

Intel FPGA Monitor Program Tutorial for ARM*

For Quartus® Prime 18.1

1 Introduction

This tutorial presents an introduction to the Intel FPGA Monitor Program that can be used to compile, assemble, download and debug programs for ARM* Cortex-A9* processor, which is a processor implemented as a hardware block in Intel's Cyclone[®] V SoC FPGA devices. The tutorial is intended for a user who wishes to use an ARM-based system on an Intel Development and Education board. It gives step-by-step instructions that illustrate the features of the Monitor Program. In addition to supporting the ARM-based programs, the Monitor Program can also be used with the Nios[®] II-based programs. For this application, consult the tutorial *Intel FPGA Monitor Program for Nios[®] II*.

The Monitor Program is a software application which runs on a host PC, and communicates with an ARM-based hardware system on an FPGA board. It can be used to compile/assemble an ARM software application, download the application onto the FPGA board, and then debug the running application. It provides features that allow the user to:

- Set up an ARM project that specifies a desired hardware system and software program
- Download the hardware system onto an FPGA board
- Compile software programs, specified in assembly language or C, and download the resulting machine code into the hardware system
- Display the machine code stored in memory
- Run the ARM processor, either continuously or by single-stepping instructions
- Examine and modify the contents of processor 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

The process of downloading and debugging an ARM program requires an FPGA board that contains the ARM *hard processor system* (HPS) hardware. In this tutorial it is assumed that the reader has access to the DE1-SoC Development and Education board, connected to a computer that has Quartus[®] Prime 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.

1.1 Who should use the Monitor Program

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

2 Installing the Monitor Program

The Monitor Program is released as part of the University Program Design Suite (UPDS). Before the UPDS can be installed on a computer, it is necessary to first install the Quartus Prime CAD software (either the Lite, Standard or Pro Edition) and the Nios II Embedded Design Suite (EDS). A particular release of the Monitor Program can be used only with a corresponding version of the Quartus Prime software and Nios II EDS. This software can be obtained from the on Intel's website at *university.altera.com*.

Once the Quartus Prime software and Nios II EDS are installed, the UPDS can be installed.

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

2.1 Using a Windows Operating System

When using a Windows operating system, perform the following:

- Install the Intel UPDS from the University Program section of Intel's website. It can be found by going to *university.altera.com* and choosing *MATERIALS* followed by *Software* and then *Intel FPGA Monitor Program*. Specify the installed version of Quartus Prime software. 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.
- 3. The installer will display the License Agreement; if you accept the terms of this agreement, then click | Agree to continue.
- 4. The installer now displays the root directory where the FPGA University Program Design Suite will be installed. Click Next.
- 5. 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. These IP Cores provide a number of I/O device circuits that can be used in hardware systems to be implemented on the FPGA board.
- 6. 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.





- 7. Now, the FPGA University Program Design Suite is successfully installed on your computer, so click Finish to finish the installation.
- 8. Should an error occur during the installation procedure, a pop-up window will suggest the appropriate action. Possible errors include:
 - Quartus Prime software is not installed or the Quartus Prime version is incorrect.
 - Nios II EDS software is not installed or the version is incorrect.

2.2 Using a Linux* Operating System

When using a Linux* operating system, perform the following:

- Install the Intel UPDS from the University Program section of Intel's website. It can be found by going to university.altera.com and choosing MATERIALS followed by Software and then Intel FPGA Monitor Program. Specify the installed version of Quartus Prime software. Then click on the TAR item in the displayed table, which links to an installation tarball called altera_upds_setup.tar. Save this file to a directory of your choosing.
- 2. Using a console, navigate to the directory to which the file was saved. Extract the contents of *altera_upds_setup.tar* using the following command: **tar -xf altera_upds_setup.tar**.

🔛 Intel's FPGA Univer	rsity Program Design Suit	e Setup	_		\times
	Choose Install Lo Choose the folder Program Design Su	ocation in which to install Int vite.	el's FPGA U	niversity	
This Intel's FPGA Uni directory for version C:\altera\17	versity Program Design Suit 17.0 of the Intel FPGA sofi .0	e v17.0 must be inst tware. For example,	alled to the	root	
The University Progra	am Design Suite may not wo	ork correctly if installe	ed to anothe	er location	
Intel's FPGA softwa	are v 17.0 Root Directory —				
C:\intelFPGA\17	o		Brow	/se	
Space required: 522. Space available: 2.80	8MB GB				
Nullsoft Install System v	3.0				
		< Back N	ext >	Cano	el

Figure 2. The components that will be installed.

- 3. Among the extracted files is a shell script named *install_altera_upds* which will be used to install the UPDS. Ensure that the script is executable by using the following command: **chmod +x install_altera_upds**.
- 4. Run the installation script with superuser privileges by using the following command: sudo ./install_altera_upds.
- 5. Follow the instructions displayed by the script to complete the installation.

3 Main Features of the Monitor Program

Each ARM software application that is developed with the 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 DE1-SoC board.

If you are using a Windows*9 operating system, then 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 Intel > University Program > Intel FPGA Monitor Program. You should see a display similar to the one in Figure 3.

If you are using a Linux operating system, then start the Monitor Program software by running the *altera-monitor-program* shell script located in *<path to Intel software>/University Program/Monitor Program/bin*. You should see a display similar to the one in Figure 3.



Figure 3. The main Monitor Program display.

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 3 clicking the left mouse button on the File command opens the menu shown in Figure 4. 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 Monitor Program. 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 3 in many ways. Section 8 shows how to move, resize, close, and open windows within the Monitor Program display.



Figure 4. An example of the File menu.

3.1 Creating a Project

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

Select File > New Project to open the *New Project Wizard*, which leads to the screen in Figure 5. The 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 5 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 required.

If the file system directory specified for the project does not already exist, a message will be displayed indicating that this new directory will be created. To select an existing directory by browsing through the file system, click on the **Browse** button. Note that a given directory may contain at most one project.

The Monitor Program can be used with either an ARM-based system or a Nios II-based system. The choice of a processor is made in the window in Figure 5 in the box labeled Architecture. We have chosen the ARM Cortex-A9 architecture for this tutorial.

2. Click Next to advance to the window shown in Figure 6, which is used to specify a particular system. A hardware system to be implemented on the FPGA board is usually generated by using Quartus's Platform Designer tool. Information about creating systems using Platform Designer can be found in the *Introduction to the Intel Platform Designer Tool* tutorial, which is available in the University Program section of Intel's website.

A system designed and generated by using Quartus Prime and its Platform Designer tool is described in *SOPCInfo* and *SOF* files. The former gives a high-level description of the system. The latter represents the

Proiect directo	prv:			
D:\Monitor_Tu	utorial			Browse.
Project name:				
Monitor_Tuto	rial			
Architecture:	ARM Cortex-A9	 	 	

Figure 5. Specifying the project directory and name.

FPGA circuit that implements the designed system; this file can be downloaded into the FPGA chip on the board that is being used.

Any system which contains a *Hard Processor System* (HPS) component must also specify the preloader to be run immediately following the circuit being downloaded. This preloader is used to configure the components within the HPS with the setting required for the specific board.

The drop-down list on the Select a system pane can be used to choose the system to be used in the project. There are several possibilities: a prebuilt system based on one of the DE-series boards, a custom system created by the user, and a generic ARM Cortex-A9 System. Since in this tutorial we assume that the user has access to a DE1-SoC board, we will use a system called the DE1-SoC Computer. This computer includes a number of interfaces to input/output devices implemented in the FPGA fabric of the chip. It was created using Quartus Prime and its Platform Designer tool. It is represented by *.sopcinfo* and *.sof* files which are automatically included when this computer is selected. The DE1-SoC preloader is also automatically selected.

The user may also design and implement a custom system. If the custom system is selected, then the user must manually specify the *.sopcinfo* and *.sof* files that define the required system in the System details pane. If the custom system contains an HPS, the user must select their board from the preloader dropdown menu.

Another option is to use the generic ARM Cortex-A9 system. In this case no design files are used, and only the resources that are directly associated with the HPS part of the FPGA device are available. For example,

application programs that do not involve resources implemented in the FPGA fabric can be run using this system. A preloader must be chosen if the ARM Cortex-A9 system is chosen.

Lets go back to our DE1-SoC computer. In the top right corner of Figure 6 there is a Documentation button. Clicking on this button opens a user guide that provides all information needed for developing ARM programs for the DE1-SoC 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 command Settings > System Settings and then clicking on the Documentation button.

DE1-SoC Computer		-	Documentation	
This system, called the DE1-SoC Co organization and embedded syster processor, memory, audio and vide	mputer, is intended to be used as a ns. To support these experiments, th to devices, and some simple I/O per	platform for exp ie system contair ipherals.	eriments in compute is a number of comp	r onents: a
ystem details System description file (SOPCInfo):				
/Computer_Systems/DE1-SoC/DE	1-SoC_Computer/verilog/Compute	r_System.sopcinf	0	Browse
EPGA programming (SOE) file:				
/Computer Systems/DE1-SoC/DE	1-SoC Computer/verilog/DE1 SoC	Computer.sof		Browse
Preloader				
Preloader DE1-SoC				v

Figure 6. Specifying the desired hardware system.

- 3. Click Next to advance to the screen in Figure 7, 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 ARM assembly-language code.
 - C Program: allows the Monitor Program to be used with C code.

- AXF, ELF or SREC File: allows the Monitor Program to be used with a precompiled program, in AXF, ELF or SREC format.
- No Program: allows the Monitor Program to connect to the ARM hardware system without first loading a program; this can be useful if one wants to examine the current state of some I/O devices without running an actual program.

