

For Quartus II 13.0

1 Introduction

This tutorial presents an introduction to Altera's Qsys system integration tool, which is used to design digital hardware systems that contain components such as processors, memories, input/output interfaces, timers, and the like. The Qsys tool allows a designer to choose the components that are desired in the system by selecting these components in a graphical user interface. It then automatically generates the hardware system that connects all of the components together.

Introduction to the Altera

Qsys System Integration Tool

The hardware system development flow is illustrated by giving step-by-step instructions for using the Qsys tool in conjuction with the Quartus[®] II software to implement a simple example system. The last step in the development process involves configuring the designed hardware system in an actual FPGA device, and running an application program. To show how this is done, it is assumed that the user has access to an Altera DE-series Development and Education board connected to a computer that has Quartus II and Nios[®] II software installed. The screen captures in the tutorial were obtained using the Quartus II version 13.0; if other versions of the software are used, some of the images may be slightly different.

Contents:

- Nios II System
- Altera's Qsys Tool
- Integration of a Nios II System into a Quartus II Project
- Compiling a Quartus II Project when using the Qsys Tool
- Using the Altera Monitor Program to Download a Designed Hardware System and Run an Application Program

2 Altera DE-series FPGA Boards

For this tutorial we assume that the reader has access to an Altera DE-series board, such as the one shown in Figure 1. The figure depicts the DE2-115 board, which features an Altera Cyclone IV FPGA chip. The board provides a lot of other resources, such as memory chips, slider switches, pushbutton keys, LEDs, audio input/output, video input (NTSC/PAL decoder) and video output (VGA). It also provides several types of serial input/output connections, including a USB port for connecting the board to a personal computer. In this tutorial we will make use of only a few of the resources: the FPGA chip, slider switches, LEDs, and the USB port that connects to a computer.

Although we have chosen the DE2-115 board as an example, the tutorial is pertinent for other DE-series boards that are described in the University Program section of Altera's website.

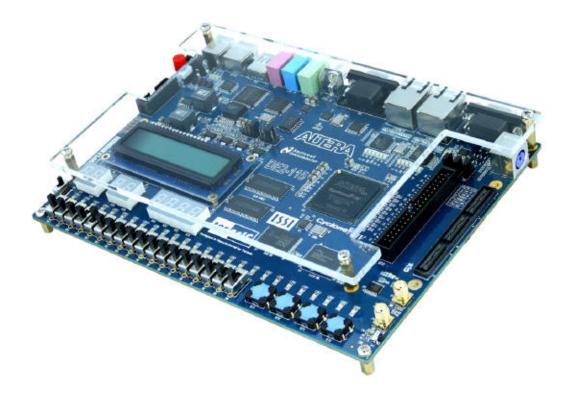


Figure 1. An Altera DE2-115 board.

3 A Digital Hardware System Example

We will use a simple hardware system that is shown in Figure 2. It includes the Altera Nios[®] II embedded processor, which is a *soft processor* module defined as code in a hardware-description language. A Nios II module can be included as part of a larger system, and then that system can be implemented in an Altera FPGA chip by using the Quartus II software.

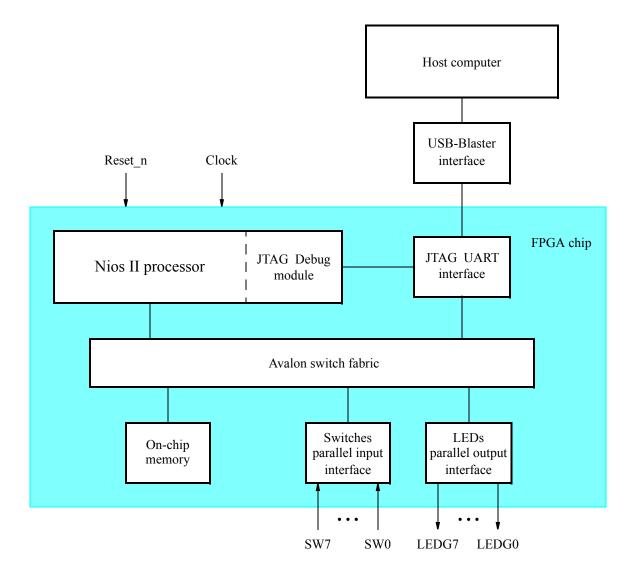


Figure 2. A simple example of a Nios II system.

As shown in Figure 2, the Nios II processor is connected to the memory and I/O interfaces by means of an interconnection network called the *Avalon switch fabric*. This interconnection network is automatically generated by the Qsys tool.

The memory component in our system will be realized by using the on-chip memory available in the FPGA chip. The I/O interfaces that connect to the slider switches and LEDs will be implemented by using the predefined modules that are available in the Qsys tool. A special JTAG UART interface is used to connect to the circuitry that provides a USB link to the host computer to which the DE-series board is connected. This circuitry and the associated software is called the *USB-Blaster*. Another module, called the JTAG Debug module, is provided to allow the host computer to control the Nios II system. It makes it possible to perform operations such as downloading Nios II programs into memory, starting and stopping the execution of these programs, setting breakpoints, and examining the contents of

memory and Nios II registers.

Since all parts of the Nios II system implemented on the FPGA chip are defined by using a hardware description language, a knowledgeable user could write such code to implement any part of the system. This would be an onerous and time consuming task. Instead, we will show how to use the Qsys tool to implement the desired system simply by choosing the required components and specifying the parameters needed to make each component fit the overall requirements of the system. Although in this tutorial we illustrate the capability of the Qsys tool by designing a very simple system, the same approach is used to design larger systems.

Our example system in Figure 2 is intended to realize a trivial task. Eight slider switches on the DE2-115 board, SW7-0, are used to turn on or off the eight green LEDs, LEDG7-0. To achieve the desired operation, the eight-bit pattern corresponding to the state of the switches has to be sent to the output port to activate the LEDs. This will be done by having the Nios II processor execute a program stored in the on-chip memory. Continuous operation is required, such that as the switches are toggled the lights change accordingly.

In the next section we will use the Qsys tool to design the hardware depicted in Figure 2. After assigning the FPGA pins to realize the connections between the parallel interfaces and the switches and LEDs on the DE2-115 board, we will compile the designed system. Finally, we will use the software tool called the *Altera Monitor Program* to download the designed circuit into the FPGA device, and download and execute a Nios II program that performs the desired task.

Doing this tutorial, the reader will learn about:

- Using the Qsys tool to design a Nios II-based system
- Integrating the designed Nios II system into a Quartus II project
- Implementing the designed system on the DE2-115 board
- Running an application program on the Nios II processor

4 Altera's Qsys Tool

The Qsys tool is used in conjuction with the Quartus II CAD software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters. To implement the system in Figure 2, we have to instantiate the following functional units:

- Nios II processor
- On-chip memory, which consists of the memory blocks in the FPGA chip; we will specify a 4-Kbyte memory arranged in 32-bit words
- Two parallel I/O interfaces
- JTAG UART interface for communication with the host computer

To define the desired system, start the Quartus II software and perform the following steps:

1. Create a new Quartus II project for your system. As shown in Figure 3, we stored our project in a directory called *qsys_tutorial*, and we assigned the name *lights* to both the project and its top-level design entity. You can choose a different directory or project name. Step through the screen for adding design files to the project; we will add the required files later in the tutorial. In your project, choose the FPGA device used on your DE-series board. A list of FPGA devices on the DE-series boards is given in Table 1.

