Using the SDRAM Memory on Altera's DE2 Board with VHDL Design

This tutorial explains how the SDRAM chip on Altera's DE2 Development and Education board can be used with a Nios II system implemented by using the Altera SOPC Builder. The discussion is based on the assumption that the reader has access to a DE2 board and is familiar with the material in the tutorial *Introduction to the Altera SOPC Builder Using VHDL Design*.

The screen captures in the tutorial were obtained using the Quartus^{\mathbb{R}} II version 8.0; 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 SOPC Builder to Generate the Nios II System Integration of the Nios II System into the Quartus II Project Using a Phase-Locked Loop The introductory tutorial *Introduction to the Altera SOPC Builder Using VHDL Design* explains how the memory in the Cyclone II 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 Altera DE2 board contains an SDRAM chip that can store 8 Mbytes of data. This memory is organized as 1M x 16 bits x 4 banks. The SDRAM chip requires careful timing control. To provide access to the SDRAM chip, the SOPC Builder implements an *SDRAM Controller* circuit. This circuit generates the signals needed to deal with the SDRAM chip.

1 Example Nios II System

As an illustrative example, we will add the SDRAM to the Nios II system described in the *Introduction to the Altera SOPC Builder Using VHDL Design* tutorial. Figure 1 gives the block diagram of our example system.

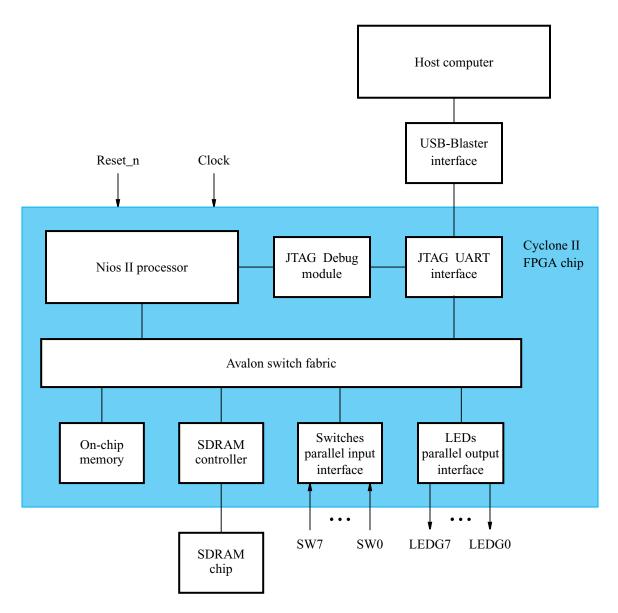


Figure 1. Example Nios II system implemented on the DE2 board.

The system realizes a trivial task. Eight toggle switches on the DE2 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 SOPC Builder 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 the SDRAM chip on the DE2 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 SOPC Builder to include an SDRAM interface for a Nios II-based system
- Timing issues with respect to the SDRAM on the DE2 board
- Using a phase-locked loop (PLL) to control the clock timing

2 The SDRAM Interface

The SDRAM chip on the DE2 board has the capacity of 64 Mbits (8 Mbytes). It is organized as $1M \times 16$ bits x 4 banks. The signals needed to communicate with this 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 SOPC Builder. The clock signal is provided separately. It has to meet the clock-skew requirements as explained in section 5. Note that some signals are active low, which is denoted by the suffix N.

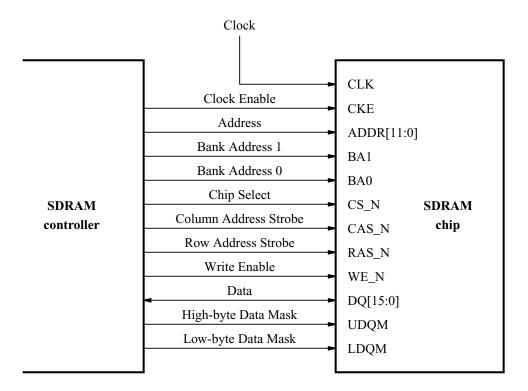


Figure 2. The SDRAM signals.

3 Using the SOPC Builder to Generate the Nios II System

Our starting point will be the Nios II system discussed in the *Introduction to the Altera SOPC Builder Using VHDL Design* tutorial, which we implemented in a project called *lights*. We specified the system shown in Figure 3.

