. The Import wizard appears.
6. Browse to your **software/app** directory.
7. Click **OK**. The Import wizard fills the project name and path fields.

8. In the **Project name** field, change the project name to `signal_tap_test` by typing `signal_tap_test`.
9. Click **Finish**.

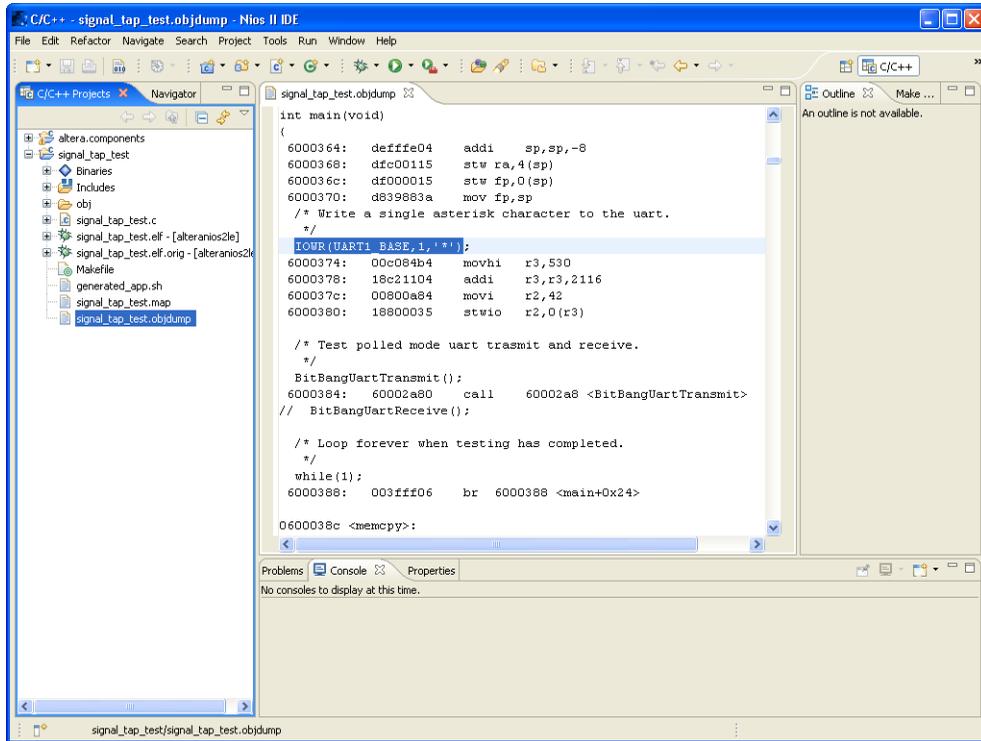
### *Finding the Instruction Trigger Condition*

The Nios II plug-in should trigger on the first write to the UART peripheral, immediately after the entry point to the `main()` function. To identify the address of the assembly instruction that writes to the UART's transmit register, use the **objdump** file generated during software compilation.

The **objdump** file format contains a human-readable version of the program you just compiled. It shows the C instructions with the corresponding assembly instructions and their locations in the Nios II processor's address space. By locating the C instruction responsible for the write operation to the UART's transmit register, and then examining the assembly instructions that correspond to the C instruction, you can retrieve the address for the assembly instruction responsible for the write to the UART's transmit register. Perform the following steps:

1. Expand the **signal\_tap\_test** folder in the **Nios II C/C++ Projects** tab.
2. Right-click on the **signal\_tap\_test.objdump** file and click **Open**.
3. On the Edit menu, click **Find/Replace**.
4. In the **Find** field of the **Find/Replace** dialog box, type `IOWR(UART1_BASE, 1, '*')`. This string is the C function call to write the UART's transmit register.
5. Click **Find**. The line containing `IOWR(UART_BASE, 1, '*')` is highlighted, as shown in [Figure 1](#).