New Project Wizard	×
Directory, Name, Top-Level Entity [page 1 of 5]	
What is the working directory for this project?	
D:\qsys_tutorial	
What is the name of this project?	
lights	
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.	
lights	
Use Existing Project Settings	
< Back Next > Finish Cancel H	lelp
	p

Figure 3. Create a new project.

Board	Device Name
DE0	Cyclone III EP3C16F484C6
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-70	Cyclone II EP2C70F896C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1. DE-series FPGA device names

2. After completing the New Project Wizard to create the project, in the main Quartus II window select Tools >

Qsys, which leads to the window in Figure 4. This is the System Contents tab of the Qsys tool, which is used to add components to the system and configure the selected components to meet the design requirements. The available components are listed on the left side of the window.

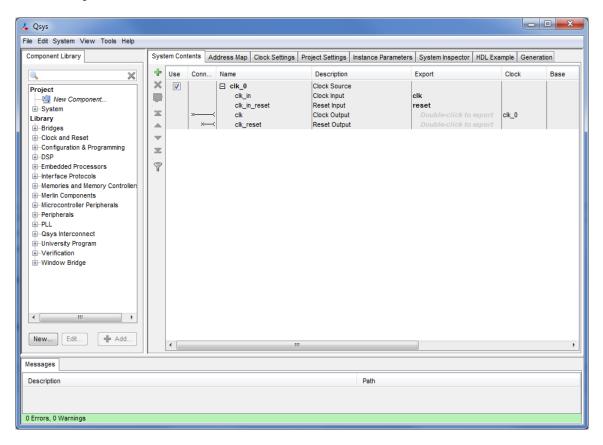


Figure 4. Create a new Nios II system.

3. The hardware system that will be generated using the Qsys tool runs under the control of a clock. For this tutorial we will make use of the 50-MHz clock that is provided on the DE2-115 board. In Figure 4 click on the Clock Settings tab (near the top of the screen) to bring this tab to the foreground, as illustrated in Figure 5. Here, it is possible to specify the names and frequency of clock signals used in the project. If not already included in this tab, specify a clock named clk_0 with the source designated as External and the frequency set to 50.0 MHz. The settings are made by clicking in each of the three columns: Name, Source and MHz.

Return to the System Contents tab.

🙏 Qsys File Edit System View Tools Help							
Component Library	t system View Tools Help nent Lbriny System Contents Address Mag Clock Settings Project Settings Instance Parameters System Inspector HDL Example Generation Clock Settings Anne Source MHz Add Remove HHz Add Remove						
	m View Tools Help Trary Image: System Contents Address Map Clock Settings Instance Parameters System Contents Address Map Clock Settings Component Image: Sustem Contents Address Map Clock Settings Image: Sustem Contents Address Map Add Seset Image: Sustem Contents Address Map External 56.0 Image: Sustem Contents Address Map Add Protocols Image: Sustem Controller mononis Image: Sustem Controller Protocols Image: Sustem Controller Protocols </td <td></td>						
Project	Name		Source		MHz	1	Add
New Component System Library Bridges Clock and Reset Configuration & Programming OSP Embedded Processors Interface Protocols Memories and Memory Controller: Merini Components Microcontroller Peripherals PLL Osys Interconnect University Program Verification Verification Verification Verification Verification	clk_0	3	ixternal		50.0		Remove
Messages	·						
Description				Path			
0 Errors, 0 Warnings							

Figure 5. The Clock Settings tab.

- 4. Next, specify the processor as follows:
 - On the left side of the Qsys window expand Embedded Processors, select Nios II Processor and click Add, which leads to the window in Figure 6.

	ory Interfaces Advanced Featur	res MMU and MPU Settings JTAG De	ebug Module
Select a Nios II Core			
Nios II Core:	Nios We		
	Nios IVs		
	Nios I/f		
	Nios II/e	Nios II/s	Nios II/f
Nios II Selector Guide	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction
Memory Usage (e.g Stratix	IV) Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache
Hardware Arithmetic Ope	aration		
Hardware multiplication type:			
Hardware divide			
Reset Vector			
Reset Vector	None	•	
Reset vector memory:	0x0000000		
Reset vector memory: Reset vector offset:	0×00000000 0×00000000		
Reset vector memory: Reset vector offset:			
Reset vector memory: Reset vector offset: Reset vector: Exception Vector			
Reset vector memory: Reset vector offset: Reset vector:	0x0000000	•	
Reset vector memory: Reset vector offset: Reset vector: Exception Vector Exception vector memory:	0x00000000	•	

Figure 6. Create a Nios II processor.

Choose Nios II/e which is the economy version of the processor. This version is available for use without
a paid license. The Nios II processor has *reset* and *interrupt* inputs. When one of these inputs is activated,
the processor starts executing the instructions stored at memory addresses known as *reset vector* and *interrupt vector*, respectively. Since we have not yet included any memory components in our design,
the Qsys tool will display corresponding error messages. Ignore these messages as we will provide the
necessary information later. Click Finish to return to the main Qsys window, which now shows the Nios
II processor specified as indicated in Figure 7.

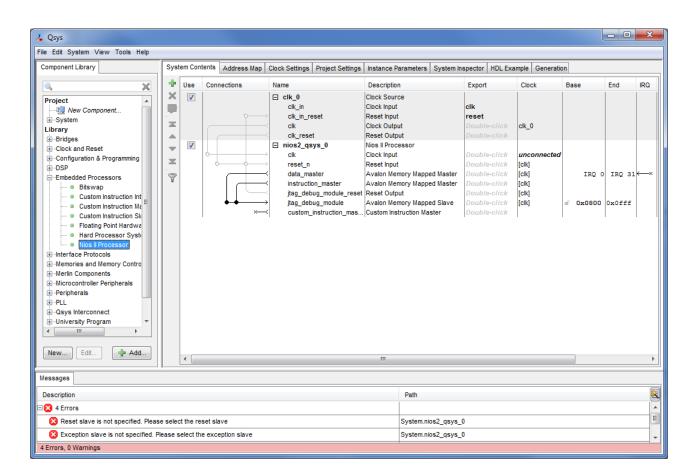


Figure 7. Inclusion of the Nios II processor in the design.

- 5. To specify the on-chip memory perform the following:
 - Expand the category Memories and Memory Controllers, and then expand to select On-Chip > On-Chip Memory (RAM or ROM), and click Add
 - In the On-Chip Memory Configuration Wizard window, shown in Figure 8, ensure that the Data width is set to 32 bits and the Total memory size to 4K bytes (4096 bytes)
 - Do not change the other default settings
 - Click Finish, which returns to the System Contents tab as indicated in Figure 9

Mogeccore On-Chip Me attera_avalon_onchi	nory (RAM or ROM)	Documentation
Block Diagram Show signals onchip_memory2_0 ckl1 clock s1 avalon reset reset	Memory type Type: RAM (Writable) Dual-port access Single clock operation Read During Write Mode: DONT_CARE Block type:	
altera_avalon_onchip_memory2		E
	Slave s2 Latency: 1 - Memory initialization Initialize memory content Enable non-default initialization file Type the filename (e.g: my_ram.hex) or select the hex file using the file brows User created initialization file: <u>onchip_mem.hex</u> Enable In-System Memory Content Editor feature	ser button.
	Instance ID: NONE Memory will be initialized from unsaved_onchip_memory2_0.hex	

Figure 8. Define the on-chip memory.

