Using the SDRAM on Intel's DE2-115 Board with VHDL Designs

For Quartus Prime 16.1

1 Introduction

This tutorial explains how the SDRAM chips on Intel's DE2-115 Development and Education board can be used with a Nios II system implemented by using the Intel Qsys tool. The discussion is based on the assumption that the reader has access to a DE2-115 board and is familiar with the material in the tutorial *Introduction to the Intel Qsys System Integration Tool*.

The screen captures in the tutorial were obtained using the Quartus® Prime version 16.1; if other versions of the software are used, some of the images may be slightly different.

Contents:

- Example Nios II System
- The SDRAM Interface
- Using the Qsys tool to Generate the Nios II System
- Integration of the Nios II System into the Quartus Prime Project
- Using the Clock Signals IP Core

2 Background

The introductory tutorial *Introduction to the Intel Qsys System Integration Tool* explains how the memory in a Cyclone series FPGA chip can be used in the context of a simple Nios II system. For practical applications it is necessary to have a much larger memory. The Intel DE2-115 board contains 2 SDRAM chips that can each store 64 Mbytes of data. Each chip is organized as 8M x 16 bits x 4 banks. The SDRAM chips require careful timing control. To provide access to the SDRAM chips, the Qsys tool implements an *SDRAM Controller* circuit. This circuit generates the signals needed to deal with the SDRAM chips.

3 Example Nios II System

As an illustrative example, we will add the SDRAM to the Nios II system described in the *Introduction to the Intel Qsys System Integration Tool* tutorial. Figure 1 gives the block diagram of our example system.

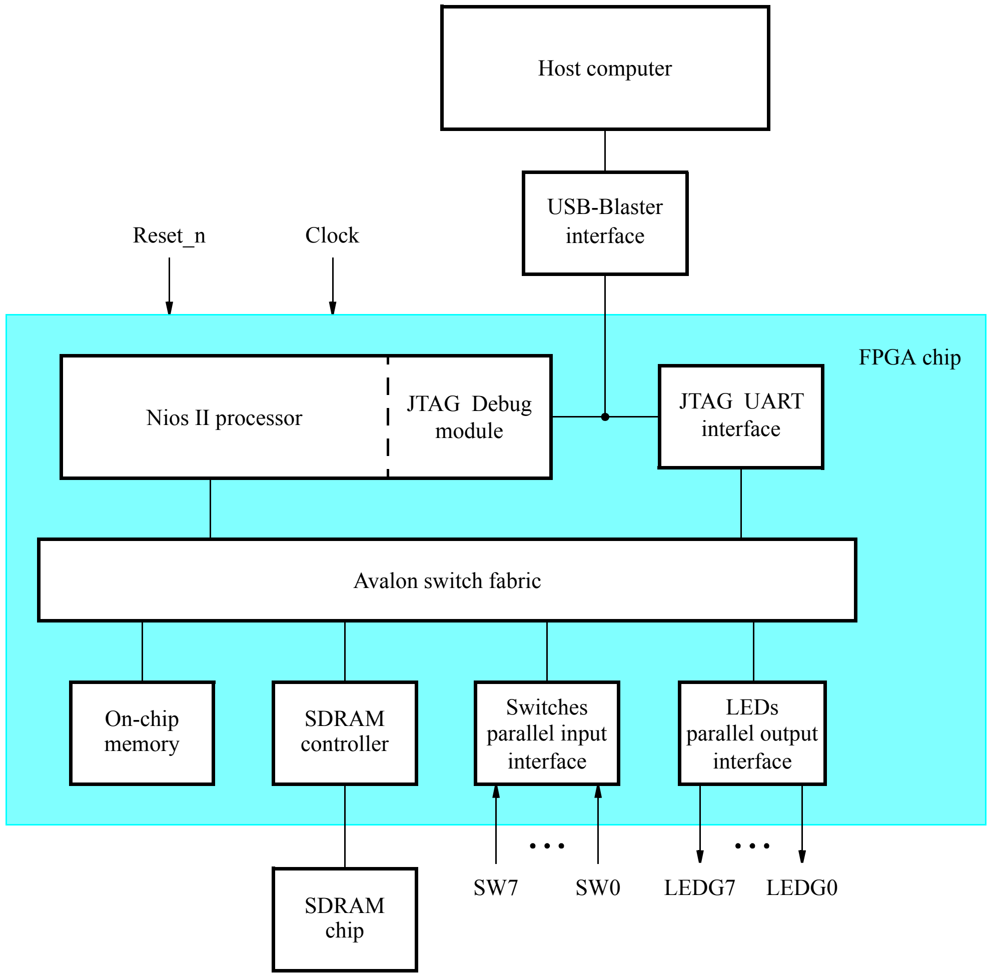


Figure 1. Example Nios II system implemented on the DE2-115 board.

The system realizes a trivial task. Eight toggle switches on the DE2-115 board, $SW7-0$, are used to turn on or off the eight green LEDs, $LEDG7-0$. The switches are connected to the Nios II system by means of a parallel I/O interface configured to act as an input port. The LEDs are driven by the signals from another parallel I/O interface configured to act as an output port. 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 an application program. Continuous operation is required, such that as the switches are toggled the lights change accordingly.

The introductory tutorial showed how we can use the Qsys tool to design the hardware needed to implement this task, assuming that the application program which reads the state of the toggle switches and sets the green LEDs accordingly is loaded into a memory block in the FPGA chip. In this tutorial, we will explain how SDRAM chips on the DE2-115 board can be included in the system in Figure 1, so that our application program can be run from the SDRAM rather than from the on-chip memory.

Doing this tutorial, the reader will learn about:

- Using the Qsys tool to include an SDRAM interface for a Nios II-based system
- Timing issues with respect to the SDRAM on the DE2-115 board

4 The SDRAM Interface

The two SDRAM chips on the DE2-115 board each have a capacity of 512 Mbits (64 Mbytes). Each chip is organized as 8M x 16 bits x 4 banks. The signals needed to communicate with a chip are shown in Figure 2. All of the signals, except the clock, can be provided by the SDRAM Controller that can be generated by using the Qsys tool. The clock signal is provided separately. It has to meet the clock-skew requirements as explained in section 7. Note that some signals are active low, which is denoted by the suffix N.

