

For Quartus II 11.0

# 1 Introduction

This tutorial presents an introduction to the Altera Monitor Program, which can be used to compile, assemble, download and debug programs for Altera's Nios II processor. The tutorial gives step-by-step instructions that illustrate the features of the Monitor Program.

Altera Monitor Program

Tutorial

The Monitor Program is a software application which runs on a host PC, and communicates with a Nios II hardware system on an FPGA board. The Monitor Program is compatible with Microsoft Windows operating systems, including XP, Vista, and Windows 7. It allows the user to assemble/compile a Nios II software application, download the application to a Nios II hardware system, and then debug the running application. The Monitor Program provides features that allow a user to:

- Set up a Nios II project that specifies a desired hardware system and software program
- Download the Nios II hardware system onto an FPGA board
- Compile software programs, specified in assembly language or C, and download the resulting machine code into the Nios II hardware system
- Disassemble and display the Nios II machine code stored in memory
- Run the Nios II processor, either continuously or by single-stepping instructions
- Examine and modify the contents of Nios II registers
- Examine and modify the contents of memory, as well as memory-mapped registers in I/O devices
- Set breakpoints that stop the execution of a program at a specified address, or when certain conditions are met
- Perform terminal input/output via a JTAG UART component in the Nios II hardware system
- Develop Nios II programs that make use of device driver functions provided through Altera's Hardware Abstraction Layer (HAL)

The process of downloading and debugging a Nios II program requires the presence of an FPGA board to implement the Nios II hardware system. In this tutorial it is assumed that the reader has access to the Altera DE2 Development and Education board, connected to a computer that has Quartus II (version 11.0) and Nios II Embedded Design Suite (EDS) software installed. Although a reader who does not have access to an FPGA board will not be able to execute the Monitor Program commands described in the tutorial, it should still be possible to follow the discussion.

The screen captures in this tutorial were obtained using version 11.0 of the Monitor Program; if other versions of the software are used, some of the images may be slightly different.

### 1.1 Who Should use the Monitor Program

The Monitor Program is intended to be used in an educational environment by professors and students. The Monitor Program is not intended for commercial use.

## 2 Installing the Monitor Program

The Monitor Program is released as part of Altera's University Program Design Suite (UPDS). Before the UPDS can be installed on a computer, it is necessary to first install Altera's Quartus II CAD software (either the Web Edition or Subscription Edition) and the Nios II Embedded Design Suite (EDS). This release (11.0) of the Monitor Program can be used only with version 11.0 of the Quartus II software and Nios II EDS. This software can be obtained from the *Download Center* on Altera's website at www.altera.com. To locate version 11.0 of the software for downloading, it may be necessary to click on the item *All Design Software* in the section of the download page labeled *Archives*. Once the Quartus II software and Nios II EDS are installed, then the Altera UPDS can be installed as follows:

- 1. Install the Altera UPDS from the University Program section of Altera's website. It can be found by going to *www.altera.com* and clicking on *University Program* under *Training*. Once in the University Program section, use the navigation links on the page to select *Educational Materials > Software Tools > Altera Monitor Program*. Then click on the *EXE* item in the displayed table, which links to an installation program called *altera\_upds\_setup.exe*. When prompted to Run or Save this file, select Run.
- 2. The first screen of the installer is shown in Figure 1. Click on the Next button.



Figure 1. Altera UPDS Setup Program.

3. The installer will display the License Agreement; if you accept the terms of this agreement, then click | Agree

to continue.

4. The next screen, shown in Figure 2, lists the components that will be installed, which include the Monitor Program software and University Program IP Cores. The University Program IP Cores provide a number of I/O device circuits that are used in Nios II hardware systems. Click Next to open the window in Figure 3, which shows the path where the Altera University Program Material will be installed. If the path is incorrect, cancel the installation, run the Quartus II 11.0 software, and re-start the installer.

🔋 Altera University Program Design Suite Setup					
c	oose Components hoose which features of Altera University Program Desi ou want to install.	gn Suite			
Check the components you war install. Click Next to continue.	nt to install and uncheck the components you don't wan	t to			
Select the type of install:	Full	~			
Or, select the optional components you wish to install:	Quartus II Material  Simulation Tools  Oesign Examples  Nos II Material  Material  Miss II Computer Systems  Description				
Space required: 168.2MB	Position your mouse over a component to see its description.				
Nullsoft Install System v2,46 ——	< Back Next > C	ancel			

Figure 2. The components that will be installed.

💭 Altera University Program Design Suite Setup
Installation Confirmation Confirm the installation path.
Click Install to proceed with the installation. Your Quartus II root directory is set to: D:\altera\11.0 As specified in the registry entry: HKEY_LOCAL_MACHINE\SOFTWARE\Altera Corporation\Quartus\Quartus Install Directory The Altera University Program Material needs to be installed under your current Quartus II path. If the path above is incorrect, cancel this installation, run the Quartus II 11.0 software (which will update the necessary registry entries and environment variables) and re-start this installer.
Nullsoft Install System v2,46

Figure 3. The path where components will be installed.

5. The installer is now ready to begin copying files. Click Install to proceed and then click Next after the installation has been completed. If you answered Yes when prompted about placing a shortcut on your Windows Desktop, then an icon is provided on the Desktop that can be used to start the Monitor Program.

- 6. Now, the Altera's Unveristy Program Design Suite is successfully installed on your computer, so click Finish to finish the installation.
- 7. Should an error occur during the installation procedure, a pop-up window will suggest the appropriate action. Possible errors include:
  - Quartus II software is not installed or the Quartus II version is incorrect (only version 11.0 is supported by this release of the Monitor Program).
  - Nios II EDS software is not installed or the version is incorrect (only version 11.0 is supported).

Note that if the Quartus II software is reinstalled at some future time, then it will be necessary to re-install the Monitor Program at that time.

# 3 Main Features of the Monitor Program

Each Nios II software application that is developed with the Altera Monitor Program is called a *project*. The Monitor Program works on one project at a time and keeps all information for that project in a single directory in the file system. The first step is to create a directory to hold the project's files. To store the design files for this tutorial, we will use a directory named *Monitor\_Tutorial*. The running example for this tutorial is a simple assembly language program that controls some lights on a DE2 board.

Start the Monitor Program software, either by double-clicking its icon on the Windows Desktop or by accessing the program in the Windows Start menu under Altera > University Program > Altera Monitor Program. You should see a display similar to the one in Figure 4. This display consists of several windows that provide access to all of the features of the Monitor Program, which the user selects with the computer mouse. Most of the commands provided by the Monitor Program can be accessed by using a set of menus that are located below the title bar. For example, in Figure 4 clicking the left mouse button on the File command opens the menu shown in Figure 5. Clicking the left mouse button on the entry Exit exits from the Monitor Program. In most cases, whenever the mouse is used to select something, the left button is used. Hence we will not normally specify which button to press.

For some commands it is necessary to access two or more menus in sequence. We use the convention Menu1 > Menu2 > Item to indicate that to select the desired command the user should first click the mouse button on Menu1, then within this menu click on Menu2, and then within Menu2 click on Item. For example, File > Exit uses the mouse to exit from the system. Many commands can alternatively be invoked by clicking on an icon displayed in the Monitor Program window. To see the command associated with an icon, position the mouse over the icon and a tooltip will appear that displays the command name.

It is possible to modify the organization of the Monitor Program display in Figure 4 in many ways. Section 10 shows how to move, resize, close, and open windows within the Monitor Program display.

### 3.1 Creating a Project

To start working on a Nios II software application we first have to create a new project, as follows:

🛹 Altera Monitor Program [Nios II]		
<u>File S</u> ettings <u>A</u> ctions <u>Wi</u> ndows <u>H</u> elp		
Disassembly	_ × Registe	
Goto instruction Address (hex) or symbol name:	Go Hide Reg Valu	Je
Terminal _ ×	Info & Errors	_ ×
	Info & Errors / GDB Server /	

Figure 4. The main Monitor Program display.



Figure 5. An example of the File menu.

Select File > New Project to open the New Project Wizard, as illustrated in Figure 6. This Wizard presents
a sequence of screens for defining a new project. Each screen includes a number of dialogs, as well as a
message area at the bottom of the window. The message area is used to display error and information messages
associated with the dialogs in the window. Double-clicking the mouse on an error message moves the cursor
into the dialog box that contains the source of the error.

In Figure 6 we have specified the file system directory *D*:\*Monitor\_Tutorial* and the project name *Monitor\_Tutorial*. For simplicity, we have used a project name that matches the directory name, but this is not

Specify a project name and directo	ч <b>у</b>
Project directory:	
D:\Monitor_Tutorial	Browse
Project name:	
Monitor_Tutorial	

Figure 6. Specifying the project directory and name.

#### required.

The file system directory specified for the project must already exist. To select a directory by browsing through the file system, click on the **Browse** button. Note that a given directory may contain at most one project.

2. Click Next to advance to the window shown in Figure 7, which is used to specify a Nios II hardware system. Nios II-based systems are described by a *.ptf* file, which is generated by the Altera SOPC Builder tool when the system is created. More information about creating systems using SOPC Builder can be found in the tutorial called *Introduction to the Altera SOPC Builder*, available in the University Program section of Altera's website. An optional *.sof* file, if specified, represents the FPGA circuit that implements the Nios II-based system; this file can be downloaded into the FPGA chip on the board that is being used.

The drop-down list on the Select a system pane can be used to choose a pre-built Nios II computer system provided with the Monitor Program, or a <Custom System> created by the user. Both the *.ptf* and the *.sof* files are automatically filled in by the Monitor Program if a pre-built system is selected. However, if <Custom System> is selected, then the files need to be specified manually in the System details pane. Section 5 shows how to use the Monitor Program with a Custom system.

As depicted in Figure 7, select the pre-built system named *DE2 Basic Computer*. In the top right corner of the screen there is a **Documentation** button. Clicking on this button opens a user guide that provides all information needed for developing Nios II programs for the DE2 Basic Computer, such as the memory map for addressing all of the I/O devices in the system. This file can also be accessed at a later time by using the

	stem	
elect a system	·	
DE2 Basic Comp	uter 🗸 🗸 🗸	umentatio
computer organi	ed the DE2 Basic Computer, is intended to be used as a platform for introductory experi zation and embedded systems. To support these beginning experiments, the system cor : a processor, memory, and some simple I/O peripherals.	
ystem details-		
System description	on (PTF) file:	
0.1\University_P	Program\NiosII_Computer_Systems\DE2\DE2_Basic_Computer\verilog\nios_system.ptf	Browse.
Quartus II progr	amming (SOF) file (optional):	
<pre>&gt;rsity_Program\N</pre>	liosII_Computer_Systems\DE2\DE2_Basic_Computer\verilog\DE2_Basic_Computer.sof	Browse.
Program can be	resents the FPGA programming file for the Nios II system. If it is specified here, then the used to download this programming file onto the board. Otherwise, the system will need ng some other method (for example, by using Quartus II).	

Figure 7. Specifying the Nios II hardware system.

command Settings > System Settings and then clicking on the Documentation button.

- 3. Click Next to advance to the screen in Figure 8, which is used to specify the program source files that are associated with the project. The Program Type drop-down list can be used to select one of the following program types:
  - Assembly Program: allows the Monitor Program to be used with Nios II assembly language code
  - C Program: allows the Monitor Program to be used with C code
  - Program with Device Driver Support: this is an advanced option, which can be used to build programs that make use of device driver software for the I/O devices in the Nios II hardware system. Programs that use this option can be written in either assembly, C, or C++ language (or any combination). More information about writing programs that use device drivers can be found in Section 9.
  - ELF or SREC File: allows the Monitor Program to be used with a pre-compiled program, in ELF or SREC format
  - No Program: allows the Monitor Program to connect to the Nios II hardware system without first loading a program

For this example, set the program type to Assembly Program. When a pre-built Nios II computer system has been selected for the project, as we did in Figure 7, it is possible to click on the selection Include a



Figure 8. Selecting a program type and sample program.

sample program with the project. As illustrated in Figure 8 several sample assembly language programs are available for the DE2 Basic Computer. For this tutorial select the program named *Getting Started*. Click Next to advance to the screen in Figure 9. When a sample program has been selected, the source code file(s) associated with this program are listed in the Source files box. In this case, the source file is named *getting\_started.s*; this source file will be copied into the directory used for the project by the Monitor Program. If a sample program is not used, then it is necessary to click the Add button and browse to select the desired source file(s).