Rew Project Wizard	×
Specify a program type	
Program Type: Assembly Program	•
Lets you specify a program written in assembly language.	
Include a sample program with the project	
Select a sample program Getting Started Arm A9 Generic Interrupt Interrupt Example JTAG UART Simple Program Arm A9 Timer Example It performs the following: 1. displays a rotating pattern on the LEDs 2. if a KEY is pressed, uses the SW switches as the pattern	
< Back Next > Save Car	ncel

Figure 7. Selecting a program type and sample program.

For our example, set the program type to Assembly Program. When the DE1-SoC computer has been selected for the project, it is possible to click on the selection Include a sample program with the project. As illustrated in Figure 7, several sample assembly-language programs are available for this prebuilt computer. For our tutorial select the program named *simple_program*. This is a very simple program which continuously reads the state of the slider switches on the DE1-SoC board and displays their state on the red LEDs. The source code for the program is:

```
.text
.equ LEDs, 0xFF200000
.equ SWITCHES, 0xFF200040
.global _start
_start:
    LDR R1, =LEDS /* Address of red LEDs. */
    LDR R2, =SWITCHES /* Address of switches. */
LOOP: LDR R3, [R2] /* Read the state of switches. */
    STR R3, [R1] /* Display the state on LEDs. */
    B LOOP
.end
```

Click Next to advance to the screen in Figure 8.

C:/Desktop/N	Aonitor_Tutorial/getting_started.s	Add
C./Desktop/w	ionitor_rutorial/address_map_arm.s	Remove
		Up
		Down
rogram ontig	ons	
ogram optic		



When a sample program has been selected, the source code file(s) associated with this program is listed in the **Source files** box. In this case, the source file is named *simple_program.s*; this 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 8 shows that the first instruction is indicated by the label *_start*. In the ARM architecture this is not editable.

4. Click Next to advance to the window in Figure 9. This window is used to specify the connection to the FPGA board, the 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 ARM 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 5.

Accept the default choices that are displayed in Figure 9. If the Host Connection box is blank, make sure that the DE1-SoC board is connected to the host by a USB cable and that its power is turned on. Then, press the **Refresh** button and select the USB Blaster as the desired choice. For the DE1-SoC board the required choice is DE-SoC.

5. Click Next to reach the final screen for creating the new project, shown in Figure 10. This screen is used to specify memory settings that are needed for compiling and linking the program.

There are two modes that can be selected. In the **Basic** mode, which does not provide explicitly for the use of interrupts, the application program starts at memory address 0x00000000 as shown in the figure. A more general alternative is to use the Interrupts mode. In this case, a *.vectors* section occupies the memory locations 0x00000000 to 0x0000003F, as described in Section 7. This space is used for interrupt and exception vectors. The main program in the *.text* section may start at address 0x00000040. However, it can also start at some other address, as may be specified by the user. To change the address, double-click on the *.text* entry and change the address in the pop-up box that appears.

Click Finish to complete the creation of the new project. At this point, the Monitor Program displays the prompt shown in Figure 11. Clicking Yes instructs the Monitor Program to download the hardware 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 System. If the downloaded system contains more than one processor, the Monitor Program will prompt you to halt the processors other than the one being used for the current project. It is generally recommended to halt the other processors because they can execute without you knowing, resulting in unexpected behavior.

3.2 Compiling and Loading the Program

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

• Actions > Compile menu item or icon: compiles the source files into an AXF and SREC file. Build warnings and errors will show up in the Info & Errors window. The generated AXF and SREC files are placed in the project's directory.

New Project Wiz	rard	
pecify system	n parameters	
System paramet	ters	
Host connection	n: DE-SoC [USB-1] Ret	fresh
Processor:	ARM_A9_HPS_arm_a9_0	-
	Don't reset the processor when loading a program (ARM only)	
Terminal device	TIAG HART for ARM 0	•
reminar device	* <mark>////////////////////////////////////</mark>	

Figure 9. Specifying system settings.

- Actions > Load menu item or $\stackrel{\clubsuit}{=}$ 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 is 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). Select 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 12.

After successfully completing this step, the Monitor Program display should look similar to Figure 13. At this point the processor is halted at the first instruction of the program that has to be executed, which is highlighted in yellow shading. The main part of the display in Figure 13 is called the *Disassembly* window. It shows the machine code for

ecify program memo	ry settings	
Memory options		
Here you can specify section linker to place code and da the linker, the names must	n names and their start and end a ita at the specified addresses. To a match those identified by the asse	addresses. These sections will be used by the ensure correct use of the section names by embler directives, such as .text.
Linker Section Presets: Ba	sic	Ŧ
Section Name	Memory Device	Address Range
.text	ARM_DDR3_SDRAM	0x00000000 - 0x3FFFFFFF

Figure 10. Specifying memory settings.

Download System - Prompt	×
Would you like to d If so, make sure that	wnload the system associated with this project onto the board? the board is connected via the correct cable and is powered up.
	Yes <u>N</u> o

Figure 11. Download the hardware system.

the assembled program, as well as the addresses of memory locations in which the instructions are loaded. It also shows the assembly-language version of the assembled instructions.

Most instructions in an ARM assembly-language source program are assembled into directly-corresponding machine instructions in the object code that is loaded into the memory for execution. However, this is not the case with all



Intel FPGA Monitor Program - Monitor_Tutorial : simple_program.srec [Paused]							— D			
<u>File Edit Actions Windo</u>	ws <u>H</u> elp									
🗇 🗟 🚸 🚵 🔶 👃 .	1 🔿 🕩 🗉	0. N I								
Project Files $ \times$	Disassembly						_ ×	Registers		_ ×
Monitor_Tutorial	Goto instruction	Address (hex) or symbol n	name:		Go	Hide	Reg	Value	
Simple_program.s								pc	0x0000000	
								r0 n1	OVEFFF139C	
				.includ	le "address man a	arm.s"		r2	0x00000001	
				1 2110 2 444	ic dualette_map_c			r3	0xFFFEFC00	
				.text				r4	0xFFFF8C00	
				.global	start			r5	0xFFFF4AE4	
			_start:					r6	0x00000076	
				LDR	R1, =LEDR_BASE	E /* Address	of red LEDs.	r7	0xFFFFF014	
			_start:					20	0x00000005	
	0x00000000	E59F100C	1dr	r1, [p	c, #12] ; 14	<icce0ir+0x4></icce0ir+0x4>	22	r10	0xFFD02000	
				LDR	R2, =SW_BASE	/* Address o	f switches. * 🔤	r11	0xFFD02000	
	0X0000004	£59F200C	Tar	r2, [p	iC, #12] ; 18	<tccedir+0x8></tccedir+0x8>		r12	0x0000001C	
			LOOP	IDB	D2 (D2)	/t Dead the	state of guit	sp	0xFFFF8BF0	
			LOOP	LUR	KOV [KE]	/- Keau uie	Scace of Swit	lr	0xFFFF135B	
	0x00000008	E5923000	ldr	r3. [r	21			cpar	0x200001D3	
				STR	R3, [R1]	/* Display t	he state on L			
	0x0000000C	E5813000	str	r3, [r	1]					
				В	LOOP					
	0x00000010	EAFFFFFC	b	8 <loop></loop>	•		-			
		8800000		3388777						
	Editor Disasse	mbly Break	points / Mer	nory / Wat	tches / Trace /					
Terminal				_ >	× Info & Errors					_ ×
JTAG UART link establish	ed using cable	"DE-SoC IUS	B-11", dev	vice 2.	INFO: Non-mer	mory - ARM A9	HPS dcan0 0xffc	00000		-
instance 0x02	-				INFO: Non-mer	mory - ARM A9	HPS dcan1 0xffc	01000		
					INFO: Non-mer	mory - ARM A9	HPS 13regs 0xff	800000		
					INFO: Non-mer	mory - ARM_A9	HPS_sdrctl 0xff	c25000		
					INFO: Non-men	mory - ARM_A9	HPS_timer 0xfff	ec600		
					INFO: Non-mer	mory - ARM_A9_	HPS_scu 0xfffec	000		11
										-
						8888	11.			
					Info & Errors	GDB Server / De	bug / Variables /			

Figure 12. Compilation and loading messages.

Figure 13. The Monitor Program window after loading the program.

instructions. The ARM assembly language provides numerous *pseudo-instructions*, which are often replaced by actual instructions that look quite different but have the same effect when executed. For instance, the instruction

LDR R1, =LEDs

loads into processor register R1 the memory address of the I/O data register that is connected to the LEDs on the board. As seen in Figure 13, this instruction is replaced with the instruction

LDR R1, [PC, #12]

in the assembled code. Since Load instructions in the ARM processor cannot specify an immediate operand that is 32 bits long, the address 0xFF200000 is placed in the *literal pool* after the last instruction in the program. Then, the implemented LDR instruction uses the *Relative* addressing mode (which is the *Offset* addressing mode that uses the Program Counter as the base register) to access the desired address value. Observe that the offset used in this case is 12 bytes. The reason is that the ARM processor prefetches two instructions to facilitate pipelined execution of the program. When an instruction is prefetched, the Program Counter is incremented by four. Thus, in our example, the updated PC contents will be 0x08 when the first LDR instruction is being executed. Then, the offset of 12 bytes leads to the memory location 0x14.

Note that in an ARM assembly-language program it is possible to use both upper- and lower-case letters to denote register names and instruction mnemonics.

Information about the ARM instructions, addressing modes and literal pools can be found in the tutorial *Introduction* to the ARM Processor Using Intel Toolchain, which is available in the University Program section of Intel's website.

3.2.1 Compilation Errors

During the process of developing software, it is likely that compilation errors will be encountered. Error messages from the ARM 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 *simple_program.s*, which is in the project's directory, and replace the mnemonic STR with ST. Recompile the project to see the error shown in Figure 14. The error message indicates the type of error and it gives the line number in the file where the error was detected. Fix the error, and then compile and load the program again.