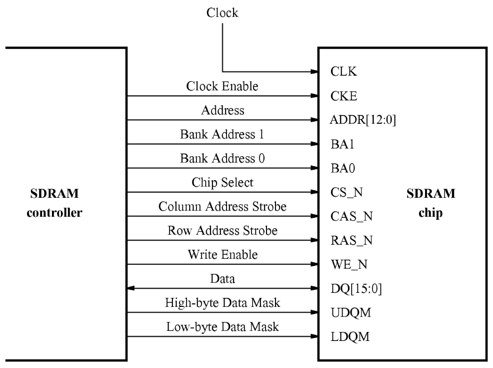
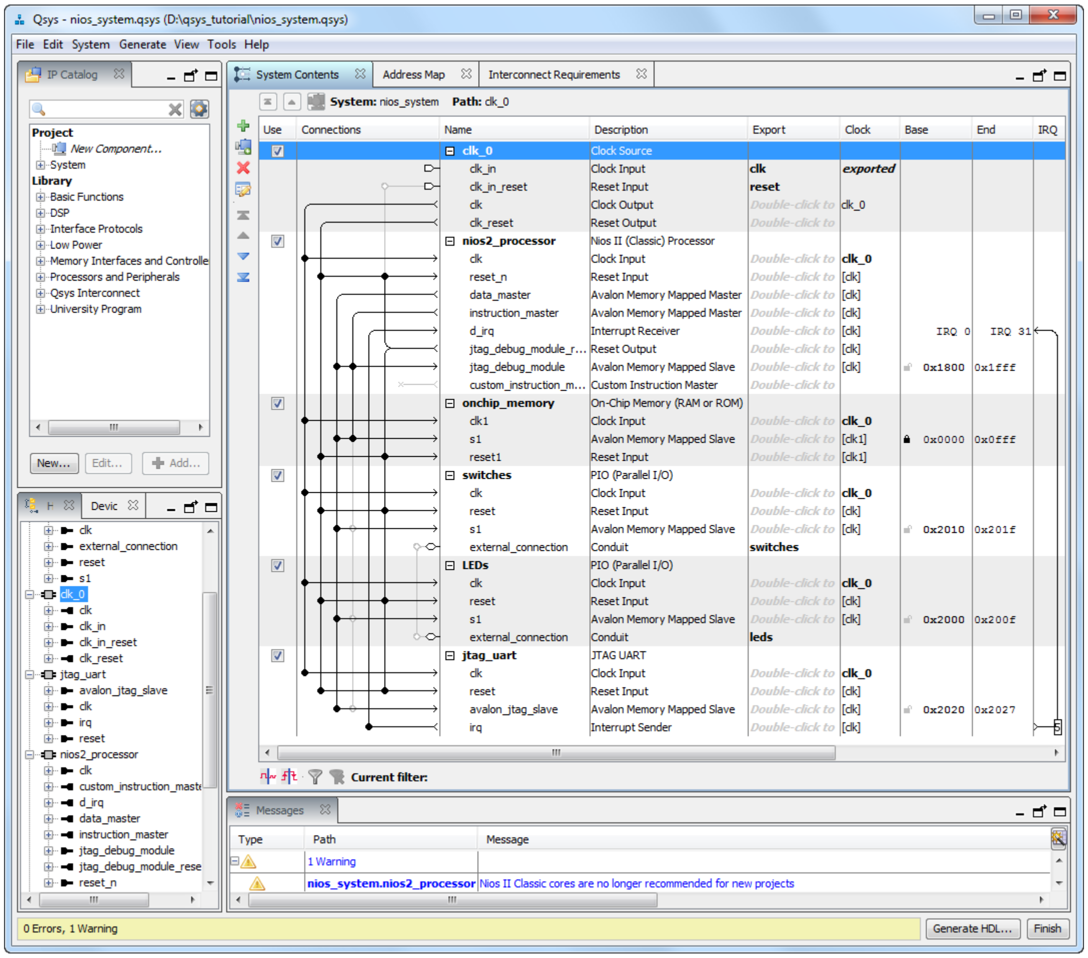


Figure 2. The SDRAM signals.

5 Using the Qsys tool to Generate the Nios II System

Our starting point will be the Nios II system discussed in the *Introduction to the Intel Qsys System Integration Tool* tutorial, which we implemented in a project called *lights*. We specified the system shown in Figure 3.



System: nios_system Path: clk_0

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
✓		clk_0	Clock Source					
		clk_in	Clock Input	**clk**	***exported***			
		clk_in_reset	Reset Input	**reset**				
		clk	Clock Output	*Double-click to*	clk_0			
		clk_reset	Reset Output	*Double-click to*				
✓		**nios2_processor**	Nios II (Classic) Processor					
		clk	Clock Input	*Double-click to*	**clk_0**			
		reset_n	Reset Input	*Double-click to*	[clk]			
		data_master	Avalon Memory Mapped Master	*Double-click to*	[clk]			
		instruction_master	Avalon Memory Mapped Master	*Double-click to*	[clk]			
		d_irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31	
		jtag_debug_module_r...	Reset Output	*Double-click to*	[clk]			
		jtag_debug_module	Avalon Memory Mapped Slave	*Double-click to*	[clk]	0x1800	0x1fff	
		custom_instruction_m...	Custom Instruction Master	*Double-click to*				
✓		**onchip_memory**	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	*Double-click to*	**clk_0**			
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk1]	0x0000	0x0fff	
		reset1	Reset Input	*Double-click to*	[clk1]			
✓		**switches**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to*	**clk_0**			
		reset	Reset Input	*Double-click to*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	0x2010	0x201f	
		external_connection	Conduit	**switches**				
✓		**LEDs**	PIO (Parallel I/O)					
		clk	Clock Input	*Double-click to*	**clk_0**			
		reset	Reset Input	*Double-click to*	[clk]			
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]	0x2000	0x200f	
		external_connection	Conduit	**leds**				
✓		**jtag_uart**	JTAG UART					
		clk	Clock Input	*Double-click to*	**clk_0**			
		reset	Reset Input	*Double-click to*	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]	0x2020	0x2027	
		irq	Interrupt Sender	*Double-click to*	[clk]			5