Figure 9 shows that it is possible to specify the label in the assembly language program that identifies the first instruction in the code. In the *getting\_started.s* file, this label is called *\_start*, as indicated in the figure.

4. Click Next to advance to the window in Figure 10. This dialog is used to specify the connection to the FPGA board, the Nios II processor that should be used (some hardware systems may contain multiple processors), and the terminal device. The Host connection drop-down list contains the physical connection links (such as cables) that exist between the host computer and any FPGA boards connected to it. The Nios II processors available in the system are found in the Processor drop-down list, and all terminal devices connected to the selected processor are displayed in the Terminal device drop-down list. We discuss terminal devices in section 6.

For this tutorial, accept the default values that are displayed in Figure 10.

5. Click Next to reach the final screen for creating the new project, shown in Figure 11. This screen is used

Source files —					
First source file	is used to determin	ne the name of the bi	nary program file.		
D:\Monitor_Tut	corial\getting_start(	ed.s			Add Remove
Program optio					Up Down
Start symbol:	_start				
ource files highlig	hted in blue are sar	mple program files, w	hich will be create	d in the project direc	tory.

Figure 9. Specifying source code files.

to specify memory settings that are needed for compiling and linking the program. Nios II programs are stored in a format that supports *sections*, which are used to divide a program into multiple parts, such as an executable code section, called *.text*, and a data section, called *.data*. The partitioning of the program into different sections is performed by the linker.

As illustrated in Figure 11, choose the SDRAM chip in the DE2 Basic Computer as the storage location for both the *.text* and *.data* sections, and use the value 0 for the offset into the memory for both sections. When the offsets for both sections are identical the linker automatically places the *.data* section immediately after the *.text* section.

The *getting\_started.s* file shows how to include *.text* and *.data* directives in an assembly language program. These directives should be included in a program when it is desirable to separate the program text and data. For example, it may be desirable to place each section into a different memory device. Note that it is also possible to use *.org* directives in an assembly language program to specify section addresses. However, this approach can cause the machine code files generated by assembling the program to be very large if there is a wide gap in addresses between the *.text* and *.data* sections.

For the sample program selected for this tutorial it is not necessary to make use of the *.text* and *.data* sections. However, other programs, such as those that use interrupts, must utilize these sections to avoid linking errors. An example of the appropriate setting when interrupts are used in a program is given in section 8.

Click Finish to complete the creation of the new project. At this point, the Monitor Program displays the

🥔 New Project V	Vizard 🛛 🔀
Specify syst	tem parameters
_ System parame	ters
Host connection:	USB-Blaster [USB-0]   Refresh
Processor:	CPU -
	Reset vector address: 0x0
	Exception vector address: 0x20
Terminal device:	JTAG_UART
	< <u>Back</u> Next > Einish Cancel

Figure 10. Specifying system settings.

prompt shown in Figure 12. Clicking Yes instructs the Monitor Program to attempt to download the Nios II system associated with the project onto the FPGA board. It is also possible to download the system at a later time by using the Monitor Program command Actions > Download SOPC Builder System.

#### 3.1.1 Downloading a Nios II Hardware System

When downloading a Nios II hardware system onto an FPGA board, it is important to consider the type of license that is included in the hardware system for the processor. The Nios II processor uses a licensing scheme that provides two modes of operation: 1. an evaluation mode that allows the processor to be used with some restrictions when no license is present, and 2. a normal mode that allows unrestricted use when a license is present. Nios II licenses can be purchased from Altera, and are also available on a donated basis through the University Program. The prebuilt computer systems provided with the Monitor Program, such as the DE2 Basic Computer, include a Nios II processor that has a license. However, if other systems are being used with the Monitor Program, then it is possible that a license is not present, and the Nios II processor may be used in the evaluation mode. In this case it is necessary to use a different scheme, which is described in section 5, to download the Nios II hardware system onto the FPGA board and activate the evaluation mode.

ocessor's reset and exce	ption vectors (read-only)	
teset vector address (hex):	0	
exception vector address (he	x): 20	
emory options		
addresses can be in the sam .text and .data sections do r	rting addresses of sections identified by .text and .da e or in different memories (on-chip, SDRAM,). They ot overlap with other sections, such as .reset and .ex ddress, the .data section will be placed right after the	can be used to ensure that the ceptions. If .text and .data are
.text section		
Memory device:	SDRAM/s1 (0h - 7fffffh)	-
Start offset in device (hex):		0
.data section		
Memory device:	SDRAM/s1 (0h - 7fffffh)	-
Start offset in device (hex):		0

Figure 11. Specifying memory settings.

Downloa	ad SOPC Builder System - Prompt 🛛 🛛 🔀
?	Would you like to download the system associated with this project onto the board? If so, make sure that the board is connected via the correct cable and is powered up.
	Yes No

Figure 12. Download the Nios II system.

### 3.2 Compiling and Loading the Program

After successfully creating a project, its software files can be assembled/compiled and downloaded onto the FPGA board using the following commands:

- Actions > Compile menu item or icon: compiles the source files into an ELF and SREC file. Build warnings and errors will show up in the Info & Errors window. The generated ELF and SREC files are placed in the project's directory.
- Actions > Load menu item or  $\stackrel{1}{\downarrow}$  icon: loads the compiled SREC file onto the board and begins a debugging session in the Monitor Program. Loading progress messages are displayed in the Info & Errors window.

• Actions > Compile & Load menu item or icon: performs the operations of both compilation and loading.

Our example project has not yet been compiled, so it cannot be loaded (the Load option is disabled). Click the Actions > Compile & Load menu item or click the icon to begin the compilation and loading process. Throughout the process, messages are displayed in the Info & Errors window. The messages should resemble those shown in Figure 13.

```
Info & Errors
Info: Quartus II Programmer was successful. O errors, O warnings
    Info: Peak virtual memory: 121 megabytes
    Info: Processing ended: Fri Jan 14 15:11:59 2011
    Info: Elapsed time: 00:00:02
    Info: Total CPU time (on all processors): 00:00:00
Compiling source files...
nios2-elf-as --gstabs -I C:/altera/10.1/nios2eds/components/altera_nios2/sdk/inc -I D:/Monitor_Tutorial D:/Mo
Linking...
nios2-elf-ld --defsym nasys_program_mem=0x0 --defsym nasys_data_mem=0x0 --section-start .exceptions=0x20 --se
ELF generated at D:\Monitor Tutorial\getting started.elf.
nios2-elf-objcopy -0 srec D:/Monitor_Tutorial/getting_started.elf D:/Monitor_Tutorial/getting_started.srec
SREC generated at D:\Monitor_Tutorial\getting_started.srec.
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Reading System ID at address 0x10002020: verified
Initializing CPU cache (if present)
0K
Downloading 00000000 ( 0%)
Downloaded 1KB in 0.0s
Verifying 00000000 ( 0%)
Verified OK
Connection established to GDB server at localhost:2399
Symbols loaded.
Source code loaded.
INFO: Program Trace not enabled, because trace requires the Nios II processor to be configured with JTAG Debug
4
```

Figure 13. Compilation and loading messages.

After successfully completing this step, the Monitor Program display should look similar to Figure 14. At this point, the Nios II processor is halted at the first instruction of the program. The main part of the display in Figure 14 is called the *Disassembly* window. We discuss this window in detail in section 3.4. It shows the source code of the program, as well as a disassembled view of the corresponding Nios II machine code that is stored in memory. In the figure, the first line of source code is the instruction movia r15, 0x1000040. This is a *pseudo-instruction*, rather than a native Nios II instruction<sup>1</sup>. The Disassembly window shows immediately below this pseudo instruction the corresponding Nios II machine code, which is stored at address 0. The movia operation is implemented by using two Nios II machine instructions: orhi and addi. As illustrated in the figure, for each line of code from the project's assembly language source code files, the Monitor Program displays the source code along with its corresponding disassembled machine code that is stored in memory. In most cases the source code and disassembled machine code are the same, but for some operations, like pseudo instructions, they are different.

<sup>&</sup>lt;sup>1</sup>More information about Nios II instructions and pseudo-instructions can be found in the tutorial *Introduction to the Altera Nios II Soft Processor*, available in the University Program section of Altera's website.

le <u>S</u> ettings	Actions <u>W</u> indo	ws <u>H</u> elp	1 🖑					
sassembly					_ × [	Registers	; _	>
oto instruction	Address (hex)	or symbol name:		Go	Hide	Reg	Value	
		· .				pc	0x00000000	4 H
		.global	_start			zero	0x0000000	5 H.
		_start:				rl	0x0000000	- 8
						r2	0x00000000	- 8
		/* init		cesses of parallel port	65 -7	r3	0x00000000	- 1
		movia	r15, 0x100	000040 /* SW sl:	ruer swrtten ba	r4	0x00000000	-118
		_start:				r5 r6	0x00000000 0x00000000	н.
0x00000000	03c40034	orhi	r15, zero, 0x10	000		r6 r7	0x00000000	-11
0x00000004	7bc01004	addi	r15, r15, 0x40			r8	0x00000000	н.
		movia	r16, 0x100	000000 /* red LI	TD hase addres	r9	0x00000000	н.
0x0000008	04040034	orhi	r16, zero, 0x10	000		r10	0x00000000	11
0x000000c	84000004	addi	r16, r16, 0x0			r11	0x00000000	- 1
		movia	r17, 0x100	000050 /* pushbu	state and TZEXY last and	r12	0x00000000	- 1
0x00000010	04440034	orhi	r17, zero, 0x10	000		r13	0x00000000	- 1
0x00000014	8c401404	addi	r17, r17, 0x50			r14	0x00000000	- 1
		movia	r18, 0x100	000010 /* green	LED base addr	r15	0x00000000	j
0x00000018	04840034	orhi	r18, zero, 0x10	000	-	r16	0x00000000	í
	0.4000.40.4	- 444				r17	0x00000000	j
			/ /		•	r18	0x00000000	
Disassembly / Br	eakpoints / Mer	mory (Watches	( <u>Trace</u> )			r19	0x00000000	
rminal			_ ×	Info & Errors			-	
'AG UART lin JSB-0]", dev		-	le "USB-Blaster	Verified OK Connection establish Symbols loaded. Source code loaded. INFO: Program Trace				

Figure 14. The Monitor Program window after loading the program.

#### 3.2.1 Compilation Errors

During the process of developing software, it is likely that compilation errors will be encountered. Error messages from the Nios II assembler or from the C compiler are displayed in the lnfo & Errors window. To see an example of a compiler error message, edit the file *getting\_started.s*, which is in the project's directory, and remove the : colon that appears at the end of the *\_start* label, in line 12. Recompile the project to see the error shown in Figure 15. The error message gives the line number in the file (12) where the error was detected. Fix the error, and then compile and load the program again.





### 3.3 Running the Program

As mentioned in the previous section, the Nios II processor is halted at the first instruction after the program has been loaded. To run the program, click the Actions > Continue menu item or click the  $\bigcirc$  icon. The *Getting Started* program performs the following actions on the DE2 board:

- Displays the DE2 board's SW switch settings on the red lights LEDR
- Displays the KEY<sub>1</sub>, KEY<sub>2</sub>, and KEY<sub>3</sub> pushbutton states on the green lights LEDG
- Shows a rotating pattern on the HEX displays. If KEY<sub>1</sub>, KEY<sub>2</sub>, or KEY<sub>3</sub> is pressed, the pattern is changed to correspond to the settings of the SW switches.

The Continue command runs the program indefinitely. To force the program to halt, select the Actions > Stop command, or click the  $\square$  icon. This command causes the processor to halt at the instruction to be executed next, and returns control to the Monitor Program. Another way to stop the execution of this program is to press the pushbutton KEY<sub>0</sub> on the DE2 board; this pushbutton is connected to the reset input of the Nios II processor in the DE2 Basic Computer. Resetting the processor causes program execution to stop and sets the processor to its reset address, which is address 0 in this system.

Figure 16 shows an example of what the display may look like when the program is halted by using the Stop command. The display highlights in yellow the next program instruction, which is at address  $0 \times 00000070$ , to be executed, and highlights in red the register values in the Nios II processor that have changed since the last program stoppage. Other screens in the Monitor Program are also updated, which will be described in later parts of this tutorial.