Info & Errors >									
Compiling source	Compiling source files								
arm-altera-eabi-as -mfloat-abi=soft -march=armv7-a -mcpu=cortex-a9gstabs -I "\$GNU_ARM_TOOL_ROOTDIR/arm-al D:/Monitor_Tutorial/simple_program.s:10: Error: bad instruction `st R3,[R1]'									
Compilation sto	opped.								
		• • • • • • • • • • • • • • • • • • •							
•	1000000								
Info & Errors	GDB Server								

Figure 14. An example of a compiler error message.

3.3 Running the Program

As mentioned in the previous section, the processor is halted at the first instruction after the program has been loaded.

To run the program, select the Actions > Continue menu item or click the icon. The *simple_program* displays the current values of DE1-SoC board's slider switches on the red LEDs. The Continue command runs the program

indefinitely. To force the program to halt, select the Actions > Stop command, or click the icon. This command causes the processor to halt at the instruction to be executed next, and returns control to the Monitor Program.

Figure 15 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 to be executed, which is at address 0×0000000 C, and highlights in red the values in the processor registers 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.

🥔 Intel FPGA Monitor Program	- Monitor_Tutoria	l : simple_prog	ram.srec [Pau	ised]					- 🗆	×
<u>File Edit Actions Windo</u>	ws <u>H</u> elp									
ቅ ଲି ቀ ଲି ቀ መ 🖬 🖕 🖉 🖕										
Project Files _ ×	Disassembly						_ ×	Registers		_ ×
൙ Monitor_Tutorial	Goto instruction	Address (hex) or symbol n	ame:		Go	Hide	Reg	Value	
Simple_program.s			,,					pc	8000000x0	
							<u> </u>	rO	OxFFFFFFFF	
				in al	ude Kedduese nen enn			r1	0xFF200000	
				. Inci	uude audress_map_arm.:	. 3		73	0x00000000	
				text				r4	0xFFFF8C00	
				. glob	al start			r5	0xFFFF4AE4	
			start:	- 9	_			r6	0x0000076	
			-	LDR	R1, =LEDR BASE /	/* Address of	red LEDs.	r7	0xFFFFF014	
			_start:					r8	0xFFFF5E28	
	0x00000000	E59F100C	ldr	r1,	[pc, #12] ; 14 <ic< td=""><td>CEOIR+0x4></td><td></td><td>r9</td><td>0x00000005</td><td></td></ic<>	CEOIR+0x4>		r9	0x00000005	
				LDR	R2, =SW_BASE /*	Address of a	switches. * 🜌	r11	0xFFD02000	
	0x00000004	E59F200C	ldr	r2,	[pc, #12] ; 18 <ic< td=""><td>CEOIR+0x8></td><td></td><td>r12</td><td>0x0000001C</td><td></td></ic<>	CEOIR+0x8>		r12	0x0000001C	
								sp	0xFFFF8BF0	
			LOOP:	LDR	R3, [R2] /*	Read the sta	ate of swit	lr	0xFFFF135B	
			LOOP:					cpar	0x200001D3	
	0x0000008	E2923000	Idr	r3,	[r2]	. Di su lass dha				
	000000000	FF912000	ata	SIR	R5, [R1] /*	· Dispiay the	state on L			
	0x00000000	F2012000	str	в.	LOOD					
	0×00000010	EAFFFFC	h	8 <1.00	IP>					
	0.00000010									
				1000000	8		•			
	Editor Disasse	mbly Break	points / Men	nory / W	Vatches / Trace /					
Terminal				_	. × Info & Errors					_ ×
JTAG UART link establishe	ed using cable	"DE-SoC [US	B-1]", dev	vice 2,	INFO: Non-memory	V - ARM A9 HP	S dcan1 0xffc0	1000		•
instance 0x02	-				INFO: Non-memory	V - ARM A9 HP	S 13regs 0xff8	00000		
					INFO: Non-memory	V - ARM A9 HP	S sdrctl 0xffc	25000		
					INFO: Non-memory	y - ARM A9 HP	S_timer 0xfffe	c600		
					INFO: Non-memory	y - ARM_A9_HP	S_scu 0xfffec0	00		
					Program stopped	@ 0x0000008	8			20
										-
						3382228				
					Info & Errors / GDE	B Server / Debu	g / Variables /			

Figure 15. The Monitor Program display after the program has been stopped.

3.4 Using the Disassembly Window

In Figure 15, the Disassembly window shows the machine instructions for our program. The leftmost column in the window gives the memory addresses, the middle column displays the machine code at these addresses, and the rightmost column shows the corresponding assembly-language instructions.

The Disassembly window can be configured to display less information on the screen, such as not showing the assembly-language instructions or not showing the machine encoding of the instructions. These choices can be made by right-clicking on the Disassembly window and selecting the appropriate menu item, as indicated in Figure 16.

Different parts of memory can be 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 go to a different region of memory by using the Goto instruction panel at the top of the Disassembly window, or by using the command Actions > Goto

		LOOP:		
0x0000008	E5923000	1dr r3, [1	:2]	Show instruction words
		STR R3, [H	a) 🚺	Show instruction words
0x0000000C	E5813000	str r3, [1	:1] 🗸	Show source code
		B LOOP		Dillow Source could
0x00000010	EAFFFFFC	b 8 <1.00P)	>	Goto instruction
0x00000014	FF200000	.word 0xfi	200000	<u>o</u> oto instruction
0x00000018	FF200040	.word 0xfi	200040	

Figure 16. Display options for the Disassembly window.

instruction. The instruction address provided for the Goto command must be a multiple of four, because ARM instructions are word-aligned.

3.5 Single Stepping Through Program Instructions

When debugging a program, it is often very useful to be able to single step through the program and observe the effect of executing each instruction. The Monitor Program has the ability to perform single-step operations. Each single step consists of executing a single machine instruction and then returning control to the Monitor Program. If the source code of the program being debugged is written in the C language, then 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 yellow in the Disassembly window. Consider our *simple_program* example. You can go to the first instruction of the program, which has the label *_start*,

by selecting Actions > Restart menu item or by clicking the \square_{\bullet} icon. If the program is running, it must first be halted before the restart command can be performed. The restart command loads into the Program Counter the address of the first instruction, thus causing the execution to start at this point in the program. Now, single step through the program and observe the displayed changes. Note that the register values are indicated in red when they change as a result of executing the last instruction.

3.6 Using Breakpoints

An *instruction breakpoint* provides a means of stopping the execution of a program when it reaches an instruction at a specific address. The procedure for setting a breakpoint is:

- 1. In the Disassembly window, scroll to display the instruction that will have the breakpoint. For example, in the window in Figure 15 scroll to the Branch instruction at address 0x00000010.
- 2. Click on the gray bar to the left of the address 0000010. As illustrated in Figure 17, the Monitor Program displays a red dot next to the address to show that a breakpoint has been set. Clicking the same location again removes the breakpoint.

Once the instruction breakpoint has been set, run the program. The breakpoint will trigger when the Program Counter value equals 0×0000010 . 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 Info & Errors pane.

🥔 Intel FPGA Monitor Program	n - Monitor_Tutorial	: simple_prog	ram.srec [Paus	ed]				- 🗆	×
<u>File Edit Actions Windo</u>	ws <u>H</u> elp								
🗢 🗟 + 🗟 🔶 👃 .	1 🔿 🕩 🖬 🤇	ls № ∲							
Project Files _ ×	Disassembly					_ ×	Registers		_ ×
➢ Monitor_Tutorial	Goto instruction	Address (hex) or symbol nar	me:	Go	Hide	Reg	Value	
simple_program.s							pc	0x00000000	
				.tex			r1	0xFF200000	
				. glol	start		r2	0xFF200040	
			start:				r3	0x00000000	
			-	LDR	R1, =LEDR BASE /* Address of red L	EDs.	r4	0xFFFF8C00	
			start:				r5	0xFFFF4AE4	
	0x00000000	E59F100C	ldr	r1,	c, #12] ; 14 <icce0ir+0x4></icce0ir+0x4>		r6	0x0000076	
				LDR	R2, =SW_BASE /* Address of switch	es. *	r7	0xFFFFF014	
	0x00000004	E59F200C	ldr	r2,	c, #12] ; 18 <icce0ir+0x8></icce0ir+0x8>		r8	0xFFFF5E28	
							r9	0x00000005	
			LOOP:	LDR	R3, [R2] /* Read the state of	swit	r10	0xFFD02000	
			LOOP:				r12	0x0000001C	
	0x0000008	E5923000	ldr	r3,	2]		an	0xFFFF8BF0	
				STR	R3, [R1] /* Display the state	on L	lr	0xFFFF135B	
	0x0000000C	E5813000	str	r3,	1]		cpar	0x200001D3	
				в	LOOP				
	0 x00000010	EAFFFFFC	b 8	- 1 0	•				
	0x00000014	FF200000	.word	0:	200000				
	0x00000018	FF200040	.word	0:	200040				
	0x0000001C	00000000	??			-			
				-		•	1		
	Editor Disasse	mbly / Break	points / Memo	ory / 1	tches / Trace /		-		
Terminal				_	× Info & Errors				_ ×
TTAG HADT link establish	ed using cable	"DE-Soc ITT	B-11" dorri	ca 2	THEOR MAN AND AND A THEORY	1 0.055	01000		-
instance 0x02	cu using cable	55-30C [03	uevi	.ce 2,	INFO: Non-memory - AKM A9 HP5 dcar	L UXIIC	01000		
Instance 0x02					INFO: Non-memory - ARM_A9_HPS_13re	gs Oxfi	-25000		
					INFO: Non-memory - ARM_A9_HP5_8dro	CI UXII	025000		
					INFO: Non-memory - ARM A9 HPS_time	Ovfffor	1000		
					Program stopped & 0x00000008	UXITIE	5000		1000
					Lindiam scobben 6 0x0000000				
					4				
						-1-1 /			
					Info & Errors / GDB Server / Debug / Var	ables			

Figure 17. Setting a breakpoint.