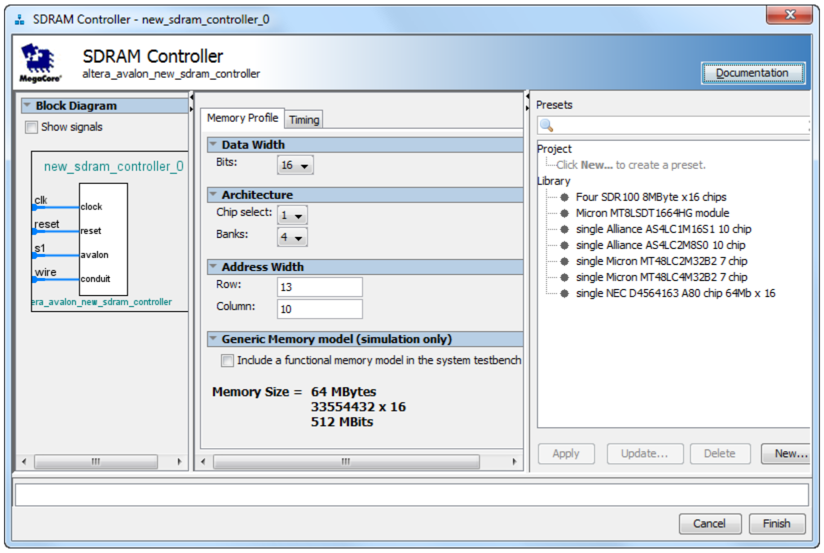
Type	Path	Message
	1 Warning	
	nios_system.nios2_processor	Nios II Classic cores are no longer recommended for new projects

0 Errors, 1 Warning

Figure 3. The Nios II system defined in the introductory tutorial.

If you saved the *lights* project, then open this project in the Quartus Prime software and then open the Qsys tool. Otherwise, you need to create and implement the project, as explained in the introductory tutorial, to obtain the system shown in the figure.

To add the SDRAM, in the window of Figure 3 select Memory Interfaces and Controllers > SDRAM > SDRAM Controller and click Add. A window depicted in Figure 4 appears. Set the **Data Width** parameter to **32** bits, the Row Width to **13** bits, the Column Width to **10** bits, and leave the default values for the rest. Since we will not simulate the system in this tutorial, do not select the option Include a functional memory model in the system testbench.


SDRAM Controller - new_sdram_controller_0
SDRAM Controller
altera_avalon_new_sdram_controller
Documentation
Block Diagram
Show signals
new_sdram_controller_0
clk
clock
reset
reset
s1
avalon
wire
conduit
era_avalon_new_sdram_controller
Memory Profile
Timing
Presets
Project
Click New... to create a preset.
Library
Four SDR100 8MByte x16 chips
Micron MT8LSDT1664HG module
single Alliance AS4LC1M16S1 10 chip
single Alliance AS4LC2M8S0 10 chip
single Micron MT48LC2M32B2 7 chip
single Micron MT48LC4M32B2 7 chip
single NEC D4564163 A80 chip 64Mb x 16
Apply
Update...
Delete
New...
Cancel
Finish

Data Width
Bits: 16

Architecture
Chip select: 1
Banks: 4

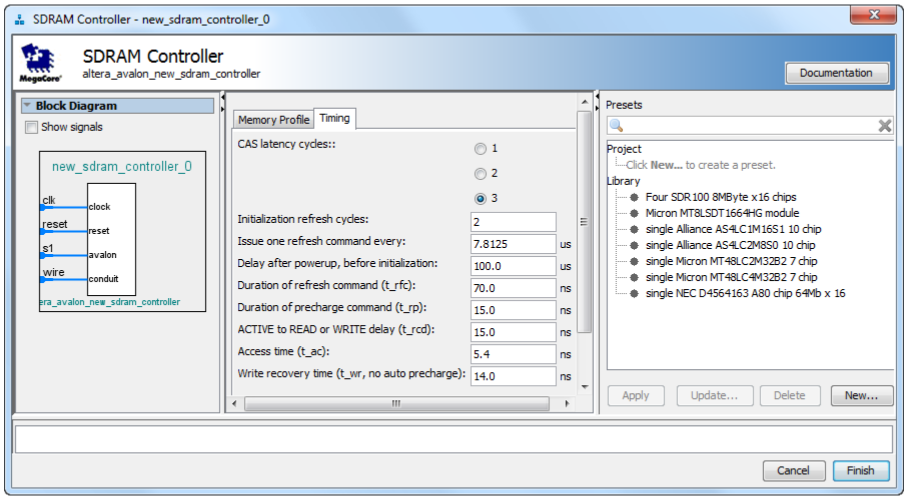
Address Width
Row: 13
Column: 10

Generic Memory model (simulation only)
☐ Include a functional memory model in the system testbench

Memory Size = 64 MBytes
33554432 x 16
512 MBits

Figure 4. Add the SDRAM Controller.

Select the *Timing* tab to get to the window in Figure 5. Configure the SDRAM timing parameters by setting **Issue one refresh command every** to **7.8125** microseconds and **Delay after powerup, before initialization** to **200** microseconds. Click Finish.