### 3.4 Using the Disassembly Window

In Figure 16 the Disassembly window shows six machine instructions, at the memory addresses  $0 \times 0000005c$ ,  $0 \times 00000064$ ,  $0 \times 00000068$ ,  $0 \times 0000006c$ , and  $0 \times 00000070$ . The leftmost column in the window gives the memory addresses, the middle column displays the machine code at that address, and the rightmost column shows both the original source code for the instruction, in a brown color, and the disassembled view of the machine code that is stored in memory, in a green color. As shown in the figure, the program may be implemented with different instructions from those given in the source code. For example subi r7, r7, 1 is implemented in this program by using addi r7, r7, -1.

The Disassembly window can be configured to display less information on the screen, such as not showing the source code from the *.s* assembly language file or not showing the machine encoding of the instructions. These settings can be changed by right-clicking on the Disassembly window and selecting the appropriate menu item, as shown in Figure 17. The color scheme used in the Disassembly window is given in Table 1.

Different regions of memory can be disassembled and displayed by scrolling, using either the vertical scrollbar on the right side of the Disassembly window or a mouse scroll wheel. It is also possible to scroll the display to a region of memory by using the Goto instruction panel at the top of the Disassembly window, or using the command Actions > Goto instruction. The instruction address provided for the Goto command must be a multiple of four,

NO_BUTTON:         pc         0x00000000           stwio         r6,0(r20)         /* store to HEX3         pc         0x00000000           a1800035         stwio         r6,0(r20)         /* store to HEX3         r2         0x00000000           a9800035         stwio         r6,0(r21)         /* store to HEX7.         r3         0x0000000           a9800035         stwio         r6,0(r21)         /* rotate the display         r5         0x0000000           300c107a         roli         r6,r6,1         /* rotate the display         r6         0x0000000           300c107a         roli         r7,r2,r00000         /* delay counter *         r9         0x0000000           01c000b4         orhi         r7,r7,r0x1         0x0000000         r11         0x0000000           break         r7,r7, -0x1         break         r14         0x0000000           r15         0x1000000         r15         0x1000000         r14         0x1000000           subi         r7,r7,r0x1         break         r18         0x1000000         r19         0x0000000	Loto instruction         Address (nex) or symbol name:         Loo         PC         0x00000070           0x0000005c         al800035         stwio         r6, 0(r20)         /* store to HEX3         r1         0x00000000           0x0000006c         al800035         stwio         r6, 0(r20)         /* store to HEX3         r1         0x00000000           0x0000006c         al800035         stwio         r6, 0(r21)         /* store to HEX7         r4         0x00000000           0x0000006d         a9800035         stwio         r6, 0(r21)         /* rotate the display         r6         0x000000000           0x0000006d         300c107a         roli         r6, r6, 1         /* rotate the display         r6         0x00000000           0x0000006c         39e1a804         addi         r7, r7, r0x7960         r10         0x00000000           DELAY:         subi         r7, r7, r7, 1         DELAY:         r14         0x10000000           0x00000070         39ffffc4         addi         r7, r7, r0x1         r15         0x10000000           r17         0x00000007         r17         0x00000000         r17         0x10000000           r18         0x10000000         r17         0x10000000         r16         0x10000000
N0_BUTTON:         zero         0x0000000           stwio         r6,0(r20)         /* store to HEX3         r1         0x0000000           a1800035         stwio         r6,0(r20)         /* store to HEX7         r2         0x0000000           a9800035         stwio         r6,0(r21)         /* store to HEX7         r4         0x0000000           a9800035         stwio         r6,0(r21)         /* rotate the display         r5         0x00000000           a9800036         roli         r6, r6, 1         /* rotate the display         r6         0x00000000           300c107a         roli         r6, r6, 0x1         /* delay counter */         r6         0x00000000           movia         r7, 100000         /* delay counter */         r10         0x00000000           01c000b4         orhi         r7, r7, r0         0x00000000         r11         0x00000000           subi         r7, r7, r0         ox00000000         r14         0x00000000           subi         r7, r7, r0         ox10000000         r14         0x00000000           subi         r7, r7, r0         r15         0x10000000         r14         0x00000000           subi         r7, r7, -0x1         r15         0x10000000 </th <th>N0_BUTTON: stwio         r6,0(r20)         /* store to HEX3         zero         0x0000000 r1         0x0000000 cox00000060           a180035         stwio         r6,0(r20)         /* store to HEX3         r1         0x0000000 r2         0x0000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r4         0x00000000 r6         0x00000000 r6         0x00000000 r6         0x00000000 r6         0x00000000 r10         0x00000000 r10         0x00000000 r10         0x00000000 r10         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x000000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x000000000000 r11         0x000000000 r11         0x000000000000 r11         0x00000000 r11         0x00000000000000000000000000000000000</th>	N0_BUTTON: stwio         r6,0(r20)         /* store to HEX3         zero         0x0000000 r1         0x0000000 cox00000060           a180035         stwio         r6,0(r20)         /* store to HEX3         r1         0x0000000 r2         0x0000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r3         0x00000000 r4         0x00000000 r6         0x00000000 r6         0x00000000 r6         0x00000000 r6         0x00000000 r10         0x00000000 r10         0x00000000 r10         0x00000000 r10         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x000000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x00000000 r11         0x000000000000 r11         0x000000000 r11         0x000000000000 r11         0x00000000 r11         0x00000000000000000000000000000000000
stwio       r6,0(r20)       /* store to HEX3       r1       0x00000000         N0_BUTTON:       stwio       r6,0(r20)       r2       0x00000000         a800035       stwio       r6,0(r21)       /* store to HEX7.       a0x00000000         a9800035       stwio       r6,0(r21)       /* store to HEX7.       a0x00000000         a9800035       stwio       r6,0(r21)       /* rotate the display       r6       0x00000000         300c107a       r01i       r6,r6,0x1       /* delay counter */       0x00000000       r7       0x00000000         movia       r7,100000       /* delay counter */       r0       0x00000000       r11       0x00000000         gela804       addi       r7,r7,r0x1       r1       0x00000000       r12       0x00000000         bELAY:       subi       r7,r7,-0x1       r13       0x10000000       r14       0x10000000         12       0x00000000       r14       0x10000000       r14       0x10000000       r14       0x00000000         13       0x00000000       r13       0x00000000       r14       0x00000000       r17       0x10000000       r17       0x10000000       r14       0x010000000       r17       0x100000000       r18	0x0000005c         al800035         stwio         r6, 0(r20)         /* store to HEX3         r1         0x0000000 r2         0x00000000 r2         0x00000000 r3         0x000000000 r4         0x00000000000000000000000000000000000
N0_BUTTON:       r2       0x0000000         a1800035       stwio       r6,0(r20)       /* store to HEX7       r3       0x0000000         a9800035       stwio       r6,0(r21)       /* store to HEX7       r4       0x0000000         a9800035       stwio       r6,0(r21)       /* rotate the display       r4       0x0000000         300c107a       roli       r6, r6, 1       /* rotate the display       r6       0x0000000         movia       r7, 100000       /* delay counter */       9       0x00000000         01c000b4       orhi       r7, zero, 0x2       9       0x00000000         39e1a804       addi       r7, r7, -0x7960       r12       0x0000000         DELAY:       subi       r7, r7, -0x1       r16       0x1000000         39ffffc4       addi       r7, r7, -0x1       r16       0x1000000         r18       0x10000005       r18       0x1000005       r18       0x1000005	N0_BUTTON:         r2         0x00000000           0x0000005c         al800035         stwio         r6, 0(r20)         r3         0x00000000           stwio         r6, 0(r21)         /* store to HEX7         r3         0x00000000           0x00000060         a9800035         stwio         r6, r6, 1         /* rotate the display         r5         0x00000000           0x00000064         300c107a         roli         r6, r6, 1         /* rotate the display         r5         0x00000000           0x00000064         300c107a         roli         r7, r200000         /* delay counter */         r8         0x0000000           0x00000066         01c000b4         orhi         r7, r2 ero, 0x2         r10         0x00000000           0x00000066         39e1a804         addi         r7, r7, r0x7960         r13         0x00000000           DELAY:         subi         r7, r7, r0x1         r14         0x00000000         r15         0x10000000           10         0x00000000         subi         r7, r7, r0x1         r16         0x10000000         r17         0x10000000           11         0x00000000         r17         r18         0x10000000         r17         r18         0x10000000
a1800035       stwio       r6, 0(r20)         stwio       r6, 0(r21)       /* store to HEX7         a9800035       stwio       r6, 0(r21)         roli       r6, r6, 1       /* rotate the display         300c107a       roli       r6, r6, 1         movia       r7, 100000       /* delay counter */         01c000b4       orhi       r7, zero, 0x2         39e1a804       addi       r7, r7, -0x7960         DELAY:       subi       r7, r7, -0x1         39ffffc4       addi       r7, r7, -0x1         widi       r7, r7, -0x1       r16         01/2 000000       widi       r7, r7, 0x1	0x0000005c         a1800035         stwio         r6, 0(r20)         /* store to HEX7.           0x00000060         a9800035         stwio         r6, 0(r21)         /* store to HEX7.           0x00000060         a9800035         stwio         r6, 0(r21)         /* rotate the display           0x00000064         300c107a         roli         r6, r6, 1         /* rotate the display           0x00000068         01c000b4         orhi         r7, zero, 0x2         r8         0x0000000           0x0000006c         39e1a804         addi         r7, r7, -0x7960         r11         0x0000000           DELAY:         subi         r7, r7, r0x1         r14         0x0000000           0x00000070         39ffffc4         addi         r7, r7, -0x1         r17         0x1000000           r14         0x0000000         r15         0x1000000         r17         0x1000000
stwio       r6,0(r21)       /* store to HEX7       r4       0x00000180         a9800035       stwio       r6,0(r21)       /* rotate the display       r5       0x0000000         300c107a       roli       r6,r6,1       /* rotate the display       r6       0x00f0000         300c107a       roli       r6,r6,01       /* rotate the display       r6       0x00f0000         movia       r7,100000       /* delay counter */       r8       0x00000000         01c000b4       orhi       r7,r2,r0,0x2       r10       0x00000000         39e1a804       addi       r7,r7,r0x1       r12       0x0000000         DELAY:       subi       r7,r7,-0x1       r14       0x0000000         r17       0x10000000       r15       0x1000000         r16       0x10000000       r14       0x0000000         r17       0x10000000       r15       0x10000000         r17       0x10000050       r18       0x10000010         r19       0x00000007c       r19       0x00000007c	0x00000060       a9800035       stwio       r6, 0(r21)       /* store to HEX7       r4       0x00000060         0x00000064       300c107a       r01i       r6, r6, 1       /* rotate the display       r5       0x00000062         0x00000064       300c107a       r01i       r6, r6, 1       /* rotate the display       r6       0x00000062         0x00000068       01c000b4       orhi       r7, zero, 0x2       0x00000000       r10       0x00000000         0x00000060       39e1a804       addi       r7, r7, r0x7960       r12       0x00000000         DELAY:       subi       r7, r7, r0x1       r14       0x00000000         r14       0x0000000       r14       0x0000000       r12       0x0000000         r14       0x0000000       r14       0x1000000       r15       0x1000000         r15       0x1000000       r14       0x0000000       r14       0x1000000         r15       0x1000000       r17       0x1000000       r17       0x1000000         r14       0x0000000       r17       0x1000000       r17       0x1000000         r15       0x1000000       r17       0x1000000       r17       0x1000000         r16       0x0000000
a9800035       stvio       r6, 0(r21)         roli       r6, r6, 1       /* rotate the display         300c107a       roli       r6, r6, 1         movia       r7, 100000       /* rotate the display         01c000b4       orhi       r7, zero, 0x2         39e1a804       addi       r7, r7, r0x7960         DELAY:       subi       r7, r7, r7, 1         39ffffc4       addi       r7, r7, -0x1         movia       r7, r7, r0, DELAY:       r7, r7, r0, r1         39ffffc4       addi       r7, r7, r0, DELAY:         r17       0x1000000         r18       0x1000000         r14       0x0000000         r15       0x1000000         r16       0x1000000         r17       0x1000000         r18       0x1000000         r19       0x0000000         r18       0x1000000         r19       0x0000000	0x00000060         a9800035         stwio         r6, 0(r21)         roli         r6, r6, 1         /* rotate the display         r5         0x0000006/r6           0x00000064         300c107a         roli         r6, r6, 1         /* rotate the display         r6         0x0000006/r2         0x0000006/r2         0x0000000/r2         0x000000/
300c107a       roli       r6, r6, 1       /* rotate the display       r6       0x0000000         300c107a       roli       r6, r6, 0x1       /* rotate the display       r7       0x000006b2e         movia       r7, 100000       /* delay counter *       r0       0x00000000       r9       0x00000000         01c000b4       orhi       r7, zero, 0x2       /* delay counter *       r10       0x00000000         39e1a804       addi       r7, r7, -0x7960       r12       0x00000000         DELAY:       subi       r7, r7, r7, 1       0x10000000       r14       0x00000000         39fffro4       addi       r7, r7, -0x1       r16       0x10000000         r18       0x10000000       r17       0x10000000       r17       0x10000000	volume       roli       r6, r6, l       /* rotate the display       r6       0x0000060         0x00000064       300cl07a       roli       r6, r6, 0x1       r7       0x0000062         movia       r7, 100000       /* delay counter       r9       0x0000000         0x00000068       01c000b4       orhi       r7, r2ero, 0x2       r9       0x0000000         0x00000062       39ela804       addi       r7, r7, -0x7960       r10       0x0000000         DELAY:       subi       r7, r7, r7, 1       r14       0x0000000         0x00000070       39ffffc4       addi       r7, r7, -0x1       r15       0x1000000         r17       0x10000000       r17       0x10000000       r17       0x10000000         r18       0x10000000       r19       0x00000007       r19       0x00000000
300c107a       roli       r6, r6, 0x1       r7       0x00000b2e         movia       r7, 100000       /* delay counter */       r8       0x00000000         01c000b4       orhi       r7, zero, 0x2       r9       0x00000000         39e1a804       addi       r7, r7, -0x7960       r10       0x00000000         DELAY:       subi       r7, r7, r7, 1       0x00000000       r14       0x00000000         39ffffc4       addi       r7, r7, -0x1       r16       0x10000000         16       0x10000000       r18       0x10000000         17       0x10000000       r16       0x10000000         18       0x10000000       r18       0x10000000         19       0x00000007c       r19       0x00000000	0x00000064       300c107a       roli       r6, r6, 0x1       r7       0x00000000         movia       r7, 100000       /* delay counter       r8       0x00000000         0x00000068       01c000b4       orhi       r7, r2ero, 0x2       r8       0x00000000         0x00000066       39e1a804       addi       r7, r7, -0x7960       r11       0x00000000         DELAY:       subi       r7, r7, r7, 1       r14       0x0000000         0x00000070       39ffffc4       addi       r7, r7, -0x1       r15       0x1000000         18       0x1000001       r19       0x0000007       r19       0x0000000
movia       r7, 100000       /* delay counter *         01c000b4       orhi       r7, zero, 0x2         39e1a804       addi       r7, r7, -0x7960         DELAY:       subi       r7,	0x00000068       01c000b4       orhi       r7, 100000       /* delay counter       r8       0x00000000         0x00000062       39e1a804       orhi       r7, r7, -0x7960       r10       0x00000000         DELAY:       subi       r7, r7, 1       0x00000000       r14       0x00000000         0x00000070       39fffc4       addi       r7, r7, -0x1       r15       0x10000000         1       0x00000000       r14       0x00000000       r15       0x10000000         r14       0x00000000       r16       0x10000000       r17       0x10000000         r18       0x10000000       r19       0x00000000       r19       0x0000000
movia         r7, 100000         /* delay counter *         r10         0x00000000           01c000b4         orhi         r7, zero, 0x2         r10         0x00000000           39e1a804         addi         r7, r7, -0x7960         r11         0x00000000           DELAY:         subi         r7, r7, 1         r13         0x00000000           39ffffc4         addi         r7, r7, -0x1         r16         0x1000000           16         0x10000000         r18         0x1000000         r19         0x0000000	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
01c000b4         orhi         r7, zero, 0x2           39e1a804         addi         r7, r7, -0x7960           DELAY:         subi         r7, r7, r7, 1           DELAY:         0x00000000           39fffrc4         addi         r7, r7, -0x1	0x00000068         01c000b4         orhi         r7, zero, 0x2         r11         0x00000000           0x0000006c         39e1a804         addi         r7, r7, -0x7960         r12         0x00000000           DELAY:         subi         r7, r7, 1         r13         0x00000000           0x00000070         39ffffc4         addi         r7, r7, -0x1         r15         0x10000000           1         0x00000000000000000000000000000000000
39ela804       addi r7, r7, -0x7960       r11       0x00000000         DELAY:       subi r7, r7, 1       0x00000000       r12       0x00000000         DELAY:       subi r7, r7, 1       0x00000000       r15       0x10000000         39ffffc4       addi r7, r7, -0x1       r16       0x10000000         17       0x10000000       r17       0x10000000         18       0x10000000       r19       0x00000000	0x0000006c         39ela804         addi r7, r7, -0x7960         r12         0x0000000 r12         0x0000000 r13         0x00000000 r14         0x00000000 r13         0x00000000 r14         0x00000000 r14         0x00000000 r14         0x000000000 r15         0x000000000 r14         0x000000000 r14         0x000000000000 r15         0x100000000000000000000000000000000000
DELAY: subi r7, r7, 1 DELAY: 39fffc4 addi r7, r7, -0x1 113 0x0000000 r14 0x0000000 r15 0x1000000 r16 0x1000000 r17 0x1000050 r18 0x10000050 r18 0x10000050 r19 0x0000007c	DELAY: DELAY: subi r7, r7, 1 DELAY: 0x00000070 39ffffc4 addi r7, r7, -0x1 Comparing the subi r7,
DELAY:       r14       0x0000000         subi       r7, r7, 1       r15       0x10000040         DELAY:       r16       0x10000000       r17       0x10000000         39fffcd       addi       r7, r7, -0x1       r18       0x10000000         118       0x10000000       r17       0x10000000       r18       0x10000000         118       0x10000000       r19       0x0000007c	DELAY: subi r7, r7, 1 DELAY: subi r7, r7, 1 DELAY: 0x00000070 39fffrd4 addi r7, r7, -0x1 Control of the subi r7, r7, -0x1 Control of th
subi     r7, r7, 1       DELAY:       39ffffc4       addi     r7, r7, -0x1       r16     0x10000000       r17     0x10000000       r18     0x10000000       r19     0x0000007c	subi     r7, r7, 1       DELAY:       0x00000070       39ffffc4       addi       r7, r7, -0x1
DELAY: 39ffffc4 addi r7, r7, -0x1 x0 DELAY r16 0x1000000 r17 0x1000050 r18 0x10000010 r19 0x000007c	DELAY: 0x00000070 39fffc4 addi r7, r7, -0x1 16 0x1000000 r17 0x1000000 r18 0x1000000 r19 0x0000070
39ffffc4         addi         r7, r7, -0x1         r17         0x10000050           x0         DELAY         x1         x1         0x10000010           x19         0x0000007c         x19         0x0000007c	0x00000070 39ffffc4 addi r7, r7, -0x1
r18 0x1000010 r19 0x000007c	
r19 0x000007c	
_ × Info & Errors _	rminal _ × Info & Errors _
	FAG UART link established using cable "USB-Blaster Symbols loaded.
nk established using cable "USB-Blaster Symbols loaded.	JSB-0]", device 1, instance 0x00 Source code loaded.
_ × Info & Errors	Terminal _ × Info & Errors