3.7 Examining and Changing Register Values

The **Registers** window on the right-hand side of the Monitor Program display shows the values of processor registers. It also allows the user to edit most of the register values. The number format in which the register values are displayed can be changed by right-clicking in the **Registers** window and selecting the desired format, as illustrated in Figure 18.

Registers	_	\times
Reg	Value	
рс	0x000000C	
r0	0xFFFF8104	
r1	0xFF200000	
r2	0xFF200040	
r3	0x0000056	
<u>B</u> inary Octal		
<u>o</u> ctui		
<u>D</u> ecim	al	
• <u>H</u> exad	ecimal	
<u>S</u> igned	representa	tion
ah	UNFFFFHJUU	_

Figure 18. 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 edit the register values while the program is halted. Any edits made are visible to the processor when the program's execution is resumed.

As an example of editing a register value, set the slider switches on the DE1-SoC board to some pattern of 0s and 1s. Run the *simple_program* and observe that the LEDs display the selected pattern. Next, stop the execution of the program and set a breakpoint at the Store instruction at address 0×0000000 C. Run the program and after the execution stops at the breakpoint, observe that the value in register R3 corresponds to the current setting of the slider switches. Now, as indicated in Figure 19, double-click on the contents of register R3 and change them to the value FFF. Press Enter on the computer keyboard, or click away from the register value to apply the edit. Then, single-step the program to see that all LEDs will be turned on.

Disassembly									_ ×		Registers	$-\times$
Goto instruction	Address (h	ex) or sym	nbol na	me:			Go		Hide	1	Reg	Value
	1									4	pc	0x000000C
		LOOP:									rO	0xFFFF8104
0x00000008	E5923000	ldr	r3,	[r2]							r1	0xFF200000
		STR	R3,	[R1]	/* Display	the :	state or	n LEDs.	*/		r2	0xFF200040
0x00000000	E5813000	str	r3,	[r1]							r3	0x00000FFF
		В	LOOP								r4	0xFFFF80E4
0x00000010	EAFFFFFC	b	8 <l0< th=""><th>)P></th><th></th><th></th><th></th><th></th><th></th><th></th><th>r5</th><th>0x01000040</th></l0<>)P>							r5	0x01000040
0x00000014	FF200000	.wor	d 0:	kff200000							r6	0x0000076
0x00000018	FF200040	.wor	d 0:	xff200040							r7	0xFFFFF014

Figure 19. Editing a register value.

3.8 Examining and Changing Memory Contents

The Memory window, depicted in Figure 20, 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 20, the address of the third word in the second row is $0 \times 00000010 + 0 \times 8 = 0 \times 00000018$. The displayed contents of this memory location are FF200040, which is the address of the slider switches that is placed into the *literal pool* when the pseudo-instruction

LDR R2, =SWITCHES

is assembled.

If a 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 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 cause the window to display memory-mapped I/O locations, click on the check mark beside Query Devices, and then click Refresh. For example, set the slider switches to some pattern and press Refresh. Figure 21 shows the display we obtained when choosing the pattern 0x30F.

The color of a memory word displayed depends on whether that 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, as in Figure 20. Memory-mapped I/O is shown in 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, as in Figure 21.