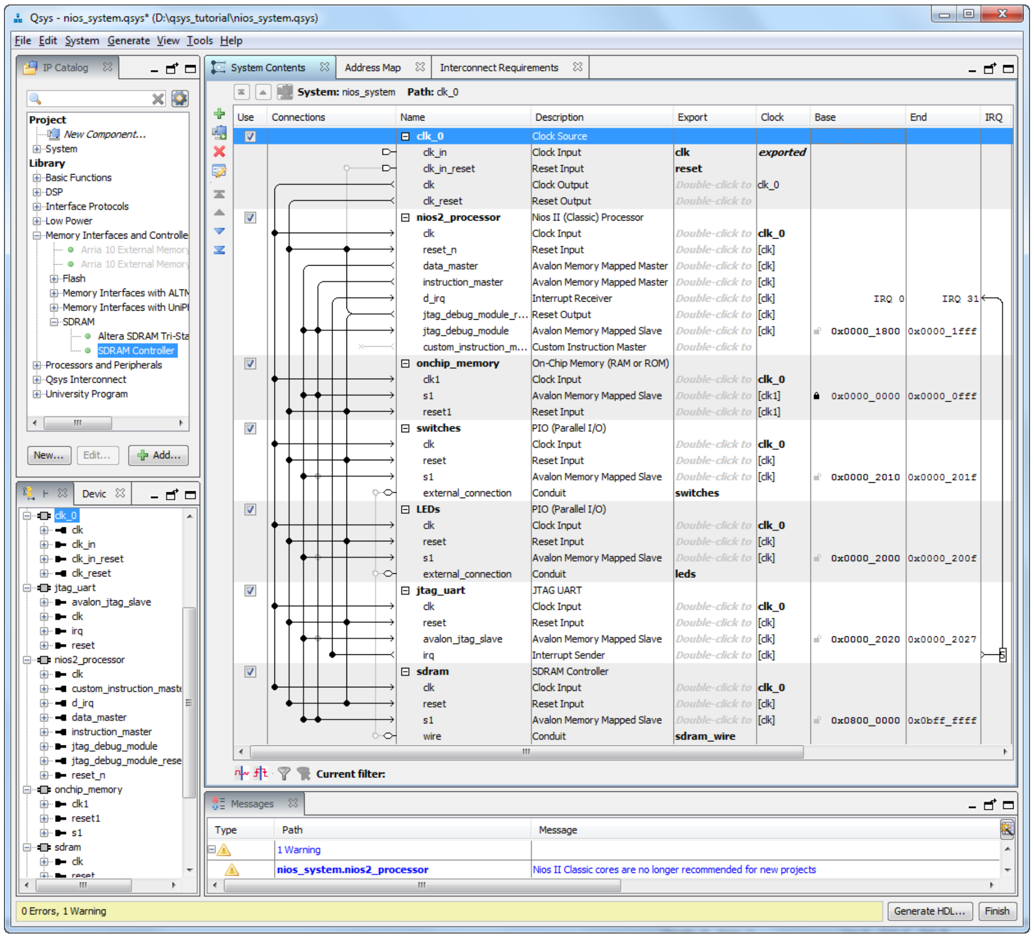

SDRAM Controller - new_sdram_controller_0
SDRAM Controller
altera_avalon_new_sdram_controller
Documentation
Block Diagram
Show signals
new_sdram_controller_0
clk
clock
reset
reset
s1
avalon
wire
conduit
era_avalon_new_sdram_controller
Memory Profile
Timing
Presets
Project
Click New... to create a preset.
Library
Four SDR100 8MByte x16 chips
Micron MT8LSDT1664HG module
single Alliance AS4LC1M16S1 10 chip
single Alliance AS4LC2M8S0 10 chip
single Micron MT48LC2M32B2 7 chip
single Micron MT48LC4M32B2 7 chip
single NEC D4564163 A80 chip 64Mb x 16
Apply
Update...
Delete
New...
Cancel
Finish

CAS latency cycles:: 1 2 [✓ 3]

Initialization refresh cycles:	2	
Issue one refresh command every:	7.8125	us
Delay after powerup, before initialization:	100.0	us
Duration of refresh command (t_rfc):	70.0	ns
Duration of precharge command (t_rp):	15.0	ns
ACTIVE to READ or WRITE delay (t_rcd):	15.0	ns
Access time (t_ac):	5.4	ns
Write recovery time (t_wr, no auto precharge):	14.0	ns

Figure 5. SDRAM Timings

Now, in the window of Figure 3, there will be an **sdram controller** module added to the design. Rename this module to *sdram*. Connect the SDRAM to the rest of the system in the same manner as the on-chip memory, and export the SDRAM wire port. Double-click on the Base Address of the *sdram* and enter the value 0x08000000 to produce the assignment shown in Figure 6.



Qsys - nios_system.qsys* (D:\qsys_tutorial\nios_system.qsys)

System: nios_system Path: clk_0

Use	Name	Description	Export	Clock	Base	End	IRQ
✓	clk_0	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	Double-click to	clk_0			
	clk_reset	Reset Output	Double-click to				
✓	nios2_processor	Nios II (Classic) Processor					
	clk	Clock Input	Double-click to	clk_0			
	reset_n	Reset Input	Double-click to	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]			
	d_irq	Interrupt Receiver	Double-click to	[clk]	IRQ 0	IRQ 31	
	jtag_debug_module_r…	Reset Output	Double-click to	[clk]			
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_1800	0x0000_1fff	
	custom_instruction_m…	Custom Instruction Master	Double-click to				
✓	onchip_memory	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to	clk_0			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk1]	0x0000_0000	0x0000_0fff	
	reset1	Reset Input	Double-click to	[clk1]			
✓	switches	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_2010	0x0000_201f	
	external_connection	Conduit	switches				
✓	LEDs	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_2000	0x0000_200f	
	external_connection	Conduit	leds				
✓	jtag_uart	JTAG UART					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_2020	0x0000_2027	
	irq	Interrupt Sender	Double-click to	[clk]			5
✓	sdram	SDRAM Controller					
	clk	Clock Input	Double-click to	clk_0			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_0000	0x0bff_ffff	
	wire	Conduit	sdram_wire				