Figure 16. The Monitor Program display after the program has been stopped.

0x00000070	39ffffc4	addi x7 x7 Out
		✓ Show instruction words
0x00000074	383ffele	✓ Show source code
		Goto instruction

Figure 17. Pop-up menu to configure the display of the Disassembly window.

because Nios II instructions are word-aligned. As an example, enter the label DELAY or the address 70, and press Go. The Disassembly window scrolls to the address  $0 \times 00000070$ , as depicted in Figure 18, and highlights the instruction using a pink color.

Register and memory values can be examined in the Disassembly window while the Nios II processor is *halted*. This is done by hovering the mouse over a *register* or *register* + *offset* name for an instruction in the window, as illustrated in Figure 19. If the instruction loads or stores a value from/to memory, then the Monitor Program displays the current value of the memory location in the pop-up.

The Disassembly window also produces clickable links in its display of branch and call instructions. Clicking on one of these links scrolls the display to show the target instruction of the branch or call. Figure 20 shows an

Color	Description
Brown	Source code
Green	Disassembled instruction name
Blue	Registers
Orange	Immediate & offset values
Dark blue	Address values & labels
Purple	Clickable link
Gray	Machine encoding of the instruction

Table 1. Disassembly window color scheme.

sassembly					<b>-</b> ×	Register	5 _	
-	1					Rea		Г
oto instruction	Address (hex)	or symbol name:	DELAY	<u>G</u> o	Hide	pc	0x00000000	1
						zero	0x00000000	
		DELAY:				rl	0x00000000	11
		subi	r7, r7, 1			r2	0x00000000	1
		DELAY:				r3	0x00000000	
0x00000070	39ffffc4	addi	r7, r7, -0x1			r4	0x00000000	
		bne	r7, r0, DELA			r5	0x0000000	
0x00000074	383ffele	bne		0x00000070: DELAY)		r6	0x0000000	
			- , ,	,		r7	0x0000000	
		br	DO DISPLAY		1000	r8	0x0000000	11
0x00000078	003ff006	br	-0x40 (0x000000	C. DO DISPLAY)	222	r9	0x00000000	11
0.00000070	00011000		0440 (040000000			r10	0x0000000	11
		HEX bits:				r11	0x0000000	11
0x0000007c	0000000£	ldh	zero, O(zero)			r12	0x00000000	11
0x00000080	20000226	beq	r4, zero, 0x8 ()	x000008c)		r13 r14	0x00000000	11
0x00000084	00000ec0	call	0x0000003b (0x0)			r14 r15	0x00000000 0x00000000	11
0x00000088	00000306	br	0xc (0x00000098			r16	0x00000000	
0x0000008c	df401215	stw	ea, 72(sp)			r17	0x00000000	
0x000000000	01401213	Labor	ea, 72(sp)		-	r18	0x00000000	
•					Þ	r19	0x00000000	11
Disassembly $\int B$	reakpoints / Me	mory ( Watches	/ Trace /			r20	0x00000000	- 1
erminal			_ ×	Info & Errors			-	
FAG UART lin	k establish	ed using cab	le "USB-Blaster	Connection established t	to GDB server	at loca	lhost:2399	٦
JSB-0]", dev	ice l, inst	ance OxOO		Symbols loaded.				
				Source code loaded.				
				Source code rodded.				

Figure 18. Goto instruction panel in the Disassembly window.

example of a clickable link for a call instruction.

The Disassembly window attempts to show disassembled code for all words in memory, even though some memory words may not corresponds to Nios II executable code. For example, in Figure 18 the memory word at address  $0 \times 0000007$ C has the value  $0 \times 0000000$ f and represents data that is used by the program. Even though the Disassembly window attempts to show a corresponding Nios II assembly language instruction for this memory word, the disassembled machine code is not meaningful because this data does not represent executable code.



Figure 19. Examining a register value in the Disassembly window.

		display: call UPDATE_HEX_DISPLAY display:
0x00001028	00011b00	call 0x0000046c (0x000011b0: UPDATE_HEX_DISPLAY)
		<pre># delay loop of app; Goto instruction label 'UPDATE_HEX_DISPLAY' (0x000011b0</pre>

Figure 20. A clickable link in the Disassembly window.

#### 3.5 Single Stepping Program Instructions

Before discussing the single step operation, it is convenient to restart execution of the *Getting Started* program from the beginning. Click the Actions > Restart menu item or click the <sup>1</sup>/<sub>2</sub> icon to restart the program. Note that if the program is running, it must first be halted before the restart command can be performed.

The Monitor Program has the ability to perform single-step operations. Each single step consists of executing a single Nios II machine instruction and then returning control to the Monitor Program. If the source code of the program being debugged is written in C, each individual single-step will still correspond to one assembly language (machine) instruction generated from the C code.

The single-step operation is invoked by selecting the Actions > Single step menu item or by clicking on the  $\bigcirc$  icon. The instruction that is executed by the processor is the one highlighted in the Disassembly window before the single step.

Since the first step in this section was to restart the program, the first single step will execute the instruction at address 0, which will set the upper bits of the Nios II register r15 to the value  $0 \times 1000$ . Subsequent single steps will continue to execute one instruction at a time, in sequential order. Single stepping at a branch instruction may jump to a non-sequential instruction address if the branch is taken. This behavior can be observed by single stepping to the address  $0 \times 0000004c$ , which is a beq instruction. Single stepping at this instruction will set the *pc* value to  $0 \times 0000005c$ , which is the location of the instruction executed at this point in the *Getting Started* program when no pushbutton KEY is being pressed on the DE2 board.