Bit Edit Actions Windows Help roject Files - × Monitor_Tutorial - ×
Image: Solution of the
roject Files × Memory - × Registers × * Monitor_Tutorial
Monitor_Tutorial Got address (hex or symbol name): Go Query Devices Refresh Hidd Reg Value imple program.s +0x:0 +0x:4 +0x:4 +0x:4 +0x:0 +1:1 +1:2 +1:2 -0 xe:0 -1:2 -0 xe:0 +0x:0 +0 +0
→ simple_program.s +0x4
+0x0 +0x4 +0x6 +0x6 0x00000000 ESF1200C ES52000C ES530000 ES13000 0x00000010 EAFFFFFC F7200000 F7200040 00000000 0x00000010 EAFFFFFC F7200000 F7200040 00000000 0x00000010 EAFFFFFC F7200000 F7200040 00000000 0x00000010 EAFFFFFC F7200000 F7200040 10 0x0000000 0x00000000 EFFETDEI ALF0DSF E3DAB52E E2BFFA0 10 0xFFFF0044 0x00000000 F4F6E95D 33EE195 6063295F 633D6CDB 12 0xFFFF14244 0x00000000 F4F6E95D 33EE195 6063295F 633D6CDB 12 0xFFFF1424 0x00000000 F2632707CA SEDB53C F3DE1677 0xFFFF1528 9 0x00000005 0x00000000 CBAFFFF D5DA552D SCA220F 0x11 0xFFF15287 10 0xFFF15287 0x00000000 CESDF757 SF5555 SEADT794 CS77753 11 0xFFF15287 1x 0x20000015 12 0x00000000 <
CX0000000 ES972000 ES9720000 ES9720000 ES9720000 F1 CXF7200000 CX00000010 ES9720000 F2000000 00000000 00000000 F2 CXF7200000 CX00000010 ES95555 SS5555 SS5555 SS5555 SS5555 CXF7FFC00 CX00000000 EFFETDEL ALF055F E3DA552 E2B6FA0 F3 CX0000076 CX00000000 EFFETDEL ALF055F E3DA552 E2B6FA0 F3 CX0000076 CX00000000 EFFETDEL ALF055F E3DA552 E2B6FA0 F3 CX0000076 CX00000000 EFFETDEL ALF055F F350CDB F3 CX00000076 F3 CX00000076 CX00000000 EFFETDEL CYTTPERSEC CYTPTCA SEEDEF5 G050E577 F3 F3 CX00000007 F1 CXTFTF0240 CX000000000 EAFFF7 EFF0EEF7 F7DF5C F5FFF1C CX00000000 F1 CXTFTF024 F3 CX00000000 F1 CXTFTF024 F3 CX00000000 F1 CXTFTF024 F3 CX00000000 F1 CXTFTF024
0x00000010 CAFFFFC F2200040 00000000 0000000 0000000 000000000 0000000000 0000000000
0x0000020 00000000 00000000 00000000 00000000 0x00000000 0FFETDB1 ALFODSP E3DAB52E E2BBFEAD 0x00000000 FFETDB1 ALFODSP E3DAB52E E2BBFEAD 0x00000000 FFETDB1 ALFODSP E3DAB52E E2BBFEAD r5 0x1FFF142A4 0x00000000 F4F6ES9D 33EEE195 6063295F 633D6CDB r7 0xFFFTF142A 0x00000000 F4F6ES9D 33EEE195 6063295F 633D6CDB r7 0xFFFTF152E 0x00000000 C2G3A76 6FALDB74 EDP55513 F2ABE779 r10 0xFFTF022000 0x00000000 CBABFTF7 D5DA652D 9C42228F r9 0x0000000 0x00000000 CBABFTF7 D5DA652D 9C42228F r10 0xFFTF0202000 0x00000000 CBABFTF7 D5DA652D 9C42228F r0 0x00000000 0x00000000 CBABFTF7 D5DA652D 9C42228F r0 0xFFFTB20 0x00000000 CFTBF56 64DA107B7 35BF62AD r0 0x0000010 r0 0x02FFTF135B
0x00000000 03030333 03030303 03030303 03030303 0x00000000 0FBTETDB1 A1FD05BF 0305BF2A0 15 0x1FFF04A4 0x00000000 FEDFB35 ED62BE2 722BABA 72EA057E 16 0x0000000 0x00000000 FF0FB55 C502B59F 6330CDB FE 16 0x0000000 0x00000000 FF0EBE7C C50707CA SEEDB35C AFDB16F7 0xFFFF02A4 0x00000000 FF1BEFC C5707FCA SEEDF35 0652B57 732BABA F9 0x0000000 0x00000000 EFTDEETC C5707FCA SEEDF35 052B577 750F571 10 0xFFF002000 0x00000000 EE3E3A22 6DA252D 9C422087 72 0x0000000 11 0xFFF002000 0x00000000 FE3E3A32 6DA253D 9EAL07B7 C58FF71 0x1 0xFF7002000 0x00000000 FE3E3A32 6DA253D 9FE3C57 739D775 0x17F75785 0x20000102 0x00000000 FE3E3A32 6DA252D 9C422087 9FE3C4D 0x20000103 0x200000102 0x00
0x00000000 FB07BB35 ED6258E3 772B6ABA 7EEA057E 6 0x0000006 0x00000000 F4762950 332EEL195 60528957 63306CDB 7 0xFFFF7014 0x00000000 F4762950 332EEL195 60528957 63306CDB 7 0xFFFF7014 0x00000000 FF7DEFC C5707FCA 92EDB35C AF95E4PA 7 0xFFFF522 0x00000000 EF7DEFC C5707FCA 92EDB35C AF95E4PA 7 0xFFF7522 0x00000000 BECFFF7 EFD6E571 57DF75C F5BF511 F2AET79 710 0xFF7020000 0x00000000 BECFFF7 D5DE7C7 D5DA52D 9C42226F 0xFF7F5757 710 0xFF7F5855 0x00000000 F2LAB12C LEDC2D7 SFF5CAD 0x00000010 0x017877 6557CAD 0x20000120 92AF7F71358 cpar 0x20000120 0x00000000 F2LAB12C LEDC2D7 SFF5CAD 0x20000120 0x20000120 0x20000120 0x20000120 0x20000120 0x20000120 0x2000
0x00000060 F4F6E95D 33EE195 6063E95F 633D6CDB 0x00000070 FF46E2A4 F991B669 E72B92C6 F95E4DFA 0xFFFFF014 0x00000090 22C33A76 6FALDE74 ED95513 F2AEE779 10 0xFFF702000 0x00000090 22C33A76 6FALDE74 ED955513 F2AEE779 10 0xFFF702000 0x00000000 CEABFFF7 D5A652D 9C42228F 0x0000000 12 0x0000000 0x00000000 07DB76F 645937F 756DE777 36E76CAD 0x0000010 0x20000103 0x20000103 0x00000010 07DB76F 645937F 3750E2F 1876E785E 1876E785E 1876E785E 1876E785 0x20000
0x00000000 FF4BE2A4 FB91B869 EF2B89C6 FB5E4DFA 0x00000000 EFTDEETC C5707FLA 9EEDB35C AFDB16F7 0x00000000 EFTDEETC C5707FLA 9EEDB35C AFDB16F7 0x00000000 EFTDEETC S5707FLA 9EEDB35C AFDB16F7 0x00000000 EFTDEETF PF06EFT5 SF70F51 FX8E779 9C00000000 0x00000000 EESB83A2 6DDA2591 9EAD775 PS6F571 0xFF775780 0x00000000 FE1AB1C1 1EUC207 SFF75556 EBAD7794 0xFF775780 0x00000000 FE1AB1C1 1EUC207 SFF75570 0xFF7758780 0xF77757877 0x00000000 FE1AB1C1 SFF7570 SFF7570 0xF77757860 0x20000103 0x00000000 FE1AB1C1 SFF7570 SFF7570 OxF777578 OxF777578 0x00000000 FE1AB1C1 SFF7570 SFF7570 Ox777579 Ox7777578 0x00000010 C75786E G739CC2013 SFF7579 SF00522 SF75780 <
0x00000000 EFFDBEFC C5707FCA 9EEDB35C AFDB16F7 0x00000000 29C33A76 6FALD574 EP55513 F2ABE779 0x00000000 29C33A76 6FALD574 EP55513 F2ABE779 0x00000000 CEABEFF7 D5DE9TC7 D50A6520 9C42226F 0x00000000 CATTERD2000 r1 0xFTF135B 0x00000000 07DB7F6 6459377F F76DE773 36EF6CAD 0x00000000 07DB7F76 6459377F F76DE773 36EF6CAD 0x200001D3 0x20000010 F737ECCL E6J5566E F27AF7A53 Ox200001D3 0x200001D3 0x00000010 F77566B6 6737ECCL E6J5566E F27AF7A53 Ox200001D3 0x200001D3 </td
0x00000000 29C35A76 6FALDB74 EDF55513 F2ABE779 0x00000000 0ECFTETP9 ETD6EET8 ST/DF75C F3BF5F1C 0x010 0x00000000 EECFTETP9 ETD6EET8 ST/DF75C F3BF5F1C 0x010 10 0xFTD02000 0x00000000 EE3E332 6DA2581 9EALD78F CS47757 11 0xFTF8E70 11 0xFTF8E70 0x00000000 VE12B332 6DA2581 9EALD78F CS47757 11 0xFTF8E70 11 0xFTF8E70 11 0xFTF8E70 11 0xFTF8E70 11 0xFTF8E70 11 0xFTF8E70 0x0000010 0x15757 11 0xFTF8E70 11 0xFTF8E70 0x20000103 0x757575 11 0xFTF8E70 11 0xFTF8E70 11 0xFTF8E70 0x20000103 0x2
0x000000A0 GECFFEPS ETDOBETOS SFIDP75C FSBFSFLC 0x00000A0 CEBAFFF7 DSDEDTC7 DGDAS2D 9C42226F 12 Ox00001C 0x00000000 CEBAFFF7 DSDEDTC7 DGDAS2D 9C42226F 12 Ox00001C 0x00000000 CEBAFFF7 DSDEDTC7 DGDAS2D 9C42226F 12 Ox00001C 0x00000000 CEBAFF77 DSDEDTC7 DSDEDTC7 SEPCETO7
0x00000000 CBABFFF7 D5DEB7C7 D6DA6520 9C42228F PC
0x00000000 EE3E33.82 60PA25B1 9EALD78F CS67F757 0x1F7757 0x1F7757 0x00000000 7E1BA13.2 11EDC2D7 SFFE556 EBADE784 0x1F77573 0x1F77573 0x00000000 07D3676F 6459377F 776D8777 35EFC4AD 0x1F7757336 0x200001D3 0x00000100 07D3676F 6459377F 37ED3C56 0x200001D3 0x200001D3 0x00000101 C777586B0 6737ECC0 E60586B2 F2FAFFA3 0x200001D3 0x00000102 DAFA76AF P4CEF702 AFABE33 FEED48B V Editor / Disassembly / Breakpoints Memory / Watches / Trace / V Info & Errors - ×
0x00000000 7ELABISC 11EDC2D7 SFFE5656 EEADE784 0x00000000 07DB76F 6459377F 776DB777 36EF6CAD 0x200001D3 0x00000000 07DB76F 6459377F 776DB777 36EF6CAD 0x200001D3 0x00000000 07DB76F 6459377F 776DB777 36EF6CAD 0x200001D3 0x00000010 F06DFCFC F359EC52 EBFC91E3 6FCB7B9E 0x200001D3 0x00000100 F06DFCFC F359EC52 EBFC91E3 6FCB7B9E 0x200001D3 0x00000100 BATA76AF BATA76AF BATA76AF BATA76AF BATA76AF Editor / Disassembly Breakpoints Memory Watches / Trace X
0x000000E0 07FD37F6 6459377F F76D8777 36EF6CAD 0x000000E0 07FD37F6 6459377F F76D8777 36EF6CAD 0x000000E0 F05DF7CF 7539D758 3F924576 3FD37656 0x0000010 F05DF7CF 5359652 EFC91E3 6FCB789E 0x0000010 F775660 6737ECC0 E6D5866E F27AFFA3 0x00000120 BAFA76AF B4CEF702 AFAAEB33 FFEED48B Editor / Disassembly / Breakpoints Memory / Watches / Trace / X Info & Errors X
0x000000F0 659A6738 7F9407F5 3F9ECE?7 3F2D3C56 0x00000100 CF047756B 6F37ECCD EGD586BE F2FAFFA3 0x00000101 C77786BD 6F37ECCD EGD586BE F2FAFFA3 0x00000101 DBAFA76AF B4CEFT02 AFAAEB33 FEED48B 1 Editor / Disassembly / Breakpoints / Memory / Watches / Trace /
0x00000100 FG0DPCFC F359BC52 EBFC9LE3 GFCB7B9E 0x00000100 C777B6BD GF37ECCD E605B6BE F2FAFFA3 0x00000120 BAXFA6AF BAXFA6AF BAXFA6AF BAXFA6AF Editor / Disasembly / Breakpoints Memory / Watches / Trace / Info & Errors - ×
0x00000101 C777566D 67372ECC E6D5866E F2FAFFA3 0x00000120 BAFA76AF B4FEPT6AF B4FEPT6AF B4FEPT6AF B4FEPT6AF 1 Editor / Disassembly / Breakpoints / Memory / Watches / Trace / X Info & Errors X
Correction Descention Descented Descention Descention Descention Descention Descention Descent
Image: Contract
Editor / Disassembly / Breakpoints / Memory / Watches / Trace / erminal - × Info & Errors - ×
erminal _ × Info & Errors ×
TAG HART link established using cable "DF-SoC HISE-11" device 2
INFO WALL THE CONTROL OF AN AND A CONTROL OF
INCOLOGY AND AND A
INTO: NOTIMENTY - ANERS_ILS_SILCE/ VALUES
INFO: Non-memory - INP 30 MPS and Office 000

Figure 20. The Memory window.

Memory						_ ×
Goto address	(hex or symb	ool name):		FF200040 G	Query Devices	Refresh Hide
	+0x0	+0x4	+0x8	+0xc		
0xFF200040	0000030F	00000000	00000000	00000000		
0xFF200050	00000000	00000000	00000000	00000000		
0xFF200060	FFFFFFFF	00000000	00000000	00000000		

Figure 21. Displaying the I/O locations.

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 address** panel, which is analogous to the **Goto instruction** panel discussed in Section 3.4. Note that in Figure 21 we reached the I/O device by typing the address FF200040 in this panel.

As an example of editing a memory value, go to address FF200000 which is the address of LEDs. Double-click on the memory word at this address and type the data value FFF. Press Enter on the computer keyboard, or click away from the memory word to apply the edit. This should cause all LEDs to be turned on.

When accessing an I/O device, some reads may be destructive. Namely, after some register in the I/O interface is read, its contents may no longer be valid. Therefore, it is not appropriate to read all I/O registers when refreshing

the information in the Memory window. Instead, it is prudent to read only the registers that are of specific interest. This can be accomplished by left-clicking on the address of interest, then right-clicking and then selecting **Read** Selected Address Range to update the displayed contents. Several consecutive addresses can be selected by clicking on the first address and dragging across the other addresses.

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. The Memory window provides additional features that are described in more detail in Appendix A of this tutorial.

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 *.amp*, 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: Displays the five most recently used project files, and allows these projects to be reopened.
- 4. Save Project: Saves the current project's settings after they have been modified by using the Settings command.

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 File > Edit Project > System Settings in the Monitor Program. 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 and modified by using the command File > Edit Project > Program Settings. To 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, or clicking the \checkmark icon.

5 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, called *Monitor_Terminal*. When creating the project, follow the same steps shown for the *Monitor_Tutorial* project, which were described in Section 3.1. For the screen shown in Figure 7 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 13 is called

JTAG_UART.s. It communicates using memory-mapped I/O with the JTAG UART in the DE1-SoC Computer that is selected as the **Terminal device** in the screen of Figure 9.