6. Observe that while the Nios II processor and the on-chip memory have been included in the design, no connections between these components have been established. To specify the desired connections, examine the Connections area in the window in Figure 9. The connections already made are indicated by filled circles and the other possible connections by empty circles, as indicated in Figure 10.

Clicking on an empty circle makes a connection. Clicking on a filled circle removes the connection.

Make the following connections:

- Clock inputs of the processor and the memory to the clock output of the clock component
- Reset inputs of the processor and the memory to both the reset output of the clock component and the *jtag_debug_module_reset* output
- The s1 input of the memory to both the data_master and instruction_master outputs of the processor

The resulting connections are shown in Figure 11.

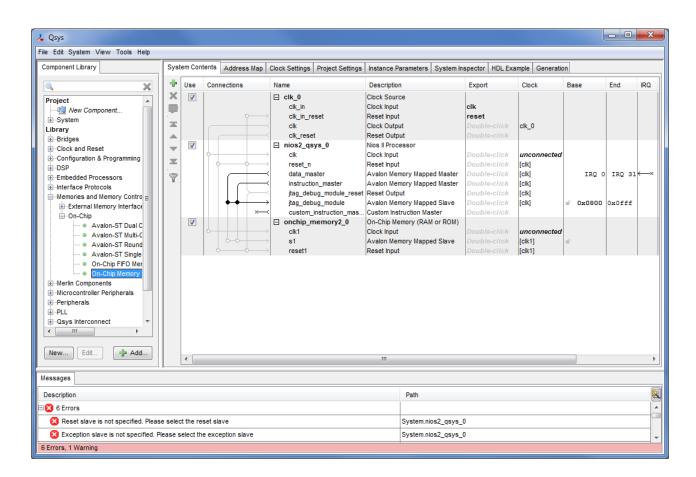


Figure 9. The on-chip memory included on a DE-series board.

Connections	Name	Description	Export
	⊟ clk_0	Clock Source	
	clk_in	Clock Input	clk
¢	→ clk_in_reset	Reset Input	reset
	clk	Clock Output	Double-clic
	clk_reset	Reset Output	Double-clic
	nios2_qsys_0	Nios II Processor	
¢	\rightarrow clk	Clock Input	Double-clic
¢	→ reset_n	Reset Input	Double-clic
	≺ data_master	Avalon Memory Mapped Ma	ster Double-clic
	→ instruction_master	Avalon Memory Mapped Ma	ster Double-clic
$ \rightarrow$	jtag_debug_module_	re Reset Output	Double-clic
	→ jtag_debug_module	Avalon Memory Mapped Sla	ve Double-clic
×-	— custom_instruction_	m Custom Instruction Master	Double-clic
	onchip_memory2_	0 On-Chip Memory (RAM or R	(MO)
<u> </u>	\rightarrow clk1	Clock Input	Double-clic
0-0	→ s1	Avalon Memory Mapped Sla	ve Double-clic
	→ reset1	Reset Input	Double-clic

Figure 10. Connections that can be made.

Connections	Name	Description	Export
	⊟ clk_0	Clock Source	
	clk_in	Clock Input	clk
φ—	→ clk_in_reset	Reset Input	reset
	clk	Clock Output	Double-clic
	clk_reset	Reset Output	Double-clic
	□ nios2_qsys_0	Nios II Processor	
↓	→ clk	Clock Input	Double-clic
++	→ reset_n	Reset Input	Double-clic
	data_master	Avalon Memory Mapped Ma	ster Double-clic
		Avaion Memory Mapped Ma	ster Double-clic
$ \succ$		re Reset Output	Double-clic
• •	→ jtag debug module		ave Double-clic
×		m Custom Instruction Master	Double-clic
	onchip_memory2	-	ROM)
↓ ↓ ↓ ↓ ↓	\rightarrow clk1	Clock Input	Double-clic
	→ s1	Avalon Memory Mapped Sl	ave Double-clic
	→ reset1	Reset Input	Double-clic

Figure 11. The connections that are now established.

- 7. Specify the input parallel I/O interface as follows:
 - Select Peripherals > Microcontroller Peripherals > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard in Figure 12
 - Specify the width of the port to be 8 bits and choose the direction of the port to be Input, as shown in the figure.
 - Click Finish.

PIO (Parallel I/O) - pio_0 PIO (Parallel I/O) Megedeer Altera_avalon_pio	
Block Diagram Show signals clk elook reset est external_connection conduit altera_avalon_pio	Basic Settings Width (1-32 bits): Bidir Bidir Inout Output Output Port Reset Value: 0 output Output Register Enable individual bit setting/clearing ✓ Edge capture register Synchronously capture Edge Type: RISING ♥ Enable bit-clearing for edge capture register ✓ Interrupt Generate IRQ IRQ Type: LEVEL ♥ Level: Interrupt CPU when any unmasked VD pin is logic true Edge: riser us logic true. Available when synchronous capture is enabled ✓ Test bench wiring Hardwire PIO inputs in test bench Drive inputs to:
Info: pio_0: PIO inputs are not hardwired in test bench. Undefined value	alues will be read from PIO inputs during simulation.
	Cancel Finish

Figure 12. Define a parallel input interface.

- 8. In the same way, specify the output parallel I/O interface:
 - Select Peripherals > Microcontroller Peripherals > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard again
 - Specify the width of the port to be 8 bits and choose the direction of the port to be Output.
 - Click Finish to return to the System Contents tab
- 9. Specify the necessary connections for the two PIOs:
 - Clock input of the PIO to the clock output of the clock component
 - Reset input of the PIO to the reset output of the clock component and the *jtag_debug_module_reset* output
 - The *s1* input of the PIO the *data_master* output of the processor

The resulting design is depicted in Figure 13.

	ntents Address Map Clo	ock Settings Project Settings	Instance Parameters System In		xample Ge	neration	
Use	Connections	Name	Description	Export	Clock	Base	End IR
V		⊟ clk_0	Clock Source				
		clk_in	Clock Input	clk			
		clk_in_reset	Reset Input	reset			
		< clk	Clock Output	Double-click	clk_0		
		< clk_reset	Reset Output	Double-click			
1		nios2_qsys_0	Nios II Processor				
	+	Clk	Clock Input	Double-click	clk_0		
		→ reset_n	Reset Input	Double-click	[clk]		
		data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ	0 IRQ 31
		instruction_master	Avalon Memory Mapped Master		[clk]		
		jtag_debug_module_re.		Double-click	[clk]		
		jtag_debug_module		Double-click	[Clk]		0 0x0fff
	×	custom_instruction_m		Double-click			
1		onchip_memory2_0	On-Chip Memory (RAM or ROM)				
	♦ →	clk1	Clock Input	Double-click	clk_0		
		s1	Avalon Memory Mapped Slave	Double-click	[clk1]		0 0x0fff
		reset1	Reset Input	Double-click	[clk1]		
1		⊟ pio_0	PIO (Parallel VO)				
	↑ →	+ clk	Clock Input	Double-click	clk_0		
		reset	Reset Input	Double-click	[clk]		
		• s1	Avalon Memory Mapped Slave	Double-click	[clk]		0 0x000f
_		external_connection	Conduit Endpoint	Double-click			
1		⊡ pio_1	PIO (Parallel VO)				
	• • • • • • • • • • • • • • • • • • • •	→ clk	Clock Input	Double-click	clk_0		
		reset	Reset Input	Double-click	[Clk]		
		• s1	Avalon Memory Mapped Slave	Double-click	[clk]		10 0x000f
	· · ·	external_connection	Conduit Endpoint	Double-click			

Figure 13. The system with all components and connections.