Another way to perform the single-step operation is to use the Step Over Subroutine command in the Actions menu. This command performs a normal single step, unless the current instruction is a call instruction. In this case the program will run until the called subroutine is completed.

### 3.6 Using Breakpoints

An *instruction breakpoint* provides a means of stopping a Nios II program when it reaches a specific address. A simple procedure for setting an instruction breakpoint is:

- 1. In the Disassembly window, scroll to display the instruction address that will have the breakpoint. As an example, scroll to the instruction at the label *NO\_BUTTON*, which is address 0x0000005c.
- 2. Click on the gray bar to the left of the address 0000005c in the Disassembly window. As illustrated in Figure 21 the Monitor Program displays a red dot next to the address to show that an instruction breakpoint has been set. Clicking the same location again removes the breakpoint.

🥔 Altera Monito	or Program [	Nios II] - Moni	itor_Tutorial.ncf : g	etting_started.srec [Paused]			×
Eile <u>S</u> ettings <u>A</u>	<u>A</u> ctions <u>W</u> indo	ows <u>H</u> elp					
👱 🗈  🔒 🕹	🔒 🔿 E	⊫ III 0 <sub>6</sub> ↓	4 🖑				
Disassembly				_ >	Register	5 _	×
Goto instruction	Address (hex)	or symbol name:	5c	Go Hide	Reg	Value	
					pc	0x0000000	
					zero	0x00000000	
					rl	0x00000000	
		NO_BUTTON:			r2	0x00000000	
		stwio	r6, 0(r20)	/* store to HEX3	r3	0x00000000	
		NO_BUTTON:			r4	0x00000180	1295
0x0000005c	a1800035	stwio	r6, 0(r20)		r5	0x00000000	1895
		stwio	r6, 0(r21)	/* store to HEX7 .	r6	0x000000f	
0x00000060	a9800035	stwio	r6, 0(r21)		r7	0x000170ed	
		roli	r6, r6, l	/* rotate the display	r8	0x00000000	
0x00000064	300c107a	roli	r6, r6, 0x1		2 r9	0x00000000	
			,,		r10	0x00000000	
		movia	r7, 100000	/* delay counter */	r11	0x00000000	
0x0000068	01c000b4	orhi	r7, zero, 0x2	, acity counter ,	r12	0x0000000	
0x0000006c	39e1a804	addi	r7, r7, -0x7960		r13	0x0000000	
0x00000000	39614004	auur	L7, L7, -0x7500		r14	0x0000000	
		DELAY:			r15	0x10000040	
					r16	0x10000000	
		subi	r7, r7, 1		r17	0x10000050	
4				•	r18 r19	0x10000010	
Disassembly / Bre	eakpoints / Me	mory / Watches	/ Trace /		r20	0x0000007c 0x10000020	
			<u>,</u> ,		LZU	0810000020	
[erminal			_ ×	Info & Errors		-	×
JTAG UART link [USB-0]", devi		-	le "USB-Blaster	Source code loaded. INFO: Program Trace not enabled, b Program stopped @ 0x00000070 BREAK: Program break @ 0x0000005c	ecause tra	ce requires	
				Info & Errors / GDB Server /			

Figure 21. Setting an instruction breakpoint.

Once the instruction breakpoint has been set, run the program. The breakpoint will trigger when the pc register value equals  $0 \times 0000005c$ . Control then returns to the Monitor Program, and the Disassembly window highlights in a yellow color the instruction at the breakpoint. A corresponding message is shown in the lnfo & Errors pane.

Some versions of the Nios II processor support other types of breakpoints in addition to instruction breakpoints. Other types of breakpoints are described Appendix A of this document.

### 3.7 Examining and Changing Register Values

The **Registers** window on the right-hand side of the Monitor Program display shows the value of each register in the Nios II processor and allows the user to edit most of the register values. The number format of the register values can be changed by right-clicking in the **Registers** window, as illustrated in Figure 22.

Registe	rs _	×
Reg	Value	
pc	0x00000000	-
zero	0x00000000	
rl	0x00000000	
r2	0x00000000	
r3	0x00000000	
r4	0x00000180	395
r5	0x00000000	
Bina	ry	
<u>O</u> cta	al	
<u>D</u> eci	mal	
<ul> <li><u>Неха</u></li> </ul>	adecimal	
ーーパ Sign	ed representation	
r13	0x00000000	
r13 r14		
	0x00000000	
r14	0x00000000 0x00000000	
r14 r15	0x00000000 0x00000000 0x10000040	
r14 r15 r16	0x00000000 0x00000000 0x10000040 0x10000000	
r14 r15 r16 r17	0x00000000 0x00000000 0x10000040 0x10000000 0x10000050	

Figure 22. Setting the number format for displaying register values.

Each time program execution is halted, the Monitor Program updates the register values and highlights any changes in red. The user can also edit the register values while the program is halted. Any edits made are visible to the Nios II processor when the program's execution is resumed.

As an example of editing a register value, first scroll the Disassembly window to the label DELAY, which is at address  $0 \times 00000074$ . Set a breakpoint at this address and then run the program. After the breakpoint triggers and control returns to the Monitor Program, notice that there is a large value in register r7. This value is used as a counter in the delay loop. As indicated in Figure 23, double-click on the contents of register r7 and edit it to the value 1. Press Enter on the computer keyboard, or click away from the register value to apply the edit. Now, single-step the program to see that it exits from the delay loop after one more iteration, when r7 becomes 0.

		DELAY: subi DELAY:	r7, r7, 1			r1 r2 r3	0x00000000 0x00000000 0x00000000
0x00000070	39ffffc4	addi	r7, r7, -0x1		- H F	r4	0x00000180
		bne	r7, r0, DELAY		1	r5	0x00000000
0x00000074	383ffele	bne	r7, zero, -0x8 (0x00000070: DELAY)			r6	0x0000001e
						r7	0x00000001
		br	DO DICRIAV		_ 1	r8	0x00000000
			DO_DISPLAY	8	8	r9	0x00000000
0x00000078	003ff006	br	-0x40 (0x0000003c: D0_DISPLAY)		1	r10	0x00000000

Figure 23. Editing a register value.

### 3.8 Examining and Changing Memory Contents

The Memory window, depicted in Figure 24, displays the contents of the system's memory space and allows the user to edit memory values. The leftmost column in the window gives a memory address, and the numbers at the top of the window represent hexadecimal address offsets from that corresponding address. For example, referring to Figure 24, the address of the last word in the second row is  $0 \times 00000010 + 0 \times c = 0 \times 0000001c$ .

Altera Mo	nitor Progra	am [Nios II] - M	onitor Tuta	rial.ncf : g	etting_started.srec [Paused]			X
<u>File S</u> ettings		<u>W</u> indows <u>H</u> elp						
👱 🗈  🗟	<b>₽</b> 🚴 🤇	ə 🗈 🗉 🗞	M 🖑					
Memory					_ ×	Registers	; _	×
Goto memory	address Ad	ldress (hex):			o Query All Devices Refresh Memory Hide	Reg	Value	
•		· · ·				pc	0x000003c	
000000000	+0x0	+0x4 7bc01004	+0x8	+0xc		zero rl	0x00000000 0x00000000	
0x00000000	03c40034		04040034	84000004		r2	0x00000000	
0x00000010	04440034	8c401404	04840034	94800404		r3	0x00000000	
0x00000020	05040034	a5000804	05440034	ad400c04		r4	0x00000180	
0x00000030 0x00000040	04c00034 81000035	9cc01f04 89400037	99800017 91400035	79000037 28000326		r5	0x00000000	1000
0x00000040 0x00000050	200d883a	89400037 89400037	91400035 283ffele	a18000326		r6	0x0000001e	1000
0x000000050	a9800035	300c107a	20311010 01c000b4	39ela804		r7	0x0000000	
0x000000000 0x000000070	39ffffc4	383ffele	01000004 003ff006	00000000f		r8	0x00000000	
0x000000070 0x00000080					1999	r9	0x0000000	
0x00000080 0x00000090	20000226	00000ec0	00000306	df401215 df401217		r10	0x0000000	
0x00000090 0x000000a0	e8bfff17	003da03a	d9401117			r11	0x0000000	
0x0000000a0 0x0000000b0	dfc00017	2801707a	d8400217	d8800317		r12	0x00000000	
0x0000000000000000	d8c00417	d9000517	d9400617	d9800717		r13	0x00000000	
0x000000000 0x0000000d0	d9c00817 dac00c17	da000917	da400a17	da800b17		r14	0x00000000	
		db000d17	db400e17	db800f17		r15	0x10000040	
0x000000e0	dbc01017	dec01304	ef80083a	defffd04		r16	0x1000000	
0x000000f0	dfc00215	dc000115	dc400015	0009313a	-	r17	0x10000050	
•		1133111111123	OAL JAMIA	1111160073		r18 r19	0x10000010 0x0000007c	
Disassembly /	Breakpoints	Memory / Watc	hes / Trace /			r20	0x10000020	
Terminal				_ ×	Info & Errors		-	×
JTAG HAPT 1	ink establ	ished using (	-able "USB-	Blagter	Program stopped @ 0x00000070			
		nstance 0x00	- and the owner	2140001	BREAK: Program break @ 0x0000005c			
[ 00D 0] 7 d					BREAK: Program break @ 0x0000005c			
					BREAK: Program break @ 0x00000074			335
							•	-
					Info & Errors / GDB Server /			_

Figure 24. The Memory window.

If a Nios II program is running, the data values displayed in the Memory window are not updated. When the program is stopped the data can be updated by pressing the Refresh Memory button. By default, the Memory window shows only the contents of memory devices, and does not display any values from memory-mapped I/O devices. To configure the window to display memory-mapped I/O, click on the check mark beside Query All Devices, and then click Refresh Memory.

The color of a memory word displayed depends on whether that memory location corresponds to an actual memory device, a memory-mapped I/O device, or is not mapped at all in the system. A memory location that corresponds to a memory device will be colored black, memory-mapped I/O is shown in a blue color, and a non-mapped address is shown in grey. If a memory location changed value since it was previously displayed, then that memory location is shown in a red color.

Similar to the Disassembly window, it is possible to view different memory regions by scrolling using the vertical scroll bar on the right, or by using a mouse scroll wheel. There is also a Goto memory address panel, which is analogous to the Goto instruction window discussed in section 3.4. Click to turn on the check mark beside Query All Devices in the memory window. In the Goto memory address panel type the address 0x10000000, and then press Go. The display scrolls to the requested address, which corresponds to memory-mapped I/O devices in the DE2 Basic Computer. Click the Refresh Memory button. The data displayed in blue at address 0x10000040 corresponds to the settings of the 18 SW switches on the DE2 board. Experiment with different SW switch settings and press Refresh Memory to see that the switch values are properly displayed.

As an example of editing a memory value, double-click on the memory word at address 0x10000000 and type the hexadecimal data value 15555. Press Enter on the computer keyboard, or click away from the memory word to apply the edit. This memory-mapped address in the DE2 Basic Computer corresponds to the red lights LEDR on the DE2 board. Experiment by editing this memory location to different values and observe the LEDs.

It is possible to change the appearance of the Memory window in a number of ways, such as displaying data as bytes, half-words, or words, and so on. The Memory window provides additional features that are described in more detail in the Appendix A of this document.

# 4 Working with Project Files

Project files store the settings for a particular project, such as the specification of a hardware system and program source files. A project file, which has the filename extension *ncf*, is stored into a project's directory when the project is created.

The Monitor Program provides the following commands, under the File menu, for working with project files:

- 1. New Project: Presents a series of screens that are used to create a new project.
- 2. Open Project: Displays a dialog to select an existing project file and loads the project.
- 3. Open Recent Project: This command displays the five most recently-used project files, and allows these projects to be reopened.
- 4. Save Project: Saves the current project's settings. This command can be used to save a project's settings after they have been modified by using the Settings command, which is described below.

### 4.1 Modifying the Settings of an Existing Project

After a project has been created, it is possible to modify many of its settings, if needed. This can be done by clicking on the menu item Settings > System Settings in the Monitor Program, or the icon. This action will display the existing System Settings for the project, and allow them to be changed. Similarly, the program settings for the

project can be displayed or modified by using the command Settings > Program Settings, or the  $\square$  icon. The change these settings, the Monitor Program has to first be disconnected from the system being debugged. This can be done by using the command Actions > Disconnect.