Figure 1. Nios II Objdump View



6. Scroll down a few lines until you find the line containing the `stwio r2,0(r3)` assembly instruction, and note the address to the left of the instruction. This instruction corresponds to the Nios II processor's write to the UART's transmit register. In this example, the address for this instruction is `0x6000380`. However, in your particular case this address may be different. You can use the address of this instruction as a trigger for the Nios II plug-in to begin capturing instruction trace data.

In this example, when the Nios II processor executes the assembly instruction `stwio r2,0(r3)`, it writes to the UART transmit control register, which then transmits a character. You can identify the write instruction from the following indicators:

- The instruction appears in the block of assembly instructions that correspond to the C instruction `IOWR(UART1_BASE,1,'*')`.
- The other assembly instructions contained in the block perform operations only on the Nios II processor's registers.

- The assembly language instruction `stwio` performs a store-word operation to a peripheral.



For information about assembly language instructions, refer to the *Instruction Set Reference* chapter of the *Nios II Processor Reference Handbook*.

## Configuring the Hardware Project

Next, configure the Nios II plug-in for operation in the SignalTap II Logic Analyzer.



In the project provided in the **full\_featured** directory, Incremental Compilation is already turned off.

Perform the following actions in the Quartus II software:

1. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.



This operation may take a few minutes to complete.

2. On the Tools menu, click **SignalTap II Logic Analyzer**.
3. Right-click in the SignalTap II node list. Point to **Add Nodes with Plug-In** and click **Nios II**. The **Select Hierarchy Level** dialog box appears.
4. In the **Select Hierarchy Level** dialog box, select `<your-nios2-board-type>_sopc_instance | cpu:the_cpu`.
5. Click **OK**. The **Plug-In Options** dialog box appears.
6. In the **Plug-In Options** dialog box, click the browse button next to the **Setting** text field.
7. In the **Select File** dialog box, browse to the location of the **.elf** file you compiled in the Nios II command shell. This **.elf** file should be present in your **software/app** directory and be named **signal\_tap\_test.elf**. Select the file.
8. Click **Open**. The **Select File** dialog box closes.
9. Click **OK**. The **Plug-In Options** dialog box closes.

10. In the **Trigger Conditions** column for the Nios II plug-in, enter the address that corresponds to the write to the UART's transmit register. In this example, the address is 0x6000380 for the trigger condition, as shown in [Figure 1 on page 16](#).
11. Add the RS-232 UART's transmit data signal to the node list by performing the following steps:
  - a. Right-click on the node list pane and click **Add Nodes**.
  - b. Ensure that the **Look in** field is set to `|<your-nios2-board-type>_full_featured|` and the **Named** field contains only an asterisk (\*).
  - c. Ensure that the **Filter** field is set to **SignalTap II: pre-synthesis**.
  - d. Click **List**.
  - e. In the **Nodes Found** pane, double-click the node `txd_from_the_uart1`. The node is added to the **Selected Nodes** pane.
  - f. Click **OK**.
12. In the Signal Configuration pane, click the browse button next to the **Clock** text field to bring up the **Node Finder** dialog box.
13. In the **Node Finder** dialog box, perform the following steps:
  - a. Click the browse button next to the **Look in** text field. The **Select Hierarchy Level** dialog box appears.
  - b. In the **Select Hierarchy Level** dialog box, expand the hierarchy list under `<your-nios2-board-type>_full_featured_sopc_instance`.
  - c. In the expanded hierarchy list, click the entity **pll:the\_pll**.
  - d. Click **OK** to close the list.
  - e. Ensure that the **Filter** field is set to **SignalTap II: pre-synthesis**.
  - f. In the Node Finder, click **List** to list all of the nodes.
  - g. Double-click the node `c0` to add it to the **Selected Nodes** field.
  - h. Click **OK**. The **Node Finder** dialog box closes.

14. In the Signal Configuration pane, set the **Sample Depth** field to 256.
15. In the Quartus II window, on the Processing menu, click **Start Compilation** to build the hardware for the project. If you are prompted to save the SignalTap II instance and enable the SignalTap II Logic Analyzer for this project, click **Yes**.