- 10. We wish to connect to a host computer and provide a means for communication between the Nios II system and the host computer. This can be accomplished by instantiating the JTAG UART interface as follows:
 - Select Interface Protocols > Serial > JTAG UART and click Add to reach the JTAG UART Configuration Wizard in Figure 14
 - Do not change the default settings
 - Click Finish to return to the System Contents tab

Connect the JTAG UART to the clock, reset and data-master ports, as was done for the PIOs. Connect the Interrupt Request (IRQ) line from the JTAG UART to the Nios II processor by selecting the connection under the IRQ column, as shown in Figure 15. Once the connection is made, a box with the number 0 inside will appear on the connection. The Nios II processor has 32 interrupt ports ranging from 0 to 31, and the number in this box selects which port will be used for this IRQ. Click on the box and change it to use port 5.

👃 JTAG UART - jtag_uart_0		×
JTAG UART attera_avalon_itag_uart		Documentation
Block Diagram Show signals jtag_uart_0 clk elock interupt reset avalon_jtag_slave avalon attera_avalon_jtag_uart	Write FIFO (Data from Avalon to JTAG) Buffer depth (bytes): 64 IRQ threshold: 8 Construct using registers instead of memory blocks Pread FIFO (Data from JTAG to Avalon) Buffer depth (bytes): 64 RQ threshold: 8 Construct using registers instead of memory blocks Allow multiple connections Allow multiple connections to Avalon JTAG slave	
	Ca	Incel Finish

Figure 14. Define the JTAG UART interface.

syste	em Cont	tents	Address Map 0	Clock Settings	Project Settings	Instance Parameters	System Inspe	ctor HDL E	xample	Generatio	n		
+	Use	Conn	ections	Name		Description	Ex	port	Clock	B	ase	End	IRC
×	V			□ clk_0		Clock Source							
2				clk_in	ı	Clock Input	clk						
			¢	→ clk_in	_reset	Reset Input	res	set					
Z		-		clk		Clock Output	Do	uble-click	clk_0				
▲					eset	Reset Output	Do	uble-click					
-	-			🗆 nios2	_qsys_0	Nios II Processor							
-		♦ -		→ clk		Clock Input	Do	uble-click	clk_0				
×		🛉	+	→ reset	_n	Reset Input	Do	uble-click	[clk]				
7			-	-≺ data_	master	Avalon Memory Map	ped Master Do	uble-click	[clk]		IRQ 0	IRQ 31	.←
Ш				instru	ction_master	Avalon Memory Map	ped Master Do	uble-click	[clk]				
			≻—	≺ jtag_¢	debug_module_re.	Reset Output	Do	uble-click	[clk]				
			+ + -	→ jtag_o	debug_module	Avalon Memory Map	ped Slave Do	uble-click	[clk]		0x0800	0x0fff	
			×		m_instruction_m	Custom Instruction M	laster Do	uble-click					
	V			🗆 onchij	_memory2_0	On-Chip Memory (RA	AM or ROM)						
		♦ -	+ + +	→ clk1		Clock Input	Do	uble-click	clk_0				
			++	→ s1		Avalon Memory Map	ped Slave Do	uble-click	[clk1]		0x000x0	0x0fff	
		🛉	+ + +	→ reset	1	Reset Input	Do	uble-click	[clk1]				
	1			🗆 pio_0		PIO (Parallel I/O)							
		+-		→ clk		Clock Input	Do	uble-click	clk_0				
		🛉	+ + +	→ reset	1	Reset Input	Do	uble-click	[clk]				
			+ +	→ s1		Avalon Memory Map	ped Slave Do	uble-click	[clk]		0x000x0	0x000f	
				exter	nal_connection	Conduit Endpoint	Do	uble-click					
	v			🗆 pio_1		PIO (Parallel I/O)							
		+-+		→ clk		Clock Input	Do	uble-click	clk_0				
		🛉	+ +	→ reset	:	Reset Input	Do	uble-click	[clk]				
			• •	→ s1		Avalon Memory Map	ped Slave Do	uble-click	[clk]		0x0000	0x000f	
				- exter	nal_connection	Conduit Endpoint	Do	uble-click					
	\checkmark			😑 jtag_u	art_0	JTAG UART							
		♦ -		→ clk		Clock Input		uble-click	clk_0				
		♦	+ + •	→ reset	:	Reset Input	Do	uble-click	[clk]				
			÷	→ avalo	n_jtag_slave	Avalon Memory Map	ped Slave Do	uble-click	[clk]		0x0000	0x0007	×

Figure 15. Connect the IRQ line from the JTAG UART to the Nios II processor.

11. Note that the Qsys tool automatically chooses names for the various components. The names are not necessarily descriptive enough to be easily associated with the target design, but they can be changed. In Figure 2, we use the names Switches and LEDs for the parallel input and output interfaces, respectively. These names can be used in the implemented system. Right-click on the pio_0 name and then select Rename. Change the name to *switches*. Similarly, change pio_1 to *LEDs*. Figure 16 shows the system with name changes that we made for all components.

Use	Connections	Name	Description	Export	Clock	Base	End	
\checkmark		⊟ clk_0	Clock Source					
		clk_in	Clock Input	clk				
	¢	clk_in_reset	Reset Input	reset				
		≺ <mark>clk</mark>	Clock Output	Double-click	clk_0			
		<	Reset Output	Double-click				
1		☐ nios2_qsys_0	Nios II Processor					
	•	→ clk	Clock Input	Double-click	clk_0			
	++	→ reset_n	Reset Input	Double-click	[clk]			
		✓ data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ	0 IRQ 3	1
		✓ instruction_master	Avalon Memory Mapped Master	Double-click	[clk]			
	≻	→ jtag_debug_module_reset	Reset Output	Double-click	[clk]			
		→ jtag_debug_module	Avalon Memory Mapped Slave	Double-click	[clk]		0 0x0fff	
	×	custom_instruction_mas	Custom Instruction Master	Double-click				
V		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
	♦ 	→ clk1	Clock Input	Double-click	clk_0			
		→ s1	Avalon Memory Mapped Slave	Double-click	[clk1]	0x000 €	0 0x0fff	
		→ reset1	Reset Input	Double-click	[clk1]			
1		switches	PIO (Parallel VO)					Т
		→ clk	Clock Input	Double-click	clk_0			
•		→ reset	Reset Input	Double-click	[clk]			
		→ s1	Avalon Memory Mapped Slave	Double-click	[clk]	e 0x000	0 0x000f	
		external_connection	Conduit	Double-click				
V		LEDs	PIO (Parallel VO)					
	♦	→ clk	Clock Input	Double-click	clk_0			
		→ reset	Reset Input	Double-click	[clk]			
	♦ ↔	→ s1	Avalon Memory Mapped Slave	Double-click	[clk]		0 0x000f	
		external_connection	Conduit	Double-click				
1		⊡ jtag uart 0	JTAG UART					Т
		→ clk	Clock Input	Double-click	clk_0			
	│	→ reset	Reset Input	Double-click	[clk]			
	L .	→ avalon_itag_slave	Avalon Memory Mapped Slave	Double-click	[clk]	0x000	0 0x0007	6

Figure 16. The system with all components appropriately named.