Compile, load and run the program. The Monitor Program window should appear as shown in Figure 22. 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.

🥔 Intel FPGA Monitor Program	- JTAG_UART : JTAG	G_UART.srec [F	Running]								- 🗆	×
<u>File Edit Actions Window</u>	vs <u>H</u> elp											
♦ 8 ♦ 8 ♦ ± 1	L 🤉 🕨 🖬 🖡	6 N 🖑										
Project Files _ ×	Disassembly								_ ×	Registers		_ ×
Monitor_Tutorial	Goto instruction	Address (her) or symbol name	e.			Go		Hide	Reg	Value	
JTAG_UART.s	doto instruction	riddiress (nex	.,				20	L		pc	0x0000000	
									-	r0	OXFFFFFFF	
										r1	0xFFFF1390	
			********	*******	*******	*********	*******	**********	***	r2	0x0000001	
				.te	(t				/*	r3	OXFFFEFCOC	
				.gl	obal -	start				r4	OXFFFF8COC	
			_start:							15	0xFFFF4AE4	
				/* :	set up s	tack point	ter */			10	0x00000076	
				MOV	:	SP, #DDR_B	END - 3	// highes	tm	17	Overerese 28	
			_start:							20	0x00000005	
	0x00000000	E3E0D10F	mvn	sp, #-10	73741821	; 0xc0	0000003		200	r10	0xFFD02000	
									222	r11	0xFFD02000	
				/* 1	print a (text strir	ng */			r12	0x00000010	
				LDR	1	R4, =TEXT	STRING			sp	0xFFFF8BF0	
	0x00000004	E59F4080	ldr	r4, [pc,	#128]	; 8c <te< td=""><td>EXT_STRI</td><td>NG+0x1c></td><td></td><td>lr</td><td>0xFFFF135E</td><td></td></te<>	EXT_STRI	NG+0x1c>		lr	0xFFFF135E	
										cpsr	0x200001D3	
			LOOP:									
				LDRI	3	RO, [R4]						
			LOOP:									
	80000000x0	E5D40000	ldrb	r0, [r4]								
				CMP	1	RO, #O			-			
	1	RAFAGAGA							•	1		
	Editor Disasse	mbly Break	points / Memory	/ Watche	s / Trace	/				-		
J			·	<u> </u>								
Terminal				_ ×	Info & Er	rors						_ ×
JTAG UART link establishe	ed using cable	"DE-SoC [US	B-1]", device	2,	INFO: N	on-memory	- ARM A	9 HPS dcan0	0xffc	00000		-
instance 0x02	-				TNFO: N	on-memory	- ARM A	9 HPS dcan1	0xffc	01000		
					INFO: N	on-memory	- ARM Z	9 HPS 13regs	0xff	800000		
JTAG UART example code					INFO: N	on-memory	- ARM Z	9 HPS adret1	Oxff	c25000		
>					TNFO: N	on-memory	- ARM A	9 HPS timer	0xfff	ec600		
					INFO: N	on-memory	- ARM Z	9 HPS scu 0x	fffec	000		2224
												-
					4		3	88///A				•
					Info & F		Server /	Debug / Variab	es /			
<u> </u>					millo ce t		Jeiver	variab				

Figure 22. Using 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 1. In this table <ESC> represents the ASCII character with the code $0 \times 1B$.

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>[#₁;#₂f</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>[1J</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 1. VT100 commands supported by the Terminal window.

In addition to the JTAG_UART, there exists another option for the terminal device. In Figure 9, in the Terminal device dropdown menu there is also a Semihosting option that is useful when C programs are used, as explained in the next section.

6 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 called *Monitor_Terminal_C*. 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 Figure 23, this program includes a C source file named *JTAG_UART.c*; it has the same functionality as the assembly-language code used in the previous example. Compile and run the program to observe its behavior.

First source file is used to o	letermine the name of the binary program	n file.
C:/Desktop/Monitor_Tutor C:/Desktop/Monitor_Tutor C:/Desktop/Monitor_Tutor C:/Desktop/Monitor_Tutor	ial/main.c ial/JTAG_UART.c ial/JTAG_UART.h ial/address_map_arm.h	Add Remove
		Up Down
rogram options Additional compiler flags: Additional linker flags:	-g -01	

Figure 23. Source files for a C program.

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 *printf* and *scanf*. In this case it is necessary to use the *Semihosting* terminal option, which can be selected in the window shown in Figure 9. Instead of the JTAG_UART_for_ARM_0, choose Semihosting in the dropdown menu for the Terminal device. Semihosting is a mechanism by which a program running on an ARM processor can request services from the debugger (Monitor Program). When an ARM program is compiled by the Monitor Program, special C libraries are used which have

been modified to use the Semihosting mechanism. All C library functions that communicate with a terminal, such as *printf* and *scanf*, will send/receive text to/from the Monitor Program's Semihosting terminal. In effect, Semihosting allows the host computer to provide input and output facilities that a system implemented on a DE1-SoC board does not have. A sample program, called *Semihosting Example*, is available when specifying C as the program type in Figure 7.

6.1 Source Level Debugging

The Monitor program supports common source level debugging features such as step over, step into, step out, and visualizing variables. Using the JTAG UART sample program project you created in the previous section, go to the project settings (File > Edit Project) and navigate to the *Program Settings* tab. In the *Compiler Flags* input box, ensure that the optimization level is set to 0, by replacing -*O*, -*O1*, -*O2*, or -*O3* flag with -*O0*. An optimization level of 0 allows the Monitor Program to read and display variables from memory. Figure 24 shows the Monitor Program's text editor. The editor will be disabled during the debug session, and re-enabled when the debug session is exited. Now save the project (File > Save Project), and compile and load the program (Actions > Compile & load).



Figure 24. The Monitor Program with a source file open in editor view.

6.1.1 Using Breakpoints

Once the program is loaded, navigate to the Editor window of the Monitor Program. Go to the File menu and select File > Open... to open the C source file which contains the *main* function of you program (most likely *main.c*).

Once the program is loaded, toggle the breakpoint at a line of source code by clicking on the numbers to the left of the source code text. If a breakpoint does not show up on the line similar to Figure 25, the line of source code likely does not correspond to an instruction. If this happens, try choosing a different line.



Figure 25. Setting a breakpoint in the editor view.

Once the breakpoint is set, continue the program by clicking the green arrow on the toolbar, or Actions > Continue. Once the program halts, the Monitor Program should look similar to Figure 26. In the Disassembly view the source level breakpoint is marked with a red square as in Figure 27. This differentiates source level breakpoints from instruction level breakpoints.

A Intel FPGA Monitor Program - Monitor_Terminal : main.srec [Paused]			– 🗆 X
Eile Edit Actions Windows Help			
🔅 📾 🛎 🤌 🕹 🥼 👁 🕪 🗉 🗞 💉 🖉			
Project Files _ X Disassembly		_ × Regi	iters _×
Goto instruction Address (hex) or syn	nbol name:	Go Hide	Reg Value
main.c	ata a2 (a11 8 9)	pc	0x000002C8
	ldr r3 [r11 #-8]	T	0x00000020
0x00000224 F5532000	1dada = 10, [122, 10]	**	0x00000020
0x00000289 E3530000	com x2 #0		0x00000020
0x000002BC 15555000	has 204 main-0v20-	74	0x00000088
0x000002BC 1x77774	hile 294 shalles 0x36.	75	0x00000850
0x000002C0 Ex000000	B 206 SHEERSOKSUS	16	OXFEFFFFFF
	anite (1)	7	OxFFFFF014
	1	18	0xFFFF5E28
	c = gec_jcag ();	r9	0x00000005
	IE (0 := '\0')	r10	0xFFD02000
	put_jtag (c);	r11	0x3FFFFFE4
)	r12	0x3FFFFFCC
0x000002C4 E1A00000	nop : (nov r0, r0)	ap	0x3FFFFFB8
	put_jtag (*str);	lr	0x000002A4
		oper	0x600001D3
	/* read and echo characters */		
	while (1)		
	{		
	<pre>c = get_jtag ();</pre>		
0x000002C8 E1200070	bl 348 <get_jtag></get_jtag>		
Editor) Disassembly / Breakpoints	/ Memory / Watches / Trace /		
Terminal	- × Info & Errors		- ×
JIRG UARI TINK ESTADIISHED USING CAble "DE-SoC [USB-1]"	, device 2, INFO: Non-memory	<pre>/ - ARM_A9_HPS_dcan1 0xffc01000</pre>	-
instance 0x02	INFO: Non-memory	/ - ARM_A9_HPS_13regs 0xff80000)
	INFO: Non-memory	/ - ARM_A9_HPS_sdrct1 0xffc2500	
JIAG UARI example code	INFO: Non-memory	/ - ARM_A9_HPS_timer 0xfffec600	
>	INFO: Non-memory	/ - ARM_A9_HPS_scu 0xfffec000	
	BREAK: Program b	reak 0 0x000002C8	800 B
		3111111	

Figure 26. Hitting a breakpoint in the editor view.

6.1.2 Source Level Debugging Actions

Navigate back the editor view and perform a Step Into action by selecting Actions > Step Into, or by using the main toolbar. This will step to the next line of source code to be executed. If the program steps into a function in another file, the Monitor Program will open the file in a new tab and highlight the line.



Figure 27. Source level breakpoint in the disassembly view.



Figure 28. Step Over, Step Into, Step Out toolbar icons

Next, perform a Step Out action by selecting Actions > Step Out, or by using the main toolbar. This will step out of the current function by executing until the first line of source code after returning from the current function. The Monitor Program will print an error to the Info & Errors window if it cannot step out of the current function. This may occur if the program is currently in the *main* function, or if the function does not return. The step out function is only available for C programs, it is not available for assembly programs.