## 5 Using the Monitor Program with a Nios II Evaluation License

In our discussion of Figure 12, in section 3.1, we showed how the Monitor Program can be used to download a prebuilt Nios II hardware system onto an FPGA board, when the Nios II processor has a license. It is also possible to use the Monitor Program to debug hardware systems in which the Nios II processor includes only an evaluation license. In this case it is necessary to download the hardware system onto the FPGA board by using the *Programmer* tool provided in the Quartus II software, rather than using the Monitor Program for this purpose. The Quartus II Programmer tool provides a pop-up window, shown in Figure 25, that indicates activation of the evaluation license for the Nios II processor. This pop-up window has to remain open in order to maintain the evaluation license for Nios II. As long as the pop-up window remains open, the Monitor Program can be used to compile and download software programs into the hardware system.

OpenCore Plus Status	
Click Cancel to stop using OpenCore Plus IP.	
Time remaining: unlimited	

Figure 25. The Quartus II Programmer pop-up window.

# 6 Using the Terminal Window

This section of the tutorial demonstrates the functionality of the Monitor Program's *Terminal* window, which supports text-based input and output. For this example, create a new Monitor Program project for the DE2 board, called *Monitor\_Terminal*. Store the project in a directory of your choice.

When creating the project, follow the same steps shown for the *Monitor\_Tutorial* project, which were illustrated in Figures 6 to 11. For the screen shown in Figure 8 set the program type to Assembly Program, and select the sample program named *JTAG UART*. The source code file that will be displayed in the screen of Figure 9 is called *JTAG\_UART*.s. It communicates using memory-mapped I/O with the *JTAG\_UART* in the DE2 Basic Computer that is selected as the Terminal device in the screen of Figure 10.

Compile and load the program by following the procedure in section 3.2. Then, run the program using the steps in section 3.3. The Monitor Program window should appear as shown in Figure 26. Notice that the Terminal window displays a text prompt which is sent by the *JTAG\_UART.s* program. Click the mouse inside the Terminal window. Now, any characters typed on the computer keyboard are sent by the *Monitor* Program to the JTAG UART. These characters are shown in the Terminal window as they are typed, because the *JTAG\_UART.s* program echos the characters back to the Terminal window.

The Terminal window supports a subset of the control character commands used for a de facto standard terminal, called the *VT100*. The supported commands are listed in Table 2. In this table <ESC> represents the ASCII character with the code 0x1B.

		ows <u>H</u> elp	tor_Tutorial.ncf : JT	no_oantriaree [na	5			
4	🔉 🔊	▶ 00 <b>0</b> 6 <b>8</b>	î 🖑					
isassembly					-	× Register:	s _	. :
oto instruction	Address (bey)	or symbol name:	DFLAV	Go	Hid	Reg	Value	Γ
ioto instruction	[Hadress (Hoxy	or symbol namer [				pc	0x00000000	_
		.text			/* executable cod	zero	0x00000000	ł.
		.global	_start			rl	0x00000000	-
		_start:				r2	0x00000000	
		/* set	up stack pointer *	/		r3	0x00000000	
		movia	sp, 0x007FFFFC	/* st	arts from largest		0x00000000	
		_start:				r5	0x00000000	
0x00000000	06c02034	orhi	sp, zero, 0x80			r6	0x00000000	
0x00000004	deffff04	addi	sp, sp, -0x4			r7	0x00000000	
						r8 19	0x00000000	
		movia	r6, 0x100010	00 /*	JTAG UART base a	r10	0x00000000 0x00000000	
0x00000008	01840034	orhi	r6, zero, 0x1000			r11	0x00000000	
0x0000000c	31840004	addi	r6, r6, 0x1000			r12	0x00000000	
						r13	0x00000000	
		/* prin	t a text string */			r14	0x00000000	
		movia	r8, TEXT STR	ING		r15	0x00000000	
0x00000010	02000034	orhi	r8, zero, 0x0			r16	0x00000000	
0x00000014	42001a04	addi	r8, r8, 0x68			r17	0x00000000	j
						▼ r18	0x00000000	j
•	,				)	r19	0x00000000	J
Disassembly / Br	eakpoints / Me	mory (Watches	( <u>Trace</u> )			r20	0x00000000	1
erminal			_ × _	info & Errors			-	ļ
TAG UART lin	k establish	ed using cabl	e "USB-Blaster	Verified OK				Ĩ
USB-01", dev:		-		Connection estab	lished to GDB ser	ver at loca	alhost:2399	
				Symbols loaded.				
TAG UART exa	mple code			- Source code load	led.			
	apac oodc			INFO: Program Tr	ace not enabled,	because tra	ce requires	
								>
								2

Figure 26. Using the Terminal window.

# 7 Using C Programs

C programs are used with the Monitor Program in a similar way as assembly language programs. To see an example of a C program, create a new Monitor Program project for the DE2 board, called *Monitor\_Terminal\_C*. Store the project in a directory of your choice. Use the same settings as for the *Monitor\_Terminal* example, but set the program type for this project to C Program. Select the C sample program called *JTAG UART*. As illustrated in 27 this sample program includes a C source file named *JTAG\_UART.c*; it has identical functionality to the assembly language code used in the previous example. Compile and run the program to observe its behavior.

The C code in *JTAG\_UART.c* uses memory-mapped I/O to communicate with the JTAG UART. Alternatively, it is possible to use functions from the standard C library *stdio.h*, such as *putchar*, *printf*, *getchar*, and *scanf* for this purpose. Using these library functions impacts the size of the Nios II executable code that is produced when the C program is compiled, by about 30 to 64 KBytes, depending on which functions are needed. It is possible to minimize the size of the code generated for this library by checking the box labeled Use small C library in Figure 27. When this option is used the library has reduced functionality. Some limitations of the small C library include: no floating-point support in the output routines, such as *printf*, and no support for input routines, such as *scanf* and *getchar*.

In Figure 27 the option Emulate unimplemented instructions is checked. This option causes the C compiler to

Character Sequence	Description
<esc>[2J</esc>	Erases everything in the Terminal window
<esc>[7h</esc>	Enable line wrap mode
<esc>[71</esc>	Disable line wrap mode
<esc>[#A</esc>	Move cursor up by # rows or by one row if # is not specified
<esc>[#B</esc>	Move cursor down by # rows or by one row if # is not specified
<esc>[#C</esc>	Move cursor right by # columns or by one column if # is not spec-
	ified
<esc>[#D</esc>	Move cursor left by # columns or by one column if # is not speci-
	fied
<esc>[#<sub>1</sub>;#<sub>2</sub>f</esc>	Move the cursor to row $\#_1$ and column $\#_2$
<esc>[H</esc>	Move the cursor to the home position (row 0 and column 0)
<esc>[s</esc>	Save the current cursor position
<esc>[u</esc>	Restore the cursor to the previously saved position
<esc>[7</esc>	Same as <esc>[s</esc>
<esc>[8</esc>	Same as <esc> [u</esc>
<esc>[K</esc>	Erase from current cursor position to the end of the line
<esc>[1K</esc>	Erase from current cursor position to the start of the line
<esc>[2K</esc>	Erase entire line
<esc>[J</esc>	Erase from current line to the bottom of the screen
<esc>[2J</esc>	Erase from current cursor position to the top of the screen
<esc>[6n</esc>	Queries the cursor position. A reply is sent back in the format
	$<$ ESC> [ $#_1$ ; $#_2$ R, corresponding to row $#_1$ and column $#_2$ .

Table 2. VT100 commands supported by the Terminal window.

include code for emulating any operations that are needed to execute the C program but which are not supported by the processor. For example, the Nios II Economy version does not include a *multiply* instruction, but the C program may need to perform this operation. By checking this option, a multiply instruction will be implemented in software (by using addition and shift operations).

# 8 Using the Monitor Program with Interrupts

The Monitor Program supports the use of interrupts in Nios II programs. Two example of interrupts are illustrated below, using assembly language code and using C code.

### 8.1 Interrupts with Assembly Language Programs

To see an example using interrupts with assembly language code, create a new Monitor Program project called *Monitor\_Interrupts*. When creating the new project set the program type to assembly language and select the sample program named *Interrupt Example*. Figure 28 lists the source files for this sample program. The main program for the example is the file *interrupt\_example.s*, which initializes some I/O devices and enables Nios II interrupts. The other source files provide the reset and exception handling for the program, and two interrupt service routines.

ource files First source file is used to a	etermine the name of the binary prog	ram file	
D:\Monitor_Terminal_C\JT			Add Remove
rogram options			Up Down
Additional compiler flags:	-01 -ffunction-sections -fverbose-as	m -foo-ioline	
Additional linker flags:			
Use small C library	[	Emulate unimplemented instruction	uctions
urce files highlighted in blue	are sample program files, which will b	e created in the project director	y.

Figure 27. Specifying settings for a C program.

Figure 29 shows the offset values for the *text* and *data* sections that should be used for this program. These offsets cannot be 0 because the reset vector of the Nios II processor in the system being used is at address  $0 \times 0$  and the exception vector is at address  $0 \times 20$ . Enough space has to be left between the exception vector and the text section of the program to accommodate the exceptions processing code, which corresponds to the assembly language code in the file *exception\_handler.s*. The offset value  $0 \times 400$ , as shown in the figure, is large enough to accommodate the exceptions code.

Compile and load the program. Then, scroll the Disassembly window to the label *EXCEPTION\_HANDLER*, which is at address  $0 \times 0000020$ . This address corresponds to the exception vector address for the Nios II processor in the DE2 Basic Computer. As illustrated in Figure 30, set a breakpoint at this address. Run the program. When the breakpoint is reached, single step the program a few more instructions to determine the cause of the interrupt. The source of the interrupt is a device in the DE2 Basic Computer called the *interval timer*. This device provides the ability to generate an interrupt whenever a specific time period elapses. Single step the program until the Nios II processor enters the interrupt service routine for the interval timer. This routine first clears the timer register that caused the interrupt, so that it won't immediately occur again, and then performs other functions needed for the program.

Finally, remove the breakpoint that was set earlier, at address  $0 \times 00000020$ , and then select the **Continue** command to run the program. Observe that the program displays a rotating pattern across the HEX displays on the DE2 board. The direction of rotation can be changed by pressing the pushbuttons KEY<sub>1</sub> or KEY<sub>2</sub> on the DE2 board, and the

New Project V	Vizard	
Specify pro	gram details	
Source files		_
First source file	is used to determine the name of the binary program file.	
D:\Monitor_Pro D:\Monitor_Pro	gram_Tutorials\Monitor_Interrupts\interrupt_example.s Add gram_Tutorials\Monitor_Interrupts\exception_handler.s gram_Tutorials\Monitor_Interrupts\interval_timer.s gram_Tutorials\Monitor_Interrupts\pushbutton_ISR.s	
-Program optior	Up Down	
Start symbol:	start	1
		]
Course files highligh	thad is blue and encounter states which will be supplied in the president discriminant	
Source nies highligr	ited in blue are sample program files, which will be created in the project directory.	
	< <u>Back</u> <u>Next</u> <u>Einish</u> <u>Car</u>	cel

Figure 28. The source files for the interrupt example.

pattern can be changed to correspond to the values of the SW switches by pressing KEY<sub>3</sub>. Pressing KEY<sub>0</sub> causes a reset of the Nios II processor and returns control to the Monitor Program at the address  $0 \times 0$ .

### 8.2 Interrupts with C Programs

To see an example of a C program that uses interrupts, create a new project called *Monitor\_Interrupts\_C*. When creating this project, set the program type to C Program and select the sample program named *Interrupt Example*; this program gives C code that performs the same operations as the assembly language code in the previous example. The source files for the C code are listed in Figure 31. The main program is given in the file *interrupt\_example.c*, and the other source files provide the reset and exception handling for the C program, as well as two interrupt service routines. Complete the steps for creating the project, and then compile and load it.