12. Observe that the base and end addresses of the various components in the designed system have not been properly assigned. These addresses can be assigned by the user, but they can also be assigned automatically by the Qsys tool. We will choose the latter possibility. However, we want to make sure that the on-chip memory has the base address of zero. Double-click on the Base address for the on-chip memory in the Qsys window and enter the address 0x00000000. Then, lock this address by clicking on the adjacent lock symbol. Now, let Qsys assign the rest of the addresses by selecting System > Assign Base Addresses (at the top of the window), which produces an assignment similar to that shown in Figure 17.

Use	Connections	Name	Description	Export	Clock	Base	End	
V		⊟ clk_0	Clock Source					
		clk_in	Clock Input	clk				Ι
	Ŷ	clk_in_reset	Reset Input	reset				
		≺ clk	Clock Output	Double-click	clk_0			
		<	Reset Output	Double-click				
1		nios2_qsys_0	Nios II Processor					
	•	→ clk	Clock Input	Double-click	clk_0			
	+ - +	→ reset_n	Reset Input	Double-click	[clk]			
		✓ data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ 0	IRQ 31	١k
		instruction_master	Avalon Memory Mapped Master	Double-click	[clk]			
	≻──	jtag_debug_module_reset	Reset Output	Double-click	[clk]			
	+ +	→ jtag_debug_module	Avalon Memory Mapped Slave	Double-click	[clk]	0x1800	0x1fff	
	×	 custom_instruction_mas 	Custom Instruction Master	Double-click				
V		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
	♦	→ clk1	Clock Input	Double-click	clk_0			
	• •	→ s1	Avaion Memory Mapped Slave	Double-click	[clk1]		0x0fff	
	• •	→ reset1	Reset Input	Double-click	[clk1]			
V		switches	PIO (Parallel VO)					Ι
	♦	→ clk	Clock Input	Double-click	clk_0			
	• • •	→ reset	Reset Input	Double-click	[clk]			
		→ s1	Avalon Memory Mapped Slave	Double-click	[clk]		0x200f	
	-	external_connection	Conduit	Double-click				
V		E LEDs	PIO (Parallel VO)					
	♦	→ clk	Clock Input	Double-click	clk_0			
		→ reset	Reset Input	Double-click	[clk]			
		→ s1	Avalon Memory Mapped Slave	Double-click	[clk]		0x201f	
		external_connection	Conduit	Double-click				
v		⊟ jtag_uart_0	JTAG UART					I
		→ clk	Clock Input	Double-click	clk_0			
	• •	→ reset	Reset Input	Double-click	[clk]			
	↓	→ avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click	[clk]		0x2027	6

Figure 17. The system with assigned addresses.

- 13. The behavior of the Nios II processor when it is reset is defined by its reset vector. It is the location in the memory device from which the processor fetches the next instruction when it is reset. Similarly, the exception vector is the memory address of the instruction that the processor executes when an interrupt is raised. To specify these two parameters, perform the following:
 - Right-click on the *nios2_processor* component in the window displayed in Figure 17, and then select Edit to reach the window in Figure 18
 - Select *onchip_memory* to be the memory device for both reset and exception vectors, as shown in Figure 18
 - Do not change the default settings for offsets
 - Observe that the error messages dealing with memory assignments shown in Figure 6 will now disappear
 - Click Finish to return to the System Contents tab

ore Nios I Caches and Memory	Interfaces Advanced Features	MMU and MPU Settings JTAG D	ebug Module	
Select a Nios II Core				
Nios II Core:	Nios IVe			
	Nios IVs			
	Nios I/f			
	Nios II/e	Nios II/s	Nios II/f	1
Nios II Selector Guide	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction	
Memory Usage (e.g Stratix IV)	Two M9Ks (or equiv.)	Two M9Ks + cache	Three M9Ks + cache	
Hardware Arithmetic Opera	tion			
Hardware multiplication type:	Embedded Multipliers			
Hardware divide				
Reset Vector				
Reset vector memory:	onchip_memory2_0.s1	▼		
Reset vector offset: Reset vector:	0×0000000			
Reset vector:	0x0000000			
Exception Vector				
Exception vector memory:	onchip_memory2_0.s1	▼		
Exception vector offset:	0x0000020			
Exception vector:				

Figure 18. Define the reset and exception vectors.

14. So far, we have specified all connections inside our *nios_system* circuit. It is also necessary to specify connections to external components, which are switches and LEDs in our case. To accomplish this, double click on Double-click (in the Export column of the System Contents tab) for external_connection of the switches PIO, and type the name *switches*. Similarly, establish the external connection for the lights, called *leds*. This completes the specification of our *nios_system*, which is depicted in Figure 19.

Use	Connections	Name	Description	Export	Clock	Base	End	IR
\checkmark		⊟ clk_0	Clock Source					
		clk_in	Clock Input	clk				
	¢	> clk_in_reset	Reset Input	reset				
		< clk	Clock Output	Double-click	clk_0			
		< clk_reset	Reset Output	Double-click				
1		nios2_qsys_0	Nios II Processor					
		→ clk	Clock Input	Double-click	clk_0			
		→ reset_n	Reset Input	Double-click	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click	[clk]	IRQ 0	IRQ 31	ı⊬
		instruction_master	Avalon Memory Mapped Master	Double-click	[clk]			
		itag_debug_module_reset	Reset Output	Double-click	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	Double-click	[clk]		0x1fff	
	×	custom_instruction_mas	Custom Instruction Master	Double-click				
V		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
		→ clk1	Clock Input	Double-click	clk_0			
		→ s1	Avalon Memory Mapped Slave	Double-click	[clk1]		0x0fff	
		→ reset1	Reset Input	Double-click	[clk1]			
1		switches	PIO (Parallel I/O)					
	♦ 	→ clk	Clock Input	Double-click	clk_0			
		→ reset	Reset Input	Double-click	[clk]			
		→ s1	Avalon Memory Mapped Slave	Double-click	[clk]		0x200f	
		external_connection	Conduit	switches				
V		🗆 LEDs	PIO (Parallel VO)					
		→ clk	Clock Input	Double-click	clk_0			
		→ reset	Reset Input	Double-click	[clk]			
		→ s1	Avaion Memory Mapped Slave	Double-click	[clk]		0x201f	
		external connection	Conduit	leds				
V			JTAG UART					Г
		→ clk	Clock Input	Double-click	clk 0			
		→ reset	Reset Input	Double-click	[clk]			
		> avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click	[clk]	0x2020	0x2027	6

Figure 19. The complete system.