You must save the **.stp** file in your project directory.



The Quartus II hardware design may take a few minutes or longer to compile.

## Running the Trace Capture Session

The next step is to use the generated system to run a trace capture session with the Nios II plug-in by performing the following steps:

1. In the SignalTap II window, in the JTAG Chain Configuration pane, perform the following steps:
  - a. On the Hardware menu, select the programming cable that is connected to your Nios II development board.
  - b. In the **SOF Manager** field, click the browse button.
  - c. In the **Select Programming File** dialog box, browse to your project directory and select the **.sof** file `<your-nios2-board-type>_full_featured.sof`.
  - d. Click **Open**. The **Program Device** button is now available.
  - e. Click the **Program Device** button to download the **.sof** file to the FPGA. 
2. In the SignalTap II window, in the Instance Manager pane, click the **Run Analysis** button to start the logic analyzer capture session. 

The analysis should stop with the **Status** in the Instance Manager pane set to **Waiting for trigger**.

3. In the Nios II IDE, select the project **signal\_tap\_test** in the Projects list.
4. In the Nios II IDE, on the Run menu, click **Debug**. The **Debug** dialog box appears.

5. In the **Debug** dialog box, perform the following steps:
  - a. Click **Nios II Hardware** in the left-hand pane.
  - b. Click the **New launch configuration** button to create a new debug launch configuration. 
  - c. In the **Project** text field, type `signal_tap_test`.
  - d. In the Target hardware pane, click the **Browse** button.
  - e. In the **SOPC Builder System PTF File** dialog box, browse to your project directory and select `<your-nios2-board-type>_full_featured_socp.ptf`.
  - f. Click **Open**.
  - g. On the **Target Connection** tab, set the **Nios II Terminal communication device** to `uart1` and verify that the **Host COM port** field contains the correct setting for the serial port connected to your development board.

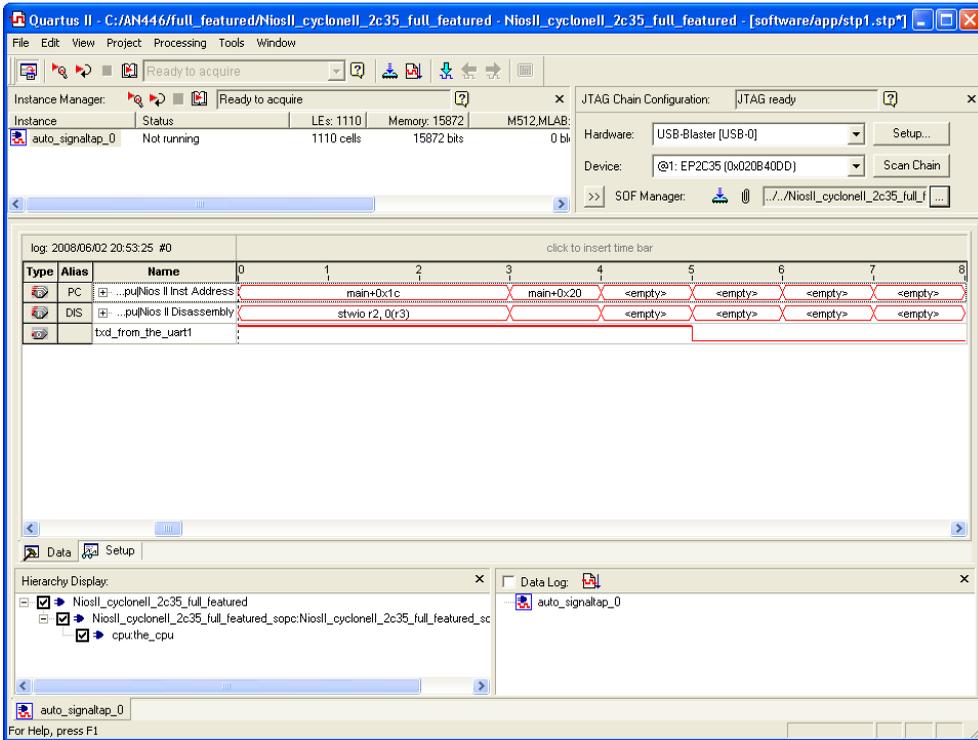
This step is necessary if you want to see the characters coming through the serial port. If you omit this step, the SignalTap II Logic Analyzer is able to capture the data, but the characters do not appear in the Nios II IDE console window.
  - h. On the **Debugger** tab, in the **Breakpoints at Start-up** field, turn on the **Break at program entry point** option. Turn off the other options.
  - i. Click **Apply**, followed by **Debug**. The `.elf` file is downloaded to the board, followed by a launch of the debugger.
  - j. In the **Confirm Perspective Switch** dialog box, click **Yes**.
6. In the Nios II IDE, in the Debug pane, click the **Resume** button to start the Nios II processor execution. 