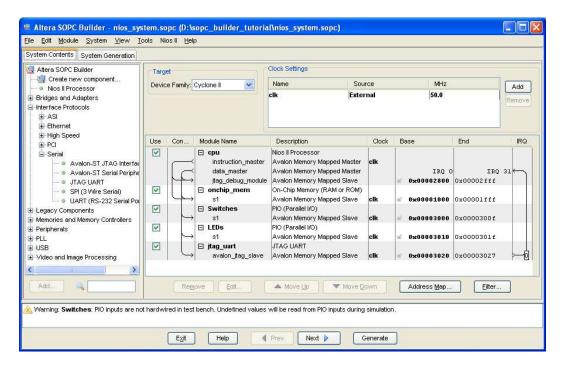


Figure 3. The Nios II system defined in the introductory tutorial.

If you saved the *lights* project, then open this project in the Quartus II software and then open the SOPC Builder. 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 Memories and Memory Controllers > SDRAM > SDRAM Controller and click Add. A window depicted in Figure 4 appears. Select *Custom* from the Presets drop-down list. Set the Data Width parameter to 16 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. Click Finish. Now, in the window of Figure 3, there will be an **sdram** module added to the design. Select the command System > Auto-Assign Base Addresses to produce the assignment shown in Figure 5. Observe that the SOPC Builder assigned the base address 0x00800000 to the SDRAM. 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 cpu and then select Edit to reach the window in Figure 6. 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.

SDR	AM Controller		Documentation
Parameter Settings			
20 August 19	iming >		
Presets: Custom		~	
Data width			
Bits: 16			
Architecture			
Chip select: 1	Banks: 4	~	
Address widths			
Row: 12	Column: 8		
Share pins via tristat	e bridge		
Controller shares	dq/dqm/addr I/O pins		
Tristate bridge selecti	on:	1	
Generic memory mo	del (simulation only)		
nclude a function	al memory model in the system	testbench	
	Memory size = 8 4194304 × 1 64 MBits		

Figure 4. Add the SDRAM Controller.

System Contents System Generation										
🙀 Altera SOPC Builder	Targ	et		Clock Settings						
Create new component Nios Il Processor	Devic	e Family:	Cyclone II 🛛 👻	Name	Sou	rce		MHz		Add
Bridges and Adapters Interface Protocols Legacy Components Memories and Memory Controllers				cik	Exter	nal		50.0		Remove
⊕ -DMA ⊕ -Flash	Use	Con	Module Name	Description		Clock	Base		End	IRQ
On-Chip DDR SDRAM Controller N DDR SDRAM High Perfor DDR2 SDRAM High Perfor DDR2 SDRAM High Perfor DDR3 SDRAM High Perfc SDRAM Controller SRAM Peripherals P-LL USB		(t t t t t	 cpu instruction_master data_master data_master flag_debug_module onchip_mem s1 Switches s1 LEDs s1 jtag_uart avalon_flag_slave slava 	Nios II Processor Avaion Memory Mi Avaion Memory Mi Avaion Memory Mi On-Chip Memory (Mi Plo (Parailel I/O) Avaion Memory Mi Plo (Parailel I/O) Avaion Memory Mi JTAG UART Avaion Memory Mi SDRAM Controller	apped Master apped Slave RAM or ROM) apped Slave apped Slave apped Slave	cik cik cik cik cik	= 0x0 = 0x0	LOO1OOO LOO3OOO LOO3O1O	IRQ 0x01002fff 0x0100300f 0x0100300f 0x0100301f 0x01003027	31
Video and Image Processing		\square	s1	Avalon Memory Ma	apped Slave	cik	* 0x0	800000	0x00ffffff	5 2
Add	hardwi	Rem red in test		Move Up	▼ Move <u>D</u> e D inputs during			ess <u>M</u> ap	<u>Filter.</u>	

Figure 5. The expanded Nios II system.

Nios II Processor	- сри			×
MegaCore Nios	s II Processor			Documentation
Parameter Settings				
	hes and Memory Interfaces	Advanced Features	MMU and MPU Settings $ ightarrow$ JTAG Det	oug Module > Custom Instructions 💙
Core Nios II				
Select a Nios II core:				
	Nios II/e	ONios II/s	ONios II/f	
Nios II Selector Guide Family: Cyclone II f _{system:} 50.0 MHz cpuid: 0	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	
Performance at 50.0 MH	Hz Up to 5 DMIPS	Up to 25 DMIPS	Up to 51 DMIPS	
Logic Usage	600-700 LEs	1200-1400 LEs	1400-1800 LEs	
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache	
Hardware Multiply:	mory: sdram	Hardware Divide Offset: 0x0	0×00800000	
Exception Vector: Mer	nory: adrem	Offset: 0x20	0x00800020	
Include MMU Only include the MMU w Fast TLB Miss Exceptio	vhen using an operating system n Vector: Memory:	that explicitly supports an MN		
				Cancel < Back (Next >) (Einish

Figure 6. Define the reset vector and the exception vector.