15. Having specified all components needed to implement the desired system, it can now be generated. Save the specified system; we used the name *nios_system*. Then, select the Generation tab, which leads to the window in Figure 20. Select None for the options Simulation > Create simulation model and Testbench System > Create testbench Qsys system, because in this tutorial we will not deal with the simulation of hardware. Click Generate on the bottom of the window. When successfully completed, the generation process produces the message "Generate Completed".

Exit the Qsys tool to return to the main Quartus II window.

👃 Qsys - nios_system.qsys (D:\qsys_tutoria	I\nios_system.qsys)						
File Edit System View Tools Help							
Component Library Sys	stem Contents Address Map Clock S	ettings Project Settings Ins	tance Parameters	System Inspector	HDL Example	Generation	
	Simulation The simulation model contains generate Create simulation model: Testbench System The testbench system is a new Qsys s Once generated, the bus functional mo Create testbench Qsys system: Create testbench simulation model:	None	ginal system, addin	g bus functional mo		e top-level interfaces.	
Memories and Memory Controller Merin Components Microcontroller Peripherals Peripherals PLL Osys Interconnect University Program Verification	Synthesis Synthesis files are used to compile the Create HDL design files for synthesis: Create block symbol file (.bsf)						
. Window Bridge	lge Path: Simulation: Testbench:		D:/qsys_tutorial/nios_system				
Image: Synthesis: D:/qsys_tutorial/nios_system/synthesis/ New Edit Image: Box Synthesis Generate							
Messages							
Description			Path				
∃ (1) 1 Info Message							
ID PIO inputs are not hardwired in test ben	ch. Undefined values will be read from	PIO inputs during simulation.	System.pio_0				
0 Errors, 0 Warnings							

Figure 20. Generation of the system.

Changes to the designed system are easily made at any time by reopening the Qsys tool. Any component in the System Contents tab of the Qsys tool can be selected and edited or deleted, or a new component can be added and the system regenerated.

5 Integration of the Nios II System into a Quartus II Project

To complete the hardware design, we have to perform the following:

- Instantiate the module generated by the Qsys tool into the Quartus II project
- Assign the FPGA pins
- Compile the designed circuit
- Program and configure the FPGA device on the DE2-115 board

5.1 Instantiation of the Module Generated by the Qsys Tool

The Qsys tool generates a Verilog module that defines the desired Nios II system. In our design, this module will have been generated in the *nios_system.v* file, which can be found in the directory *qsys_tutorial/nios_system/synthesis* of the project. The Qsys tool generates Verilog modules, which can then be used in designs specified using either Verilog or VHDL languages.

Normally, the Nios II module generated by the Qsys tool is likely to be a part of a larger design. However, in the case of our simple example there is no other circuitry needed. All we need to do is instantiate the Nios II system in our top-level Verilog or VHDL module, and connect inputs and outputs of the parallel I/O ports, as well as the clock and reset inputs, to the appropriate pins on the FPGA device.

The Verilog code in the *nios_system.v* file is quite large. Figure 21 depicts the portion of the code that defines the input and output ports for the module *nios_system*. The 8-bit vector that is the input to the parallel port *switches* is called *switches_export*. The 8-bit output vector is called *leds_export*. The clock and reset signals are called *clk_clk* and *reset_reset_n*, respectively. Note that the reset signal was added automatically by the Qsys tool; it is called *reset_reset_n* because it is active low.

Figure 21. A part of the generated Verilog module.

The *nios_system* module has to be instantiated in a top-level module that has to be named *lights*, because this is the name we specified in Figure 3 for the top-level design entity in our Quartus II project. For the input and output ports of the *lights* module we have used the pin names that are specified in the DE2-115 User Manual: *CLOCK_50* for the 50-MHz clock, *KEY* for the pushbutton switches, *SW* for the slider switches, and *LEDG* for the green LEDs. Using these names simplifies the task of creating the needed pin assignments.

5.1.1 Instantiation in a Verilog Module

Figure 22 shows a top-level Verilog module that instantiates the Nios II system. If using Verilog for the tutorial, type this code into a file called *lights.v*, or use the file provided with this tutorial.

// Implements a simple Nios II system for the DE-series board. // Inputs: SW7-0 are parallel port inputs to the Nios II system CLOCK_50 is the system clock // // KEY0 is the active-low system reset // Outputs: LEDG7–0 are parallel port outputs from the Nios II system module lights (CLOCK_50, SW, KEY, LEDG); input CLOCK_50; **input** [7:0] SW; **input** [0:0] KEY; output [7:0] LEDG; // Instantiate the Nios II system module generated by the Qsys tool: nios_system NiosII (.clk_clk(CLOCK_50), .reset_reset_n(KEY), .switches_export(SW), .leds_export(LEDG)); endmodule

Figure 22. Instantiating the Nios II system using Verilog code.

5.1.2 Instantiation in a VHDL Module

Figure 23 shows a top-level VHDL module that instantiates the Nios II system. If using VHDL for the tutorial, type this code into a file called *lights.vhd*, or use the file provided with this tutorial.

-- Implements a simple Nios II system for the DE-series board. -- Inputs: SW7-0 are parallel port inputs to the Nios II system CLOCK_50 is the system clock KEY0 is the active-low system reset ___ -- Outputs: LEDG7-0 are parallel port outputs from the Nios II system LIBRARY ieee: USE ieee.std logic 1164.ALL; USE ieee.std_logic_unsigned.ALL; **ENTITY lights IS** PORT (CLOCK_50 : IN STD_LOGIC; KEY : IN STD_LOGIC_VECTOR (0 DOWNTO 0); SW : IN STD_LOGIC_VECTOR (7 DOWNTO 0); LEDG : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)); END lights; ARCHITECTURE lights_rtl OF lights IS COMPONENT nios_system PORT (SIGNAL clk_clk: IN STD_LOGIC; SIGNAL reset_reset_n : IN STD_LOGIC; SIGNAL switches_export : IN STD_LOGIC_VECTOR (7 DOWNTO 0); SIGNAL leds_export : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)); END COMPONENT; BEGIN NiosII : nios_system PORT MAP($clk_clk => CLOCK_50,$ $reset_reset_n => KEY(0),$ switches_export => SW(7 DOWNTO 0), $leds_export => LEDG(7 DOWNTO 0)$); END lights_rtl;

Figure 23. Instantiating the Nios II system using VHDL code.

6 Compiling the Quartus II Project

Add the *lights.v/vhd* file to your Quartus II project. Also, add the necessary pin assignments for the DE-series board to your project. The procedure for making pin assignments is described in the tutorial *Quartus II Introduction Using Verilog/VHDL Designs*. Note that an easy way of making the pin assignments when we use the same pin names as in the DE2-115 User Manual is to import the assignments from a Quartus II Setting File with Pin Assignments. For example, the pin assignments for the DE2-115 board are provided in the *DE2-115.qsf* file, which can be found on Altera's DE2-115 web pages.