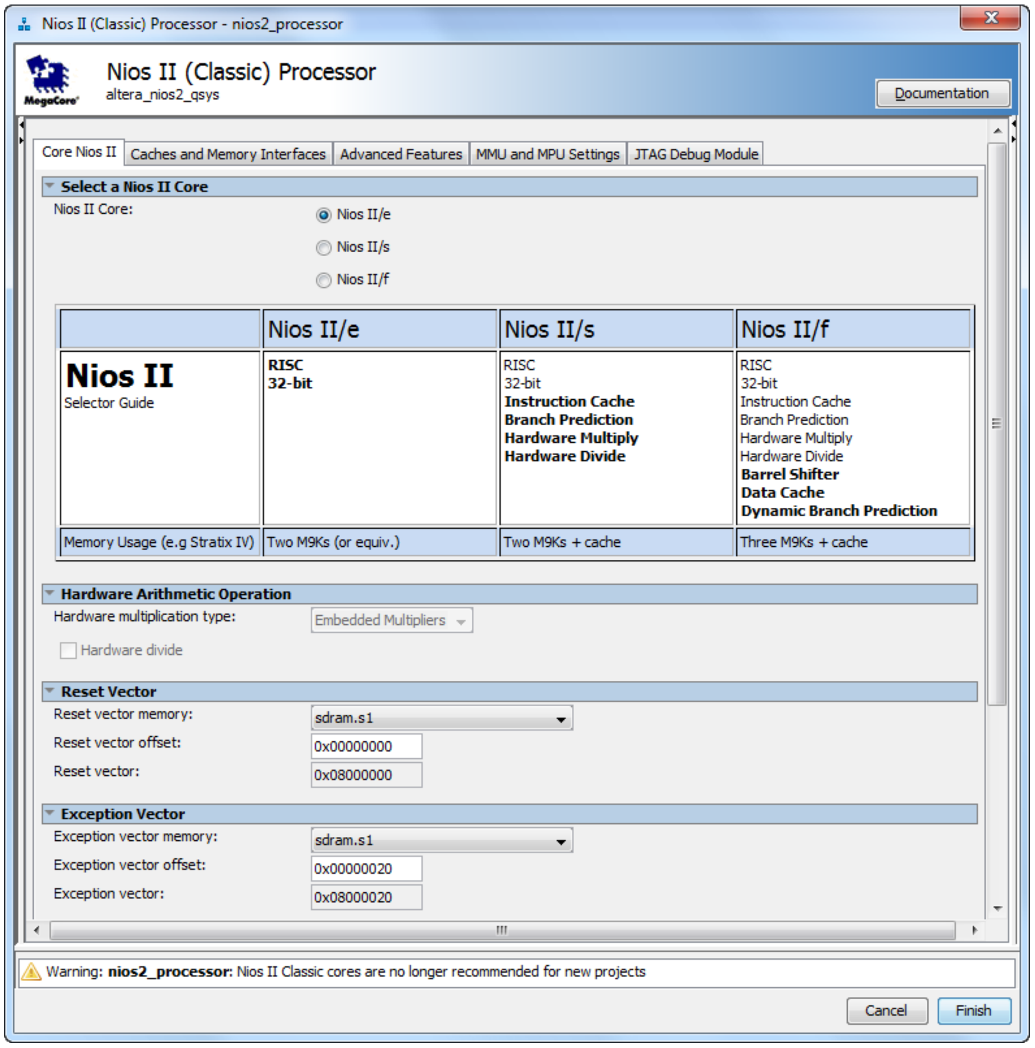
Messages

Type	Path	Message
⚠	1 Warning	
⚠	nios_system.nios2_processor	Nios II Classic cores are no longer recommended for new projects

0 Errors, 1 Warning

Figure 6. The expanded Nios II system.

To make use of the SDRAM, we need to configure the reset vector and exception vector of the Nios II processor. Right-click on the nios2_processor and then select Edit to reach the window in Figure 7. Select sdram to be the memory device for both reset vector and exception vector, as shown in the figure. Click Finish to return to the System Contents tab and regenerate the system.



Select a Nios II Core

Nios II Core: [✓ Nios II/e] Nios II/s Nios II/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 Operation

Hardware multiplication type:	Embedded Multipliers
Hardware divide	☐

Reset Vector

Reset vector memory:	sdram.s1
Reset vector offset:	0x00000000
Reset vector:	0x08000000

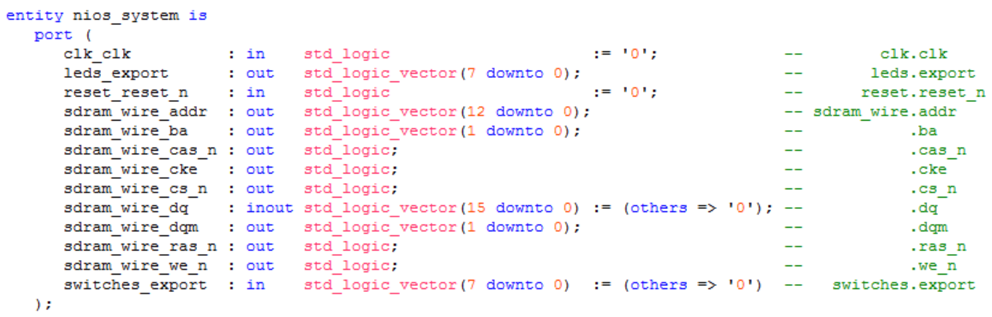
Exception Vector

Exception vector memory:	sdram.s1
Exception vector offset:	0x00000020
Exception vector:	0x08000020

Warning: **nios2_processor:** Nios II Classic cores are no longer recommended for new projects

Figure 7. Define the reset vector and the exception vector.

The Qsys tool generates an HDL file for the system, which can then be instantiated in a VHDL file. The augmented VHDL entity generated by the Qsys tool is in the file *nios_system.v* in the nios_system\synthesis directory of the project. Figure 8 depicts the portion of the code that defines the input and output signals for the entity *nios_system*. As in our initial system that we developed in the introductory tutorial, 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. A new entity, called *sdram*, is included. It involves the signals indicated in Figure 2. For example, the address lines are referred to as the **output** vector *sdram_wire_addr[12:0]*. The data lines are referred to as the **inout** vector *sdram_wire_dq[31:0]*. This is a vector of the **inout** type because the data lines are bidirectional.



```
entity nios_system is
   port (
      clk_clk          : in    std_logic                        := '0';             --        clk.clk
      leds_export      : out   std_logic_vector(7 downto 0);                        --       leds.export
      reset_reset_n    : in    std_logic                        := '0';             --      reset.reset_n
      sdram_wire_addr  : out   std_logic_vector(12 downto 0);                       -- sdram_wire.addr
      sdram_wire_ba    : out   std_logic_vector(1 downto 0);                        --           .ba
      sdram_wire_cas_n : out   std_logic;                                           --           .cas_n
      sdram_wire_cke   : out   std_logic;                                           --           .cke
      sdram_wire_cs_n  : out   std_logic;                                           --           .cs_n
      sdram_wire_dq    : inout std_logic_vector(15 downto 0) := (others => '0'); --           .dq
      sdram_wire_dqm   : out   std_logic_vector(1 downto 0);                        --           .dqm
      sdram_wire_ras_n : out   std_logic;                                           --           .ras_n
      sdram_wire_we_n  : out   std_logic;                                           --           .we_n
      switches_export  : in    std_logic_vector(7 downto 0)  := (others => '0')  --   switches.export
   );
```

Figure 8. A part of the generated VHDL entity.

6 Integration of the Nios II System into the Quartus II Project

Now, we have to instantiate the expanded Nios II system in the top-level VHDL entity, as we have done in the tutorial *Introduction to the Intel Qsys System Integration Tool*. The entity is named *lights*, because this is the name of the top-level design entity in our Quartus Prime project.