The Step Over action (Actions > Step Over) moves to the next line of source code without stepping into functions. Execution will continue to the next line of source code inside the current function.

6.1.3 Variable Values

Variables		_ ×
Name	Туре	Value
Statics		
Locals		
c	unsigned char	٥
str	unsigned char *	0x3FFFFFD6
⊡text_string	unsigned char[28]	JTAG UART exampl
Info & Errors / GDB Server	Debug Variables	

Figure 29. Monitor Program Variable View.

The Monitor Program's Variables view displays the value of C program variables when the program is halted. Some variable types such as Arrays, Typedefs, Structures and Unions will be expandable in the view. Use the + button to expand and view the variables contents. Right clicking on a variable presents the options to jump to the declaration of the variable, and the display format of the variable.

Go To Declaration will open the file the variable is declared in and scroll to the declaration line number. *Display As...* will change the format in which the variable is displayed.

Variable values are only available with an optimization level of $\mathbf{0}$ (gcc command line argument -OO). For instructions on how to change the programs optimization level, see the first paragraph of this section.

6.1.4 Enabling and Disabling Source Level Debugging

The source level debugging feature of the Monitor Program is a beta feature in the current release. The feature can be enabled and disabled at any point by going to the Edit menu and selecting Edit > Enable Source Level Debugging, or Edit > Disable Source Level Debugging, depending on whether the feature is currently disabled or enabled respectively.

6.1.5 Setting the Optimization Level in Programs with Driver Support.

To set the optimization level for a *Program with Driver Support* (or BSP), first create a TCL script in the base directory of the project (the same directory as your AMP project file). The TCL file should have a *.tcl* file extension, for example *config.tcl*. Open this file in a text editor and add the single line:

set_setting hal.make.bsp_cflags_optimization -O0

Where the argument -*OO* above is the desired optimization level. Now open the project settings in the Monitor Program and navigate to the *Program Settings* tab. In the *BSP settings TCL script* input box (shown in Figure 30) enter the path to the TCL script you just created, or use the *Browse* button to search for it.

Click the *Finish* button to close the dialog and save and compile the project. The optimization level should be set for both the generated (BSP) files, as well as your project files.

7 Using the Monitor Program with Interrupts

The Monitor Program supports the use of exceptions and interrupts in programs. In an ARM-based system, interrupt requests are handled by the Generic Interrupt Controller. Consult the tutorial *Using the ARM Generic Interrupt Controller*, which is available at the Intel website, for a description of how interrupts are processed.

7.1 Interrupts with Assembly-Language Programs

To see an example using interrupts with assembly-language code, create a new Monitor Program project. When creating the new project choose the ARM processor, set the program type to assembly language and select the sample program named *Interrupt Example*. Figure 31 lists the source files for this example. The main program is in the file

e Settings System Settings Program Type Program Settings Connection	s Settings
pecify program details	i Settings 1
/	
First source file is used to determine the name of the binary program file.	
C:/tempy/TESTFOLDER/MEDIA_HAL/media_HAL.c	Add
	Remove
	Up
	Down
Program options	
BSP settings Tcl script (optional):	
	Browse

Figure 30. Adding a TCL script to a Program with Driver Support.

interrupt_example.s. Comments given in this file explain the behavior of the program.



Figure 31. The assembly-language source files for the ARM interrupt example.

To enable the proper handling of exceptions and interrupts it is necessary to specify the memory settings as required by the ARM processor. Figure 10 indicates the memory settings if exceptions and interrupts are not used. Figure 32 shows the required memory settings for this example. The memory locations in the address range 0×0 to $0 \times 3F$ must be reserved for vectors used in various exceptions and interrupts. The main program can start at address 0×40 .

Complete the project, download the program, run it, and observe its behavior when you press KEY1, KEY2 or KEY3.

7.2 Interrupts with C Programs

To see an example using interrupts with C code, create a new Monitor Program project. When creating the new project choose the ARM processor, set the program type to C language and select the sample program named *Inter*-

Specify program memory settings

Here you can specify see the linker to place code by the linker, the names	ction names and their start and e and data at the specified address must match those identified by	nd addresses. These sections will be used by ses. To ensure correct use of the section names the assembler directives, such as .text.
inker Section Presets	Exceptions	
linker seedon resets.	exceptions	
Section Name	Memory Device	Address Range
Section Name	Memory Device	Address Range 0x00000000 – 0x0000003F
Section Name .vectors .text	Memory Device ARM_DDR3_SDRAM ARM_DDR3_SDRAM	Address Range 0x00000000 – 0x0000003F 0x00000040 – 0x3FFFFFF7

Figure 32. Memory Settings for the ARM interrupt example.

rupt Example. Figure 33 lists the source files for this example. The main program is in the file *interrupt_example.c*. Comments given in this file explain the behavior of the program.

c	Course files						
3	Source mes						
	First source file is used to determine the name of the binary program file.						
	D:/Monitor_Interrupts_C/interrupt_example.c						
	D:/Monitor_Interrupts_C/exceptions.c						
	D:/Monitor_Interrupts_C/HPS_timer_ISR.c						
	D:/Monitor_Interrupts_C/interval_timer_ISR.c						
	D:/Monitor_Interrupts_C/JTAG_UART.c						
	D:/Monitor_Interrupts_C/pushbutton_ISR.c						

Figure 33. The C-language source files for the ARM interrupt example.

Complete the project, download the program, run it, and observe its behavior when you press KEY1, KEY2 or KEY3.

8 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 Intel FPGA Monitor Program with a rearranged workspace.

9 Appendix A

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

9.1 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*, as follows:

- Read watchpoint the processor is halted when a read operation is performed on a specific address.
- Write watchpoint the processor is halted when a write operation is performed on a specific address.
- Access watchpoint the processor is halted when a read or write operation is performed on a specific address.

🥔 Intel FPGA Monitor Program - Monitor_Tutorial : simple_program.srec [Paused] — 🛛 🛛 🖉							
File Edit Actions Windows Help							
🗇 🗟 🕭 🚵 👌 👃	1 🗇 🕨 🗉 📞 🕺 🗳						
Project Files $_$ \times	Breakpoints	_ ×	Registers		_ ×		
Monitor_Tutorial	Instruction breakpoint: Address Instruction Condition Ox0000010b		Reg pc r0 r1 r2 r4 r5 r6 r7 r8 r9 r10 r11 r12 sp lr cpsr	Value (x0000000 0xFFFFFF200000 0xFFF70000 0xFFF7000 0xFFFF000 0xFFFF000 0xFFFF000 0xFFFF000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF000000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF00000 0xFFF0000 0xFF00000 0xFFF0000 0xFF00000 0xFFF0000 0xFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFFF0000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFFF0000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFF00000 0xFFF0000 0xFF00000 0xFF00000 0xFF00000 0xFF0000 0xFF0000 0xFF00000 0xFF00000 0xFF00000 0xFF0000 0xF0000 0xFF0000 0xFF0000 0xFF0000 0xFF0000 0xFF0000 0xFF0000 0xF0000 0xFF0000 0xF000			
Terminal		Info & Errors			- ×		
JTAG UART link establish	ed using cable "DE-SoC [USB-1]", device 2,	INFO: Non-memory - ARM_A9_HFS_dcan1 0xffc INFO: Non-memory - ARM_A9_HFS_l3regs 0xff INFO: Non-memory - ARM_A9_HFS_lstrefs 0xff INFO: Non-memory - ARM_A9_HFS_stuner 0xfff INFO: Non-memory - ARM_A9_HFS_scu 0xfffce Program stopped 0 0x0000008	01000 800000 c25000 ec600 000				

Figure 35. The Breakpoints window.

In Figure 35 an instruction breakpoint is shown for the address 0×0000010 . This corresponds to an address in *simple_program.s.* 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 in the *Run until* section by double-clicking on 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 instructionbreakpoint, or it can be a stand-alone condition if entered in the Run until box in the Breakpoints window. As an example, we compiled and loaded the *simple_program* project. Then, we entered the condition R3 == 5. The condition causes the breakpoint to trigger only if register R3 contains the value 5. Thus, running this program causes the LEDs to display the current state of the slider switches as these switches are set to different patterns. But, when the selected pattern is 0x005, the conditional breakpoint will stop the execution of the program.

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.



Figure 36. The Conditional Breakpoint dialog.

9.2 Working with the Memory Window

The Memory window was shown in Figure 20. This window is configurable in a variety of ways:

- 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 Section 3.7, 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.

	+0x0	+0x4 +0x8 +0xc		-
0x00000000	E59F1	Read Selected Address Range		
0x00000010	EAFFF	<u>N</u> eda Sciectea Address Range		
0x00000020	00000	View as	•	Byte (1-byte)
0x00000030	55555	<u>_icu us</u>		byte (1 byte)
0x00000040	EA000	Number of words per line		Half-word (2-bytes)
0x00000050	E2466			
0x00000060	E3A01	N <u>u</u> mber format		 Word (4-bytes)
0x00000070	E15E0	Display order	• • ["]	1
0x0000080	000A0			
0x00000090	ESACO	Cwitch to character mode		
0x000000A0	lafff	Switch to character mode		
0x000000B0	000A0			
0x000000C0	lafff	Snow equivalent ASCII characters		
0x000000D0	lafff			
0x000000E0	E28F1	Goto <u>m</u> emory address		
0x000000F0	BA000	Memory fill		
0x00000100	E3500	Memory Int		
0x00000110	E3500	Load file into memory		
0x00000120	00000	.00 00000CAU EJITPOOD EATTIC		1

Figure 37. Setting the memory element size.