Since the system we are designing needs to operate at a 50-MHz clock frequency, we can add the needed timing assignment in the Quartus II project. The tutorial *Using TimeQuest Timing Analyzer* shows how this is done. However, for our simple design, we can rely on the default timing assignment that the Quartus II compiler assumes in the absence of a specific specification. The compiler assumes that the circuit has to be able to operate at a clock frequency of 1 GHz, and will produce an implementation that either meets this requirement or comes as close to it as possible.

Finally, before compiling the project, it is necessary to add the *nios_system.qip* file (IP Variation file) in the directory *qsys_tutorial/nios_system/synthesis* to your Quartus II project. Then, compile the project. You may see some warning messages associated with the Nios II system, such as some signals being unused or having wrong bit-lengths of vectors; these warnings can be ignored.

7 Using the Altera Monitor Program to Download the Designed Circuit and Run an Application Program

The designed circuit has to be downloaded into the FPGA device on a DE-series board. This can be done by using the Programmer Tool in the Quartus II software. However, we will use a simpler approach by using the Altera Monitor Program, which provides a simple means for downloading the circuit into the FPGA as well as running the application programs.

A parallel I/O interface generated by the Qsys tool is accessible by means of registers in the interface. Depending on how the PIO is configured, there may be as many as four registers. One of these registers is called the Data register. In a PIO configured as an input interface, the data read from the Data register is the data currently present on the PIO input lines. In a PIO configured as an output interface, the data written (by the Nios II processor) into the Data register drives the PIO output lines. If a PIO is configured as a bidirectional interface, then the PIO inputs and outputs use the same physical lines. In this case there is a Data Direction register included, which determines the direction of the input/output transfer. In our unidirectional PIOs, it is only necessary to have the Data register. The addresses assigned by the Qsys tool are 0x00002000 for the Data register in the PIO called *switches* and 0x00002010 for the Data register in the PIO called *LEDs*, as indicated in Figure 17.

Our application task is very simple. A pattern selected by the current setting of slider switches has to be displayed on the LEDs. We will show how this can be done in both Nios II assembly language and C programming language.

7.1 A Nios II Assembly Language Program

Figure 23 gives a Nios II assembly-language program that implements our task. The program loads the addresses of the Data registers in the two PIOs into processor registers r_2 and r_3 . It then has an infinite loop that merely transfers the data from the input PIO, *switches*, to the output PIO, *leds*.

.equ	switche	s, 0x00002000
.equ	leds, 0x	00002010
.global	_start	
_start:	movia	r2, switches
	movia	r3, leds
LOOP:	ldbio	r4, 0(r2)
	stbio	r4, 0(r3)
	br	LOOP
.end		

Figure 24. Assembly-language code to control the lights.

The directive .global _start indicates to the Assembler that the label _*start* is accessible outside the assembled object file. This label is the default label we use to indicate to the Linker program the beginning of the application program.

For a detailed explanation of the Nios II assembly language instructions see the tutorial *Introduction to the Altera Nios II Soft Processor*, which is available on Altera's University Program website.

Enter this code into a file *lights.s*, or use the file provided with this tutorial, and place the file into a working directory. We placed the file into the directory *qsys_tutorial\app_software*.

7.2 A C-Language Program

An application program written in the C language can be handled in the same way as the assembly-language program. A C program that implements our simple task is given in Figure 24. Enter this code into a file called *lights.c*, or use the file provided with this tutorial, and place the file into a working directory.

```
#define switches (volatile char *) 0x0002000
#define leds (char *) 0x0002010
void main()
{     while (1)
         *leds = *switches;
}
```

Figure 25. C-language code to control the lights.

7.3 Using the Altera Monitor Program

The Altera University Program provides the *monitor* software, called *Altera Monitor Program*, for use with the DEseries boards. This software provides a simple means for compiling, assembling and downloading of programs onto a DE-series board. It also makes it possible for the user to perform debugging tasks. A description of this software is available in the *Altera Monitor Program* tutorial. We should also note that other Nios II development systems are provided by Altera, for use in commercial development. Although we will use the Altera Monitor Program in this tutorial, the other Nios II tools available from Altera could alternatively be used with our designed hardware system.

Open the Altera Monitor Program, which leads to the window in Figure 26.

Altera Monitor Program [Nios II]		
<u>F</u> ile <u>S</u> ettings <u>A</u> ctions <u>W</u> indows <u>H</u> elp		
山田 副本語 今日日 さや		
Disassembly	_ ×	Registers _ ×
Goto instruction Address (hex) or symbol name:	Go Hide	Reg Value
Disassembly / Breakpoints / Memory / Watches / Trace /		
Terminal _ ×	Info & Errors	_ ×

Figure 26. The Altera Monitor Program main window.

The monitor program needs to know the characteristics of the designed Nios II system, which are given in the file *nios_system.qsys*. Click the File > New Project menu item to display the New Project Wizard window, shown in Figure 27, and perform the following steps:

- 1. Enter the *qsys_tutorial\app_software* directory as the Project directory by typing it directly into the Project directory field, or by browsing to it using the Browse... button.
- 2. Enter *lights_example* (or some other name) as the Project name and click Next, leading to Figure 28.

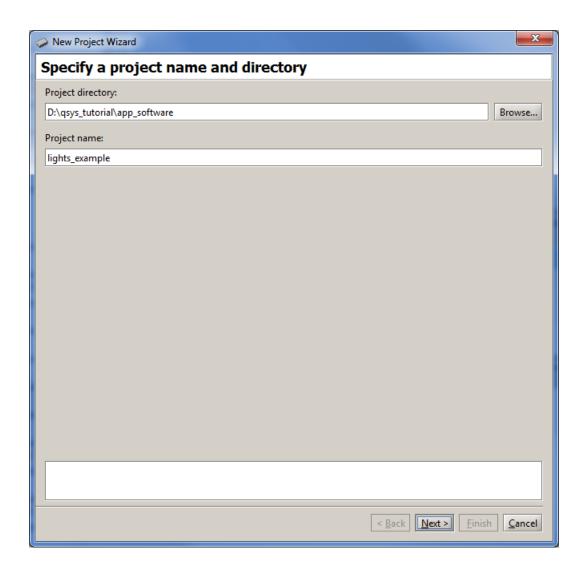


Figure 27. Specify the project directory and name.

New Project Wizard	x
Specify a system	
Select a system	
<custom system=""></custom>	n
Specify a Nios II system by selecting a system description (PTF, Qsys) file, and optional Quartus II programming (SOF) and Quartus II JTAG debugging information (JDI) files.	
System details	
System description file (PTF, Qsys or SOPCInfo):	
D:\qsys_tutorial\nios_system.qsys Browse	
Quartus II programming (SOF) file (optional):	
D:\qsys_tutorial\output_files\lights.sof Browse	
The SOF file represents the FPGA programming file for the Nios II system. If it is specified here, then the Monitor Program can be used to download this programming file onto the board. Otherwise, the system wil need to be downloaded using some other method (for example, by using Quartus II).	
Quartus II JTAG debugging information (JDI) file (optional):	
Browse	
The JDI file is required for multiprocessor systems designed in Qsys. It stores the JTAG Device IDs. These IDs are needed for communication between the Monitor Program and the system's multiple processors and JTA UARTs .	G
< <u>B</u> ack <u>Next</u> > <u>Finish</u> <u>C</u> a	ancel

Figure 28. The System Specification window.