A first attempt at creating the new entity is presented in Figure 9. The input and output ports of the entity use the pin names for the 50-MHz clock, *CLOCK_50*, pushbutton switches, *KEY*, toggle switches, *SW*, and green LEDs, *LEDG*, as used in our original design. They also use the pin names *DRAM_CLK*, *DRAM_CKE*, *DRAM_ADDR*, *DRAM_BA*, *DRAM_CS_N*, *DRAM_CAS_N*, *DRAM_RAS_N*, *DRAM_WE_N*, *DRAM_DQ*, and *DRAM_DQM*, which correspond to the SDRAM signals indicated in Figure 2. All of these names are those specified in the DE2-115 User Manual and included in the file called *DE2_115.qsf*, which can be found on Intel's DE1-SoC web page at `https://www.altera.com/support/training/university/boards.html`

Finally, note that we tried an obvious approach of using the 50-MHz system clock, *CLOCK_50*, as the clock signal, *DRAM_CLK*, for the SDRAM chips. This is specified by the last assignment statement in the code. This approach leads to a potential timing problem caused by the clock skew on the DE2-115 board, which can be fixed as explained in section 7.

```
-- 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.
--           SDRAM ports correspond to the signals in Figure 2; their names are those
--           used in the DE2-115 User Manual.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY lights IS
   PORT (
      SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      KEY : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;
      LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      DRAM_DQ : INOUT STD_LOGIC_VECTOR (31 DOWNTO 0);
      DRAM_ADDR : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
      DRAM_BA : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      DRAM_CAS_N, DRAM_RAS_N, DRAM_CLK : OUT STD_LOGIC;
      DRAM_CKE, DRAM_CS_N, DRAM_WE_N : OUT STD_LOGIC;
      DRAM_DQM : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END lights;
ARCHITECTURE Structure OF lights IS
   COMPONENT nios_system
      PORT (
            clk_clk : IN STD_LOGIC;
            reset_reset_n : IN STD_LOGIC;
            leds_export : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            switches_export : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            sdram_wire_addr : OUT STD_LOGIC_VECTOR(12 DOWNTO 0);
            sdram_wire_ba : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
            sdram_wire_cas_n : OUT STD_LOGIC;
            sdram_wire_cke : OUT STD_LOGIC;
            sdram_wire_cs_n : OUT STD_LOGIC;
            sdram_wire_dq : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
```

... continued in Part *b*

Figure 9. A first attempt at instantiating the expanded Nios II system. (Part *a*)

```
            sdram_wire_dqm : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            sdram_wire_ras_n : OUT STD_LOGIC;
            sdram_wire_we_n : OUT STD_LOGIC);
   END COMPONENT;
BEGIN
-- Instantiate the Nios II system entity generated by the Qsys tool.
   NiosII: nios_system
      PORT MAP (
            clk_clk => CLOCK_50,
            reset_reset_n => KEY(0),
            leds_export => LEDG,
            switches_export => SW,
            sdram_wire_addr => DRAM_ADDR,
            sdram_wire_ba => DRAM_BA,
            sdram_wire_cas_n => DRAM_CAS_N,
            sdram_wire_cke => DRAM_CKE,
            sdram_wire_cs_n => DRAM_CS_N,
            sdram_wire_dq => DRAM_DQ,
            sdram_wire_dqm => DRAM_DQM,
            sdram_wire_ras_n => DRAM_RAS_N,
            sdram_wire_we_n => DRAM_WE_N );
      DRAM_CLK <= CLOCK_50;
END Structure;
```

Figure 9. A first attempt at instantiating the expanded Nios II system. (Part *b*).

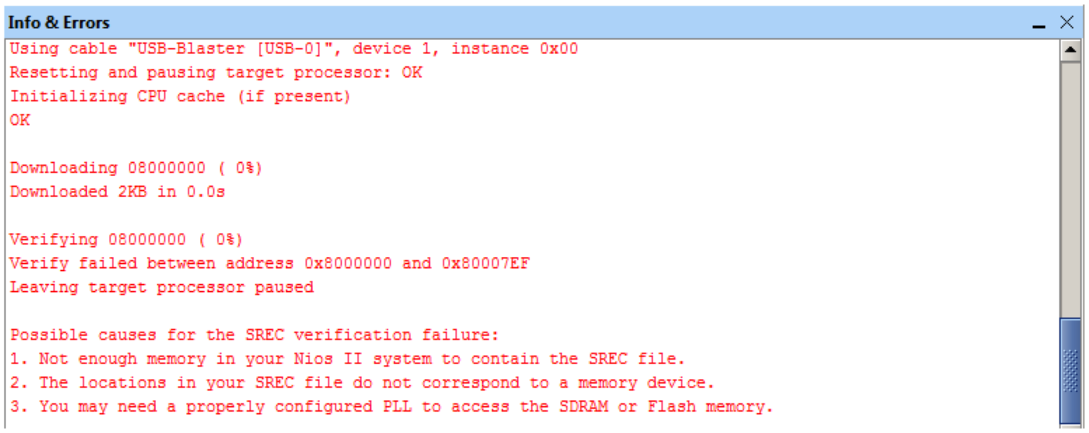
As an experiment, you can enter the code in Figure 9 into a file called *lights.vhd*. Add this file and all the *nios_system.qip* file produced by the Qsys tool to your Quartus Prime project. Compile the code and download the design into the Cyclone IV FPGA on the DE2-115 board. Use the application program from the tutorial *Introduction to the Intel Qsys System Integration Tool*, which is shown in Figure 10.

```
.equ      Switches, 0x00002010
.equ      LEDs, 0x00002000
.global   _start
_start:
          movia     r2, Switches
          movia     r3, LEDs
loop:     ldbio     r4, 0(r2)
          stbio     r4, 0(r3)
          br        loop
```

Figure 10. Assembly language code to control the lights.

Use the Intel FPGA Monitor Program, which is described in the tutorial *Intel FPGA Monitor Program Tutorial*, to assemble, download, and run this application program. If successful, the lights on the DE2-115 board will respond to the operation of the toggle switches.

Due to the clock skew problem mentioned above, the Nios II processor may be unable to properly access the SDRAM chip. A possible indication of this may be given by the Intel FPGA Monitor Program, which may display the message depicted in Figure 11. To solve the problem, it is necessary to modify the design as indicated in the next section.