	+0;	x0 +0x4 +0x8 +0xc	_	
00000000x000x0	E5	Read Selected Address Range	1	
0x00000010	EÀ			
0x00000020	00	View as		
x00000030	55	Tien do	-	
0x00000040	EA	Number of words per line	1	
0x00000050	E2	Number formet	2	
0x00000060	E3	Number format		
x00000070	El	Display order	• 4	
0x00000080	AO			.2
0x00000090	E8	Switch to character mode	<u>8</u>	~~
0x000000A0	1A	Switch to character mode	16	
0x000000B0	AO	Show aquivalent ASCII characters	10	
0x000000C0	1A	Show equivalent ASCII characters	32	
0x000000D0	1A	Coto ano and data a	-	
0x000000E0	E2	Goto memory address	64	
0x000000F0	BA	Memory fill		
0x00000100	E3	includy them	<u>A</u> uto	
x00000110	E3	Load file into memory		_
1x00000120	00	UUULSU UUUUULAS ESIFDUUU EAFFFFL4		

Figure 38. Setting the number of words per line.

9.2.1 Character Display

The Memory window can also be configured to interpret memory byte values as ASCII characters. This is useful if one wishes to examine character strings that are stored in the memory. For this purpose it is convenient to view the memory in bytes and characters simultaneously so that the characters appear in the correct sequence. This can be accomplished by clicking the Switch to character mode menu item, as illustrated in Figure 39. A sample display in the character mode is shown in Figure 40.



Figure 39. Switching to the character mode.

Memory _ ×						
Goto address	Soto address (hex or symbol name): 00020000 Go 🗌 Query Devices Refresh Hide					
	+0x0	+0x4	+0x8 +0xc	+0x0	▲	
0x00020000	61 62 63 64	65 66 67 68	69 6A 6B 6C 6D 6E	abcd efgh	ijkl mnop	
0x00020010	71 72 73 74	75 76 77 78	79 7A 7B 7C 31 32	qrst uvwx	yz (1234	
0x00020020	FF 6B B3 DF	7F F8 97 F7	D7 4D EF 70 F1 B9	0 k 0 0 0 9 0	0 M 0 p 0 0 0	
0x00020030	DB F9 DF 38	9A F5 D3 E8	F9 FF 79 6D D3 D6	0008 V00	ооут ооок	
0x00020040	B1 FD FE EF	BF D5 F6 A1	2E B5 DA E3 E0 FE		. o o ⁻ ¢ o o -	
0x00020050	37 BB 3F FB	E3 8B 6E ED	BB 8A 2B 77 7E 25	7020 tn•	□/\+ឃ~%□~	
0x00020060	5D E9 F7 F6	95 El EE 3F	DF E9 63 66 DF 6C]+□□ l£°?	0 + c f 0 1 = 0	
0x00020070	A4 B2 48 FF	69 B8 B0 FB	CE 89 6B EF FA 4D	00K0 1000	ojko om^o	
0x00020080	FC BE FD FF	DA 7F 74 DD	5D B3 ED 9E F7 1E	#000 0 t0] 🛛 🖷 🕺 🖉 👘	
0x00020090	76 5A C3 29	74 DB A5 6F	13 75 F5 EF 79 E7	v20) t00o	u00 y000	
0x000200A0	F9 F6 DF 8F	7B BE D6 EF	7C FF 7D 5F 1C 5F	0007 {000	10}00	
0x000200B0	F7 FF EB CB	D7 BF DF D5	2D 65 DA DE 8F 62		-eOO マbJO	
0x000200C0	A2 83 3E AF	Al 35 DA 6D	AF D3 A5 9E 57 F7	0 f > 0 0 5 0 m		
0x000200D0	3E B1 1A 7E	D5 C2 FD 71	56 56 FE 5F B4 E7	p000 ~ 0<	vvo_ 000+	
0x000200E0	6F B7 FD 07	7F B7 59 64	77 F7 6D F7 BD 6C	o 🛛 🖉 🖉 🖉 🖉 D 🖓 D	w0m0 0106	
0x000200F0	3A 67 DA 66	FA D7 94 7F	77 CE 9E 3F 56 B4	:gOf 00†	wo"? Voo	
0x00020100	FE FD 6D FO	52 FC 59 F3	E3 81 FC EB 9E 7F	OOmO R#YO	_ _{₹ ₩:} , × , o ▼	
•			300000			
Editor / Disa	Editor / Disassembly / Breakpoints / Memory / Watches / Trace /					

Figure 40. Character mode display.

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

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

9.2.3 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.

Memory							_ ×
Goto address (hex	Goto address (hex or symbol name): 0 Go Query Devices Refresh Hide						Hide
Load file	Hide		+0x0 E59F100C	+0x4 E59F200C	+0x8 E5923000	+0xc E5813000	
Select a file:	<u>B</u> rowse	0x00000010	EAFFFFFC	FF200000	FF200040	00000000	
		0x00000030	55555555	55555555	55555555	55555555	
File type:		0x00000040 0x00000050	E49E6023 F59F9039	0B000014 ED910043	E5830520 E3102912	E12FFF9A 4A000103	
Start address (hex):	head	0x00000060 0x00000070	F2A481FF C55E4A8A	EAC00014 5355A067	E3A08047 65205456	E12FFF1F 716D657A	333
	Load	0x00000080 0x00000090	6369657C FF601012	0E65646F FFFF5204	0420213E C4241100	08900075 10000250	
		0x000000A0	000DD698	004490BA	46400664	00844000	
		0x000000000000000000000000000000000000	AE3E83A2	6D5A2531	9EA1D7FF	904A028F 8587F757	
		0x000000D0 0x000000E0	7E1AB13C 07FDB76F	11FDE2D5 6459B77F	5EFE4656 F76DB777	EBADE7F4 26FF6CBD	-
•							
Editor (Disassembly (Breakpoints) Memory (Watches (Trace)							

Figure 41. The Load file panel.

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 hexadecimal number with an optional - sign. 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).

```
00,11,22,33
1044,2055,3066,4077
10000088,20000099,300000aa,400000bb
1,-1,2,-2
```

Figure 42. A Delimited hexadecimal value file.

9.3 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 and click Add, as illustrated in Figure 43.

Watches _ ×					
Add	Expression	Value			



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.



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.

9.4 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 HPS system on the DE1-SoC board. 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.

9.5 Running an ARM* Program from an SD Card

After developing an ARM program, you may wish execute it as a standalone application on the FPGA board without intervention from the Monitor Program. This can be accomplished by using the Actions > Generate U-Boot SD Card Binaries operation of the Monitor Program. This operation generates files that can be placed into a special microSD* card configured with the *U-Boot* loader. When the board is powered on with the SD card inserted, the ARM processor automatically runs the *U-Boot* loader, programs the FPGA (if applicable), loads the program into memory, and executes the program.

The *U-Boot* SD card images for supported DE-series boards are provided with the Monitor Program installation in the directory /<*installation path*>/*University_Program*/SD_*Images*/. Unzip the file corresponding to your board and use a tool such as *Win32DiskImager* to load the *.img* file into a microSD card (2GB or larger). Note that this action

will remove any existing data in the SD card. Once the SD card has been loaded with the image, it will contain the following files:

- program.bin
- *setup_environment.bin*
- fpga.rbf
- set_vbar.bin

The files *program.bin*, *setup_environment.bin*, and *fpga.rbf* are generated by the Actions > Generate U-Boot SD Card Binaries operation, and together they represent a Monitor Program project. The file *program.bin* is the compiled ARM program, *setup_environment.bin* is a file that contains project configuration details, and *fpga.rbf* is the FPGA programming file used by the project. All of these files except *fpga.rbf* are mandatory for successful execution (*fpga.rbf* may be omitted if the project does not use the FPGA). The default files included in the SD card image correspond to the *Getting Started* sample program bundled with the Monitor Program. The file *set_vbar.bin* is a program that is part of the boot sequence that you must not modify nor delete.

To place your own ARM application into the SD card, generate *program.bin*, *setup_environment.bin*, and *fpga.rbf* for your project by selecting Actions > Generate U-Boot SD Card Binaries in the Monitor Program. Copy over the files to the SD card and replace any prior copies.

10 Appendix B - Configuration File

The Monitor Program configuration file allows default values to be set for project creation. The monitor program searches \$(UniversityProgramRoot)/amp.config for the configuration file, where UniversityProgramRoot is the path to the University Program directory in the Quartus installation. For example C:/intelFPGA/16.1/University_Program/amp.config.

To change the default path to the configuration file, add the following command line argument when running the Monitor Program: --config-file=<Path to File>

Table 2 summarizes the configuration options available in the Monitor Program.

The configuration file uses white space or an equal sign as a delimiter, for example: flag option or flag=option. Where *flag* is one of the values in the first column of Table 2 and *option* is the default value for that flag. Number signs (#) can be used to add comments to the configuration file. Lines starting with the symbol will not be processed with the configuration file. Boolean values can use integers or case insensitive strings. Options of 'false', 'no' and '0' will all produce a false Boolean, any other values will produce a true Boolean.

Flag	Explanation
project_name	The project name.
project_path	The new project directory path.
architecture	The architecture.
system	The default sample system to be used (ex. DE1-SoC Computer)
c_compiler_flags	C Compiler flags
c_linker_flags	C Linker flags
use_small_c_lib	Boolean to use the small C Library (Nios II)
emulate_instr	Boolean to emulate unimplemented instructions
include_system_info_file	Boolean whether to include the system info header by default.
answer_for_reload_file	yes or no option to bypass the file reload dialog when files are edited outside the
	program. If undefined, the dialog will be shown.

Table 2. Configuration Flags and Default Options.

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Avalon, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel 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 Intel. Intel 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.

*Other names and brands may be claimed as the property of others.