Set a breakpoint at the address  $0 \times 00000020$ , which is the exception vector address for the Nios II processor. Also, scroll the Disassembly window to the function called *interrupt\_handler*, which is at address  $0 \times 00000698$ . As illustrated in Figure 32, set another breakpoint at this address. Now, run the program to reach the first breakpoint, at address  $0 \times 00000020$ . The code at this address, which is found in the file *exception\_handler.c*, reads the contents of a control register in the Nios II processor to determine if the interrupt is caused by an external device, then saves registers on the stack, and then calls the *interrupt\_handler* function.

ocessor's reset and exce	otion vectors (read-only)	
teset vector address (hex):	0	
Exception vector address (he	.): 20	
emory options		
addresses can be in the same .text and .data sections do n	ting addresses of sections identified by .text and .d or in different memories (on-chip, SDRAM,). The ot overlap with other sections, such as .reset and .e Idress, the .data section will be placed right after th	y can be used to ensure that the exceptions. If .text and .data are
.text section		
Memory device:	SDRAM/s1 (0h - 7fffffh)	-
Start offset in device (hex):		400
.data section		
Memory device:	SDRAM/s1 (0h - 7fffffh)	-
Start offset in device (hex):		400

Figure 29. Memory offset settings for the interrupt example.

Press Actions > Continue in the Monitor Program to reach the second breakpoint. Single stepping the program a few more instructions shows that the interrupt is caused by the interval timer in the DE2 Basic Computer, as discussed in the previous example. Additional single stepping causes the Nios II processor to enter the interrupt service routine for the interval timer, as depicted in Figure 33. This routine first clears the timer register that caused the interrupt, and then performs other functions needed for the program. Finally, clear both breakpoints that were set earlier, at addresses  $0 \times 00000020$  and  $0 \times 00000698$ , and then run the program; it displays a rotating pattern on the HEX displays of the DE2 board, as discussed in the previous example.

## 9 Using Device Drivers (Advanced)

Altera's development environment for Nios II programs provides a facility for using device driver functions for the I/O devices in a hardware system. This facility, which is called the *hardware abstraction layer* (HAL), is supported by the Monitor Program. Using device driver functions is not recommended for beginning students, and is intended for more advanced users.

To see an example of code that uses device driver functions create a project called *Monitor\_HAL*. For this project select the prebuilt system named *DE2 Media Computer*; this is a hardware system that provides more features than the DE2 Basic Computer that was used in previous examples. Set the program type to Program with Device

isassembly oto instruction	Address (hex)							
0x00000020	Address (hex)				_ ×	Registers	5 –	1
		or symbol name:	EXCEPTION_HANDLE	R <u>G</u> o	Hide	Reg	Value	
		EXCEPTION_	- HANDLER.			pc zero	0x00000450 0x00000000	4.6
	defffc04	addi				rl	0x00000000	٩.
	de000015	stw	et, O(sp)			r2	0x00000000	
0x00000028	0031313a	rdctl				r3	0x00000000	
0x0000002c	c0000126	beq		0x00000034: SKIP EA DEC)		r4	0x0000000	
0x00000030	ef7fff04	addi	ea, ea, -0x4	,		r5	0x00000000	
						r6	0x0000000	
		SKIP EA DE				r7	0x0000001	
0x00000034	df400115	stw	ea, 4(sp)		1000	r8	0x0000000	
0x00000038	dfc00215	stw	ra, 8(sp)		393	r9	0x0000000	
0x0000003c	dd800315	stw	r22, 12(sp)			r10	0x00000000	
0x00000040	0031313a	rdctl				r11	0x0000000	
0x00000044	c000011e	bne		0x0000004c: CHECK LEVEL 0)		r12	0x00000019	
0.00000011	00000110		co, 1010, 014 (			r13 r14	0x00000000 0x00000000	
		NOT EI:				r14	0x100000050	
0x00000048	00000706	br	0x1c (0x0000006	8: END ISR)		r16	0x10002000	
						r17	0x00000000	
		CHECK LEVEL	. o.			r18	0x00000000	
•					•	r19	0x00000000	
Disassembly $\int Bre$	eakpoints / Me	mory (Watches	(Trace)			r20	0x00000000	
erminal			_ ×	Info & Errors			_	i
TAG UART link	a establish	ed using cab	le "USB-Blaster	Symbols loaded.				Ī
JSB-0]", devi		-		Source code loaded.				
, ,				INFO: Program Trace not ena	bled, bed	ause tra	ce requires	
				BREAK: Program break @ 0x00			-	

Figure 30. The interrupt handler.

Driver Support, and select the sample program named *Media*. The source file for this sample program is called *media\_HAL.c.* When creating this project, the New Project Wizard does not display the screen for choosing memory settings, such as the one in Figure 29. This is because the HAL automatically chooses the necessary memory settings for projects that make use of device drivers.

The *media\_HAL* program communicates with I/O devices by making calls to device driver functions, rather than using memory-mapped I/O as has been done in previous examples in this tutorial. To see some examples of such function-calls, examine the source code in the file *media\_HAL.c.* It calls device driver functions for the audio devices in the DE2 Media Computer, the 16 x 2 character display, the VGA output port, the PS/2 port, and parallel ports. The device driver functions for each of these devices are defined in *include files* that are specified at the top of the *media\_HAL.c.* file. The set of device driver functions provided for an IP core is specified as part of the documentation for that IP core.

Compile and load the program by using the command Actions > Compile & Load. The Monitor Program automatically compiles both the *media\_HAL.c* program and all device drivers that it uses. In subsequent compilations of the program, only the *media\_HAL.c* code is compiled.

Run the program. It performs the following:

ource files		
First source file is used to	determine the name of the binary program file.	
D:\Monitor_Program_Tuto D:\Monitor_Program_Tuto	rials[Monitor_Interrupts_Clinterrupt_example.c rials]Monitor_Interrupts_Clexception_handler.c rials]Monitor_Interrupts_Clinterval_timer_ISR.c rials]Monitor_Interrupts_Clpushbutton_ISR.c	Add Remove
rogram options		Up Down
Additional compiler flags:	-01 -ffunction-sections -fverbose-asm -fno-inline	
Additional linker flags:	Emulate unimplemented in	structions
	e are sample program files, which will be created in the project direc	

Figure 31. The source files for the C code interrupt example.

		void interrupt_handler(void)	
		{	
		interrupt_handler:	
<b>0</b> x00000698	defffe04	addi sp, sp, -0x8	
0x0000069c	dfc00115	stw ra, 4(sp)	
0x000006a0	dc000015	stw r16, 0(sp)	
		int ipending;	
		NIOS2_READ_IPENDING(ipending);	888
0x000006a4	0021313a	rdctl r16, ipending	
		if ( ipending & Oxl ) // interval timer is interru	upt level 0
0x000006a8	8080004c	andi r2, r16, 0x1	
0x000006ac	10000126	beq r2, zero, 0x4 (0x000006b4)	
		{	
		<pre>interval_timer_isr( );</pre>	
0x000006b0	00006d00	call 0x000001b4 (0x000006d0: interval_timer_isr)	
		}	

Figure 32. The interrupt handler.

- Records audio for about 10 seconds when KEY[1] is pressed. LEDG[0] is lit while recording
- Plays the recorded audio when KEY[2] is pressed. LEDG[1] is lit while playing
- Draws a blue box on the VGA display, and places a text string inside the box
- Shows a text message on the 16 x 2 character LCD display



Figure 33. The interrupt service routine for the interval timer.

• Displays the last three bytes of data received from the PS/2 port on the HEX displays on the DE2 board

More details about developing programs with the Monitor Program that use HAL device drivers can be found in the tutorial *Using HAL Device Drivers with the Altera Monitor Program*, which available on the University Program section of Altera's website. More information about HAL can be found in the *Nios II Software Developer's Handbook*.

## 10 Working with Windows and Tabs

It is possible to rearrange the Monitor Program workspace by moving, resizing, or closing the internal windows inside the main Monitor Program window.

To move a particular window to a different location, click on the window title or the tab associated with the window, and drag the mouse to the new location. As the mouse is moved across the main window, the dragged window will snap to different locations. To detach the dragged window from the main window, drag it beyond the boundaries of the main window. To re-attach a window to the main window, drag the tab associated with the window onto the main window.

To resize a window, hover the mouse over one of its borders, and then drag the mouse. Resizing a window that is attached to the main window will cause any adjacent attached windows to also change in size accordingly.

To hide or display a particular window, use the Windows menu. To revert to the default window arrangement, simply exit and then restart the Monitor Program. Figure 34 shows an example of a rearranged workspace.



Figure 34. The Altera Monitor Program with a Rearranged Workspace.

Copyright ©1991-2011 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, mask work 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.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.

# 11 Appendix A

This appendix describes a number of Monitor Program features that are useful for advanced debugging or other purposes.

## 11.1 Using the Breakpoints Window

In section 3.6 we introduced instruction breakpoints and showed how they can be set using the Disassembly window. Another way to set breakpoints is to use the *Breakpoints* window, which is depicted in Figure 35. This window supports three types of breakpoints in addition to the instruction breakpoint: *read watchpoint*, *write watchpoint*, and *access watchpoint*, described below.

- 1. Read watchpoint: the Nios II processor is halted when a read operation is performed on a specific address
- 2. Write watchpoint: the Nios II processor is halted when a write operation is performed on a specific address
- 3. Access watchpoint: the Nios II processor is halted when a read or write operation is performed on a specific address

Each of the above types of breakpoints requires the use of the *Standard* or *Fast* version of the Nios II processor. These breakpoint types are not available when using the Economy version of Nios II.

In Figure 35 an instruction breakpoint is shown for the address  $0 \times 000058c$ . This corresponds to an address in the program *media\_HAL.c*, which we discussed in section 9. This program uses the DE2 Media Computer, which includes the Standard version of the Nios Ii processor. In section 3.6 we showed how to create such an instruction breakpoint by using the Disassembly window. But we could alternatively have created this breakpoint by right-clicking in a grey box under the label Instruction breakpoint in Figure 35 and then selecting Add. A breakpoint can be deleted by unchecking the box beside its address.

Setting a read, write, or access watchpoint is done by right-clicking on the appropriate box in Figure 35 and specifying the desired address.

The Monitor Program also supports a type of breakpoint called a *conditional* breakpoint, which triggers only when a user-specified condition is met. This type of breakpoint is specified by double-clicking in the empty box *under* the label Condition in Figure 35 to open the dialog shown in Figure 36. The condition can be associated with an instruction breakpoint, or it can be a stand-alone condition if entered in the Run until box in the Breakpoints window. In this example the condition entered is  $r_2 = 5$ , and is associated with the instruction breakpoint. The condition causes the breakpoint to trigger only if the Nios II register  $r_2$  contains the value 5. Note that if a stand-alone condition is entered in the Run until box, then the Run button associated with this box must be used to run the program, rather than the normal Actions > Continue command. The processor runs much more slowly than in its normal execution mode when a conditional breakpoint is being used.

## 11.2 Working with the Memory Window

The Memory window was shown in Figure 24. This window is configurable in a variety of ways:

eakpoints					
akpoints					
		_ ×	Registers	5 _	>
Instruction breakpoint:			Reg	Value	
Address Instruction	Cond	ition	pc	0x0000058c	Π
✓ 0x0000058ccall 0x00000234 (0x000008d0: check F	KEYs)		zero	0x00000000	
			rl	0x00000000	
Read watchpoint:			r2	Oxfffffff	11
Address			r3	Oxfffffff	
Write watchpoint:			r4	0x007fffb8	
Address			r5	0x007fffbc	
			r6	0x007fffc0	
Access watchpoint:			r7 r8	0x0000674c	
Address			10 r9	0x00000001 0x00000187f	
Run until:			r10	0x000001871	
Condition			r11	0x00000088	
Run			r12	0x00000001	
Null			r13	0x00000000	
			r14	0x0000000	
			r15	0x0000000	
			r16	0x0000687c	
			r17	0x00006834	
			r18	0x0000000	
isassembly Breakpoints (Memory Watches Trace			r19	0x007fffc0	
rminal		o & Errors		-	
ened character LCD device		FO: Program Trace not enabled,		trace requir	õ
ened character buffer device		EAK: Program break @ 0x0000058			
ened pixel buffer device		EAK: Program break @ 0x0000058			
ened PS2 device	P. P. P.	EAK: Program break @ 0x0000058			
ened audio device	BF	EAK: Program break @ 0x0000058	Bc		
ened pushbutton KEY device	38				_
ened green LEDs device		nfo & Errors / GDB Server /		•	1

Figure 35. The Breakpoints window.

- Memory element size: the display can format the memory contents as bytes, half-words (2-bytes), or words (4-bytes). This setting can be configured by right-clicking on the Memory window, as illustrated in Figure 37.
- Number of words per line: the number of words per line can be configured to make it easier to find memory addresses, as depicted in Figure 38.
- Number format: this is similar to the number format option in the Register window, described in the previous section, and can be configured by right-clicking on the Memory window.
- Display order: the Memory window can display addresses increasing from left-to-right or right-to-left.