If the UART port is connected, the following message appears in the console window:

```
*BIT BANG
```

7. Check the SignalTap II window to verify that the capture session has terminated because the Nios II plug-in's trigger conditions were met. Your SignalTap II window should resemble [Figure 2](#).

**Figure 2. SignalTap II Post Capture Window**



## Analyzing the Captured Data

You can now analyze the data captured by the Nios II plug-in and compare it to the **objdump** of the Nios II **.elf** file. The data samples from the Nios II plug-in should correspond exactly to the assembly language program listed in the **signal\_tap\_test.objdump** file.

[Figure 2](#) demonstrates the following results:

- The 0th sample captured by the Nios II plug-in matches the trigger condition you set up, capturing instruction `stwio r2, 0(r3)`.

- The RS-232 UART's transmit data signal began two clock cycles after the `stwio` instruction, which corresponds to the Nios II IDE writing to the transmit register of the UART. The full eight-bit, serial transmit sequence for the UART is not captured, because the SignalTap II Logic Analyzer filled its sample buffers before the UART could complete the transmission of the asterisk character. Only the first bit of the eight-bit character sequence is captured.

## Conclusion

As FPGA designs continue to increase in size and complexity, with increasing numbers of embedded processors, peripherals, buses, and bridges, designers need more comprehensive debugging tools to reduce system development time. The Nios II plug-in for the SignalTap II Embedded Logic Analyzer enables the capture of a Nios II processor's program execution. Because the Nios II plug-in works together with the SignalTap II Logic Analyzer, the instruction trace from the Nios II processor is captured with other hardware events. The display of the processor instructions allows you to find system design problems efficiently.

## Referenced Documents

This application note references the following documents:

- *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*
- *Nios II Software Developer's Handbook*
- *Nios II Core Implementation Details* chapter of the *Nios II Processor Reference Handbook*
- *Instruction Set Reference* chapter of the *Nios II Processor Reference Handbook*

## Document Revision History

Table 1 shows the revision history for this application note.

<b>Date and Document Version</b>	<b>Changes Made</b>	<b>Summary of Changes</b>
June 2008 v1.2	<p>This revision incorporates the following changes:</p> <ul style="list-style-type: none"> <li>• Updated the “<a href="#">Tutorial: Using the Nios II Plug-In</a>” section for the Nios II software build flow, including removal of many Nios II IDE screenshots.</li> <li>• Replaced the Count Binary design example with the <code>signal_tap_test</code> example.</li> </ul>	Updated document for the Quartus II software and Nios II EDS v8.0.
October 2007 v1.1	<p>This revision incorporates the following changes:</p> <ul style="list-style-type: none"> <li>• Replaced all references to <code>bit_bang_uart.c</code> with <code>signal_tap_test.c</code></li> <li>• Replaced the associated design file <code>bit_bang_uart.c</code> with the file <code>signal_tap_test.c</code></li> </ul>	Updated document for the Quartus II software and Nios II EDS v7.2.
May 2007 v1.0	Initial release.	—



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 © 2008 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 relying on any published information and before placing orders for products or services.



I.S. EN ISO 9001