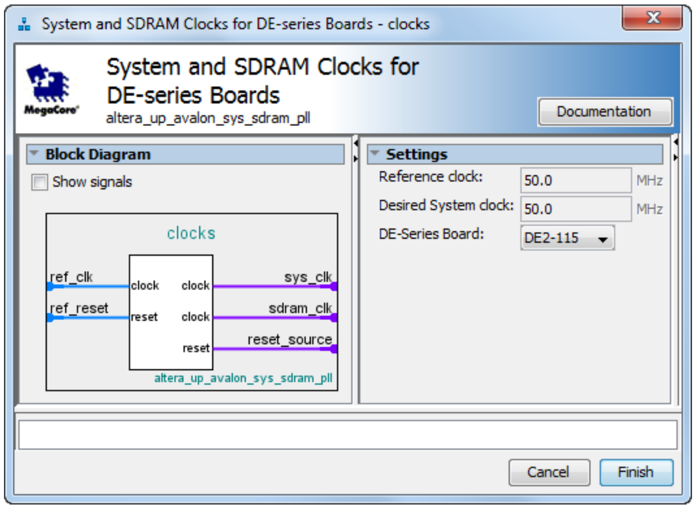
```
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK

Downloading 08000000 ( 0%)
Downloaded 2KB in 0.0s

Verifying 08000000 ( 0%)
Verify failed between address 0x8000000 and 0x80007EF
Leaving target processor paused

Possible causes for the SREC verification failure:
1. Not enough memory in your Nios II system to contain the SREC file.
2. The locations in your SREC file do not correspond to a memory device.
3. You may need a properly configured PLL to access the SDRAM or Flash memory.
```

Figure 11. Error message in the Intel FPGA Monitor Program that may be due to the SDRAM clock skew problem.

7 Using the Clock Signals IP Core

The clock skew depends on physical characteristics of the DE2-115 board. For proper operation of the SDRAM chip, it is necessary that its clock signal, *DRAM_CLK*, leads the Nios II system clock, *CLOCK_50*, by 3 nanoseconds. This can be accomplished by using a *phase-locked loop (PLL)* circuit which can be manually created using the *IP Catalog*. It can also be created automatically using the Clock Signals IP core provided by the Intel FPGA University Program. We will use the latter method in this tutorial.

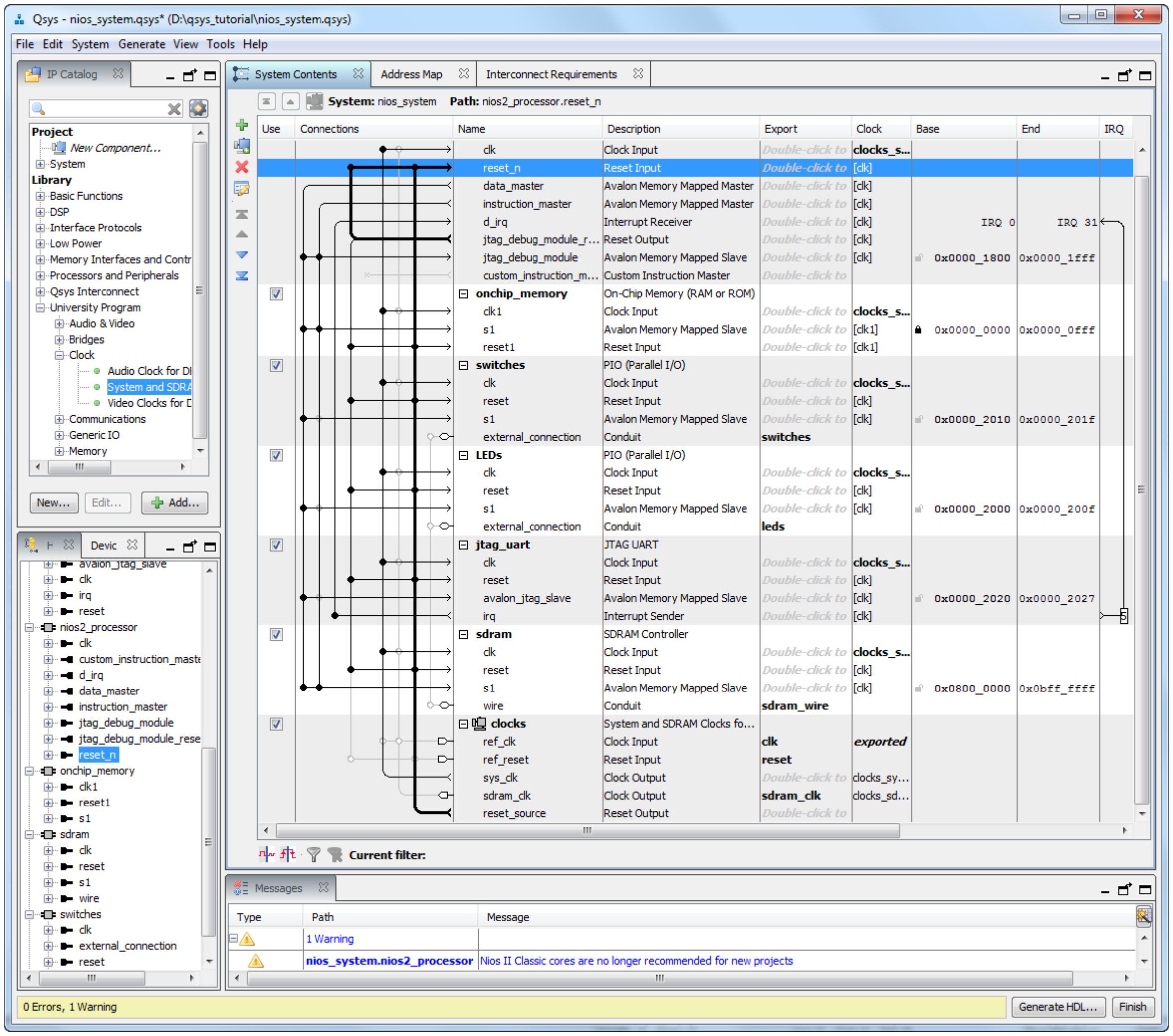
To add the Clock Signals IP core, in the Qsys tool window of Figure 5 select University Program > Clock > System and SDRAM Clocks for DE-Series Boards and click Add. A window depicted in Figure 12 appears. Select *DE2-115* from the DE Board drop-down list. Click Finish to return to the window in Figure 5.