#### 11.2.1 Character Display

The Memory window can also be configured to interpret memory byte values as ASCII characters. This can be done by checking the Show equivalent ASCII characters menu item, accessible by right-clicking on the Memory window, as shown in Figure 39.

The right side of Figure 39 shows a sample ASCII character display. Usually, it is more convenient to view the memory in bytes and characters simultaneously so that the characters appear in the correct sequence. This can be



Figure 36. The Conditional breakpoint dialog.



Figure 37. Setting the memory element size.

accomplished by clicking the Switch to character mode menu item, which can be seen in Figure 39. A sample display in the character mode is shown in Figure 40.

It is possible to return to the previous memory view mode by right-clicking and selecting the Revert to previous mode menu item.

#### 11.2.2 Memory Fill

Memory fills can be performed in the Memory window. Click the Actions > Memory fill menu item or right-click on the Memory window and select Memory fill. A Memory fill panel will appear on the left-side of the Memory

0x00000000	View as	00bffd16
0x00000010	Number of words per line	1
0x00000020		±
0x00000030	Number format	2
0x00000040	Display order	• 4
0x00000050	Switch to character mode	8
0x00000060	Switch to character mode	16
0x00000070	Show equivalent ASCII characters	32
0x00000080		-
0x00000090	Goto memory address	64
0x000000a0	Memory fill	Auto
0x000000b0	Load file into memory	
0x000000c0		da800b17

Figure 38. Setting the number of words per line.

0x00000000	·····	~~	[8 bf	10	16		: H 🗆 🗆	
0x00000010	<u>V</u> iew as	•	8 00	08	00	4000	0 V 0 O	: h 🗆 🗆
0x00000020	Number of words per line	•	12 40	d8	15			
0x00000030	Number format	•	16 40	d9	15			
0x00000040	Display order	•	9 00	da	15		z 0 🗆 🗆	
0x00000050			ld 00	db	15			
0x00000060	Switch to character mode		.1 40	d9	04			000
0x00000070	<ul> <li>Show equivalent ASCII characters</li> </ul>		10 80	28	26		:1 🛛	LOO (
0x0000080		_	13 00	00	15	« D D		
0x00000090	Goto memory address		.1 40	d9	17		: 🗆 = 🗆	
0x000000a0	Memory fill		12 40	d8	17		zp🛛 (	
0x000000b0	Load file into memory		6 40	d9	17			
0x000000c0	Logo nie inco memory		la 40	da	17			0 0 0

Figure 39. Checking the Show equivalent ASCII characters menu item.

window. Simply fill in the desired values and click Fill.

#### 11.2.3 Load File Data into Memory

Data stored in a file can be loaded into the memory by using the Memory window. This feature is accessed by selecting the command Actions > Load file into memory or by right-clicking on the Memory window. The Load file panel will appear on the left side of the Memory window, as illustrated in Figure 41, to allow the user to browse and select a data file. The user provides a base address in memory where the data should be stored. Three types of data files are supported: *Intel HEX-format* files, *Binary* files, and *Delimited hexadecimal value* files. Intel HEX format is a widely-used standard in which data is expressed as records of hexademical values. The Monitor Program supports Intel HEX-format files, which have the filename extension *.hex*, with 8-bit word size. The format of these files can easily be obtained by searching on the internet. The Binary file type is simply an unformatted data file. Each byte in the file is read by the Monitor Program in sequence and stored in the memory.

The Delimited hexadecimal value file format is unique to the Monitor Program. The format of these files is illustrated in Figure 42. The file consists of any number of lines, where each line comprises a comma-separated list of data values. Each data value is expressed as a hexidecimal number with an optional - sign. When using a delimited hexadecimal value file in the Load file panel, two additional parameters can be specified: the value of the delimiter character (comma is the default), and size in bytes of each data value (1 is the default).

Goto memory	add	ress	Ac	Idress	(hex)	:							<u>Go</u> 🗌 C	Query All Devic	es Refresh M	emory Hi	ide
	+0;	кO			+0:	(4			+0:	(8			+0x	+0x0			-
0x00020000	61	62	63	64	65	66	67	68	69	6a	6b	6c	6d	abcd	efgh	ijk.	1
0x00020010	71	72	73	74	75	76	77	78	79	7a	7b	7c	7d	qrst	uvwx	yz {	d.
0x00020020	с9	7f	fe	cf	e5	de	19	ec	a3	d9	fe	7f	bd				
x00020030	32	75	ed	b4	7b	fd	59	69	bb	88	bb	b5	72	2 u 🗆 🗆	{ 🛛 Y i		
0x00020040	ff	ac	сЗ	ff	£5	62	fc	fe	- 77	9e	67	72	de		□ b #: □	w 🛛 g :	r
0x00020050	fl	£3	aЗ	7b	9e	8b	bЗ	74	ab	b7	b9	5£	Зf	000{	OOOt		Л
0x00020060	ce	9d	3f	ad	ff	ed	9f	fe	7f	dd	e8	e6	le	0020			
0x00020070	3d	le	£5	fd	70	£5	Зb	19	96	ce	23	bf	ad	= 🗆 🗆 🗆	рО;О		
0x00020080	b5	db	f8	c4	c3	4e	с9	fd	99	99	fd	fe	fa				
0x00020090	ab	be	bl	9c	73	7f	Зb	ff	b7	7b	bd	ff	bf		s D ; D	0 { 0 (	
0x000200a0	c4	fd	72	8b	£7	dl	60	7b	f9	fb	e7	f9	Зa	OOrO			
0х000200Ъ0	Зc	36	e0	71	74	2f	07	93	ff	bf	ff	72	Зf	< 6 🛛 q	t/00		r
0x000200c0	c4	7e	£6	41	fc	bd	6d	ef	a5	df	d2	a5	£5	0~0A	*:0 m 0		
0x000200d0	Зc	77	bb	db	31	bb	d6	34	fl	bl	9a	3e	fb	< \u0 0	1004		>
0x000200e0	a7	ff	le	dd	df	9e	ed	7d	f9	de	e7	fc	ff				÷
0x000200f0	be	9f	fd	bf	bf	7b	33	53	bf	bb	dd	3f	73		០ { 3 ន		2
0x00020100	db	4a	df	7d	58	ae	eb	af	lf	£7	ef	ff	5c	0J0}	XDDD		
•								1						1		•	Ĩ

Figure 40. Character mode display.

Memory							- >
Goto memory address	Addres	s (hex):		Go	Query All Devices	Refresh Memory	Hide
Load file	Hide		+0x0	+0x4	+0x8	+0xc	-
		0x000103f0	073f5fb9	bdeb6cfb	7ab699be	fffffef2	- 1
Select a file: <u>B</u> i	owse	0x00010400	4b2c8980	046ac2f8	a2108e20	2402c680	- 1
		0x00010410	12210426	15816099	adc96558	205c43df	- 1
		0x00010420	0d546fc0	840854e4	69fc4922	00b206ad	- 1
File type:		0x00010430	346d0032	28198071	90ab009b	a8501112	
Start address (hex):		0x00010440	fe00a8bl	e28062a4	42600851	0c3a808c	
		0x00010450	70032100	04b1830e	50a1870b	le698228	- 1
	Load	0x00010460	0c0e918e	229a2cd0	4640d049	0e25091c	
		0x00010470	£5340e3d	091a9500	3920c206	74642829	
		0x00010480	c0f09124	b1c00a82	eef90181	8e606198	
		0x00010490	98200081	0b386d10	3fb53101	4eb905b2	
		0x000104a0	0b2044c4	6b60d075	998009Ъ9	308c2f18	
		0x000104b0	31e0322c	83052a14	323e0ale	19b02b34	
		0x000104c0	40d46604	4c649cfc	2452cd24	405066e5	- 1
		0x000104d0	d1b0271c	04840all	36983051	203119d3	
		0x000104e0	d9045924	7468949f	dce7lef8	cla656f5	
		0x000104f0	ba3c9e36	53105993	3a45aa2d	99643150	1
•		-					•
 Disassembly / Breakpoi	nts Me	mory / Watches	/ Trace /				

Figure 41. The Load file panel.

## 11.3 Setting a Watch Expression

Watch expressions provide a convenient means of keeping track of the value of multiple expressions of interest. These expressions are re-evaluated each time program execution is stopped. To add a watch expression:

- 1. Switch to the Watches window.
- 2. Right-click on the gray bar, as illustrated in Figure 43, and click Add.





Watches	_ ×
Expression	Value

Figure 43. The Watches window.

3. The *Edit Watch Expression* window will appear, as shown in Figure 44. The desired watch expression can then be entered, using the syntax indicated in the window. In the figure, the expression mem32(sp) is entered, which will display the value of the data word at the current stack pointer address.

ĺ	Edit watch expression 🛛 🔀
	Syntax Register values: pc, r1, r2,
	Number formats: decimal: ### hexadecimal: 0x### octal: 0x## binary: 0b###
	Operators: ==, !=, <, >, <=, >=, &&,    +, -, *, /, %
	Accessing memory: mem8( address ): byte value at address mem16( address ): half-word value at address mem32( address ): word value at address
	mem32(sp)
	Ob Carel
	<u>Ok</u> Cancel

Figure 44. The Edit Watch Expression window.

4. Click Ok. The watch expression and its current value will appear in the table. The number format of a value displayed in the watch expression window can be changed by right-clicking on the row for that value. As the program being debugged is repeatedly run, the watch expression will be re-evaluated each time and its value will be shown in the table of watch values.

### 11.4 The GDB Server Panel (Advanced)

To see this panel, select the GDB Server panel of the Monitor Program. This window will display the low level commands being sent to the GDB Server, used to interact with the Nios II system being used. It will also show the responses that GDB sends back. The Monitor Program provides the option of typing GDB commands and sending them to the debugger. Consult online resources for the GDB program to learn what commands are available.

### 11.5 Running Multiple Instances of the Monitor Program (Advanced)

In some cases, it may be useful to run more than one instance of the Monitor Program on the same computer. For example, the selected system may contain more than one Nios II processor. An instance of the Monitor Program is required to run and debug programs on each available processor. As described in Section 3.1, it is possible to select a particular processor in a system via the **Processor** drop-down list in the *New Project Wizard* and *Project Settings* windows.

The Monitor Program uses *GDB Server* to interact with the Nios II hardware system, and connects to the GDB Server using TCP ports. By default, the Monitor Program uses port 2399 as the base port, and to connect to each processor in a system, the Monitor Program will attempt to use a port located at a fixed offset from this base port. For example, a single system consisting of 4 processors corresponds to ports 2399-2402.

However, the Monitor Program does not detect any ports that may already be in use by other applications. If the Monitor Program fails to connect to the GDB Server due to a port conflict, then the base port number can be changed by creating an environment variable called ALTERA\_MONITOR\_DEBUGGER\_BASE\_PORT and specifying a different number.

It is also possible to have more than one board connected to the host computer. As described in Section 3.1, a particular board can be selected via the Host connection drop-down list in the *New Project Wizard* and *Project Settings* windows. In this case, a separate instance of the Monitor Program is needed to interact with each processor on each physical board. By default, the Monitor Program assumes a maximum of 10 Nios II processors per board. This means that ports 2399-2408 are used by the Monitor Program for the first board connected to the computer, and the first processor on the second board will use port 2409.

It is possible to specify a different value for the maximum number of processors per Nios II hardware system by creating an environment variable called ALTERA\_MONITOR\_DEBUGGER\_MAX\_PORTS\_PER\_CABLE and specifying a different number. This is useful if a system contains more than 10 Nios II processors. It is also useful if a port conflict exists and none of the systems contain 10 or more processors. In this case, decreasing this number (in conjunction with changing the base port number) may provide a solution.

### 11.6 Examining the Instruction Trace (Advanced)

An instruction trace is a hardware-level mechanism to record a log of all recently executed instructions. The *Nios II JTAG Debug Module* has the instruction trace capability, but only if a Level 3 or higher debugging level is selected in the *SOPC Builder* configuration of the JTAG Debug Module (See the *Nios II Processor Reference Handbook*, available from Altera, for more information about the configuration settings of the JTAG Debug Module). If the required JTAG Debug Module is not present, a message will be shown in the Info & Errors window of the Monitor

Program after loading a program, to indicate that instruction trace is not available.

The Trace window in the Monitor Program is provided for future expansion. It is not currently supported.