3. From the Select a System drop-down box select Custom System, which specifies that you wish to use the hardware that you designed.

Click Browse... beside the System description field to display a file selection window and choose the *nios_system.qsys* file. Note that this file is in the design directory *qsys_tutorial*.

Select the *lights.sof* file in the Quartus II programming (SOF) file field, which provides the information needed to download the designed system into the FPGA device on the DE-series board. Click Next, which leads to the window in Figure 29.

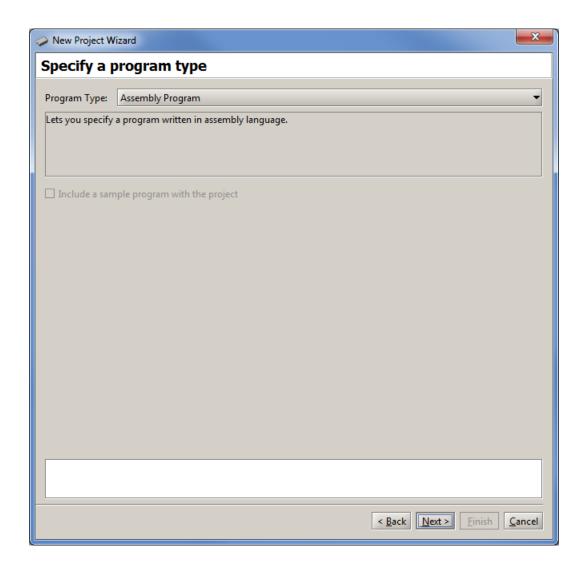


Figure 29. Specification of the program type.

- 4. If you wish to use a Nios II assembly-language application program, select Assembly Program as the program type from the drop-down menu. If you wish to use a C-language program, select C Program. Click Next, leading to Figure 30.
- 5. Click Add... to display a file selection window and choose the *lights.s* file, or *lights.c* for a C program, and click Select. We placed the application-software files in the directory *qsys_tutorial\app_software*. Upon returning to the window in Figure 30, click Next.

New Project Wizard	×
Specify program details	
_Source files	
First source file is used to determine the name of the binary program file.	
D:\qsys_tutorial\app_software\lights.s	Add Remove
	Up Down
Program options	
Start symbol:start	
< Back Next >	Einish <u>C</u> ancel

Figure 30. Specify the application program to use.

6. In the window in Figure 31, ensure that the Host Connection is set to *USB-Blaster*, the Processor is set to *nios2_processor* and the Terminal Device is set to *jtag_uart*. Click Next.

New Project Wiza	rd 📃 🔀
Specify syste	em parameters
_System paramete	rs
Host connection:	: USB-Blaster [USB-0]
Processor:	nios2_qsys_0
	Reset vector address: 0x0000000
	Exception vector address: 0x00000020
Terminal device:	jtag_uart_0
	< <u>B</u> ack <u>Next</u> > <u>Finish</u> <u>Cancel</u>

Figure 31. Specify the system parameters.

7. The Monitor Program also needs to know where to load the application program. In our case, this is the memory block in the FPGA device. The name assigned to this memory is *onchip_memory*. Since there is no other memory in our design, the Monitor Program will select this memory by default, as shown in Figure 32.

Having provided the necessary information, click Finish to confirm the system configuration. When a pop-up box asks you if you want to have your system downloaded onto the DE-series board click Yes.

🧼 New Project Wizard		x
Specify program me	emory settings	
Processor's reset and except	ion vectors (read-only)	
Reset vector address:	0x0000000	
Exception vector address:	0x00000020	
Memory options		
These addresses can be in the ensure that the .text and .dat	rting addresses of sections identified by .text and .data assembler directives. the same or in different memories (on-chip, SDRAM,). They can be used to ta sections do not overlap with other sections, such as .reset and .exceptions. If to have the same address, the .data section will be placed right after the .text	
text section		-
Memory device:	onchip_memory2_0/s1 (0x0 - 0xfff)	-
Start offset in device (hex):		0
.data section		
Memory device:	onchip_memory2_0/s1 (0x0 - 0xfff)	-
Start offset in device (hex):		0
	< <u>B</u> ack <u>N</u> ext > <u>F</u> inish <u>C</u>	ancel

Figure 32. Specify where the program will be loaded in the memory.

- 8. Now, in the monitor window in Figure 26 select Actions > Compile & Load to assemble (compile in the case of a C program) and download your program.
- 9. The downloaded program is shown in Figure 33. Run the program and verify the correctnes of the designed system by setting the slider switches to a few different patterns.

Altera Monitor Program [Nios II] - lights_example.ncf : lights.srec [Paused]							x
<u>F</u> ile <u>S</u> ettings	<u>A</u> ctions <u>W</u> i	ndows <u>H</u> elp					
👱 🗈 🔒 🚸	a 🔊 🔊	🕨 🗉 🕼 🖉 👘					
Disassembly				_ ×	Registers	_	×
Goto instruction	Address (hex)) or symbol name: 5c	Go	Hide	Reg	Value	
					pc	0x00000000	
				-	zero	0x0000000	
		.equ switches, 0x00002010			r1	0x0000000	
		.equ leds, 0x00002000			r2	0x00000000	
		.global _start			r3	0x00000000	
		_start: movia r2, switches			r4	0x00000000	- 193
		_start:			r5	0x00000000	- 100
0x00000000	00800034	orhi r2, zero, 0x0			r6	0x00000000	
0x00000004	10880404	addi r2, r2, 0x2010			r7	0x00000000	
		movia r3, leds		333	r8	0x00000000	
0x0000008	00c00034	orhi r3, zero, 0x0			r9	0x00000000	
0x0000000c	18c80004	addi r3, r3, 0x2000			r10	0x00000000	
					r11	0x00000000	11
		LOOP: 1dbio r4, 0(r2)			r12	0x00000000	11
		LOOP:			r13	0x00000000	18
0×00000010	11000027	1dbio r4, $0(r2)$			r14	0x00000000	11
0X0000010	11000027	stbio r4, 0(r3)			r15	0x00000000	11
		SCD10 14, 0(13)			r16	0x00000000	18
•					r17	0x00000000	18
Disassembly / E	Breakpoints 👝	Memory / Watches / Trace /			r18	0x00000000	110
erminal					<u>r19</u>	0*0000000	×
TAC HART Link	r ogtobligh	ed using cable "USB-Blaster	Verified OK				4
USB-1]", devi		-	Connection established to	GDB ser	ver at lo	calhost:240	24
obb-ij, devi	ice i, inst		Symbols loaded.				
			Source code loaded.				
			INFO: Program Trace not er	abled,	because t	race requir	
						•	
			Info & Errors / GDB Server /				

Figure 33. Display of the downloaded program.

Copyright ©1991-2013 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.