System and SDRAM Clocks for DE-series Boards - clocks
System and SDRAM Clocks for DE-series Boards
altera_up_avalon_sys_sdram_pll
Documentation
Block Diagram
Show signals
clocks
ref_clk
ref_reset
clock
reset
clock
clock
reset
sys_clk
sdram_clk
reset_source
altera_up_avalon_sys_sdram_pll
Settings
Cancel
Finish

Reference clock: 50.0 MHz
Desired System clock: 50.0 MHz
DE-Series Board: DE2-115

Figure 12. Clock Signals IP Core

Remove the system clock component *clk_0*. All other IP cores (including the SDRAM) should be adjusted to use the *sys_clk* output of the Clock Signal core. Rename the Clock Signal core to *clocks* and export the *sdram_clk* signal under the name *sdram_clk*, the *ref_clk* signal under the name *clk*, and *ref_reset* signal under the name *reset*. The final system is shown in Figure 13. Click on Generate > Generate HDL... and regenerate the system.



Qsys - nios_system.qsys* (D:\qsys_tutorial\nios_system.qsys)

System: nios_system Path: nios2_processor.reset_n

Use	Name	Description	Export	Clock	Base	End	IRQ
	clk	Clock Input	Double-click to	clocks_s...			
	reset_n	Reset Input	Double-click to	[clk]			
	data_master	Avalon Memory Mapped Master	Double-click to	[clk]			
	instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]			
	d_irq	Interrupt Receiver	Double-click to	[clk]	IRQ 0	IRQ 31	
	jtag_debug_module_r...	Reset Output	Double-click to	[clk]			
	jtag_debug_module	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_1800	0x0000_1fff	
	custom_instruction_m...	Custom Instruction Master	Double-click to				
✓	onchip_memory	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	Double-click to	clocks_s...			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk1]	0x0000_0000	0x0000_0fff	
	reset1	Reset Input	Double-click to	[clk1]			
✓	switches	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to	clocks_s...			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_2010	0x0000_201f	
	external_connection	Conduit	switches				
✓	LEDs	PIO (Parallel I/O)					
	clk	Clock Input	Double-click to	clocks_s...			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_2000	0x0000_200f	
	external_connection	Conduit	leds				
✓	jtag_uart	JTAG UART					
	clk	Clock Input	Double-click to	clocks_s...			
	reset	Reset Input	Double-click to	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0000_2020	0x0000_2027	
	irq	Interrupt Sender	Double-click to	[clk]			5
✓	sdram	SDRAM Controller					
	clk	Clock Input	Double-click to	clocks_s...			
	reset	Reset Input	Double-click to	[clk]			
	s1	Avalon Memory Mapped Slave	Double-click to	[clk]	0x0800_0000	0x0bff_ffff	
	wire	Conduit	sdram_wire				
✓	clocks	System and SDRAM Clocks fo...					
	ref_clk	Clock Input	clk	exported			
	ref_reset	Reset Input	reset				
	sys_clk	Clock Output	Double-click to	clocks_sy...			
	sdram_clk	Clock Output	sdram_clk	clocks_sd...			
	reset_source	Reset Output	Double-click to				

Type	Path	Message
⚠	1 Warning	
⚠	nios_system.nios2_processor	Nios II Classic cores are no longer recommended for new projects

0 Errors, 1 Warning

Figure 13. The final Nios II system.

Next, we have to fix the top-level VHDL entity, given in Figure 9, to instantiate the Nios II system with the Clock Signals core included. The desired code is shown in Figure 14. The SDRAM clock signal *sdram_clk* generated by the Clock Signals core connects to the pin *DRAM_CLK*. Note that the *sys_clk* signal is not connected since it is for internal use only.

```
-- 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.
--            SDRAM ports correspond to the signals in Figure 2; their names are those
--            used in the DE2-115 User Manual.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY lights IS
   PORT (
      SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      KEY : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      CLOCK_50 : IN STD_LOGIC;
      LEDG : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
      DRAM_DQ : INOUT STD_LOGIC_VECTOR (31 DOWNTO 0);
      DRAM_ADDR : OUT STD_LOGIC_VECTOR (12 DOWNTO 0);
      DRAM_BA : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);
      DRAM_CAS_N, DRAM_RAS_N, DRAM_CLK : OUT STD_LOGIC;
      DRAM_CKE, DRAM_CS_N, DRAM_WE_N : OUT STD_LOGIC;
      DRAM_DQM : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
END lights;
ARCHITECTURE Structure OF lights IS
   COMPONENT nios_system
      PORT (
            clk_clk : IN STD_LOGIC;
            reset_reset : IN STD_LOGIC;
            sdram_clk_clk : OUT STD_LOGIC;
            leds_export : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            switches_export : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            sdram_wire_addr : OUT STD_LOGIC_VECTOR(12 DOWNTO 0);
            sdram_wire_ba : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
            sdram_wire_cas_n : OUT STD_LOGIC;
            sdram_wire_cke : OUT STD_LOGIC;
            sdram_wire_cs_n : OUT STD_LOGIC;
            sdram_wire_dq : INOUT STD_LOGIC_VECTOR(31 DOWNTO 0);
```

... continued in Part *b*

Figure 14. Proper instantiation of the expanded Nios II system. (Part *a*).

```
                sdram_wire_dqm : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                sdram_wire_ras_n : OUT STD_LOGIC;
                sdram_wire_we_n : OUT STD_LOGIC);
    END COMPONENT;
BEGIN
-- Instantiate the Nios II system entity generated by the Qsys tool.
    NiosII: nios_system
        PORT MAP (
                clk_clk => CLOCK_50,
                reset_reset => NOT KEY(0),
                sdram_clk_clk => DRAM_CLK,
                leds_export => LEDG,
                switches_export => SW,
                sdram_wire_addr => DRAM_ADDR,
                sdram_wire_ba => DRAM_BA,
                sdram_wire_cas_n => DRAM_CAS_N,
                sdram_wire_cke => DRAM_CKE,
                sdram_wire_cs_n => DRAM_CS_N,
                sdram_wire_dq => DRAM_DQ,
                sdram_wire_dqm => DRAM_DQM,
                sdram_wire_ras_n => DRAM_RAS_N,
                sdram_wire_we_n => DRAM_WE_N );
END Structure;
```

Figure 14. Proper instantiation of the expanded Nios II system. (Part *b*).

Compile the code and download the design into the Cyclone IV FPGA on the DE2-115 board. Use the application program in Figure 10 to test the circuit.