The augmented VHDL entity generated by the SOPC Builder is in the file *nios_system.vhd* in the directory of the project. Figure 7 depicts the portion of the code that defines the port 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 *in_port_to_the_Switches*. The 8-bit output vector is called *out_port_from_the_LEDs*. The clock and reset signals are called *clk* and *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 OUT vector *zs_dq_to_and_from_the_sdram[15:0]*. This is a vector of the INOUT type because the data lines are bidirectional.

nios_s	ystem.vhd	
3242	library ieee;	1
3243	use ieee.std logic 1164.all;	
3244	use ieee.std logic arith.all;	
3245	use ieee.std_logic_unsigned.all;	
3246		
3247	Entity nios_system is	
3248	port (
3249	1) global signals:	
3250	signal clk : IN STD_LOGIC;	
3251	signal reset_n : IN STD_LOGIC;	
3252		
3253	the LEDs	
3254	signal out_port_from_the_LEDs : OUT STD_LOGIC_VECTOR (7 DOWNTO D);	
3255		
3256	the_Switches	
3257	signal in_port_to_the_Switches : IN STD_LOGIC_VECTOR (7 DOWNTO 0);	
3258		
3259	the_sdram	
3260	signal zs_addr_from_the_sdram : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);	
3261	signal zs_ba_from_the_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO 0);	
3262	signal zs_cas_n_from_the_sdram : OUT STD_LOGIC;	
3263	<pre>signal zs_cke_from_the_sdram : OUT STD_LOGIC;</pre>	
3264	signal zs_cs_n_from_the_sdram : OUT STD_LOGIC;	
3265	signal zs_dq_to_and_from_the_sdram : INOUT_STD_LOGIC_VECTOR (15_DOWNTO_0);	
3266	signal zs_dqm_from_the_sdram : OUT STD_LOGIC_VECTOR (1 DOWNTO D);	
3267	signal zs_ras_n_from_the_sdram : OUT STD_LOGIC;	
3268	signal zs_we_n_from_the_sdram : OUT STD_LOGIC	
3269);	
3270 3271	end entity nios_system;	
		>

Figure 7. A part of the generated VHDL entity.

4 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 Altera SOPC Builder Using VHDL Design*. The entity is named *lights*, because this is the name of the top-level design entity in our Quartus II project.

A first attempt at creating the new entity is presented in Figure 8. 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_1*, *DRAM_BA_0*, *DRAM_CS_N*, *DRAM_CAS_N*, *DRAM_RAS_N*, *DRAM_WE_N*, *DRAM_DQ*, *DRAM_UDQM*, and *DRAM_LDQM*, which correspond to the SDRAM signals indicated in Figure 2. All of these names are those specified in the DE2 User Manual, which allows us to make the pin assignments by importing them from the file called *DE2_pin_assignments.csv* in the directory *DE2_tutorials\design_files*, which is included on the CD-ROM that accompanies the DE2 board and can also be found on Altera's DE2 web pages.

Observe that the two *Bank Address* signals are treated by the SOPC Builder as a two-bit vector called *zs_ba_from_the_sdram[1:0]*, as seen in Figure 7. However, in the *DE2_pin_assignments.csv* file these signals are given as separate signals *DRAM_BA_1* and *DRAM_BA_0*. This is accommodated by our VHDL code. Similarly, the vector *zs_dqm_from_the_sdram[1:0]* corresponds to the signals (*DRAM_UDQM* and *DRAM_LDQM*).

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 chip. 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 board, which can be fixed as explained in section 5.

- -- 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 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_CLK, DRAM_CKE : OUT STD_LOGIC; DRAM_ADDR : OUT STD_LOGIC_VECTOR(11 DOWNTO 0); DRAM_BA_1, DRAM_BA_0 : BUFFER STD_LOGIC; DRAM_CS_N, DRAM_CAS_N, DRAM_RAS_N, DRAM_WE_N : OUT STD_LOGIC; DRAM_DQ : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0); DRAM_UDQM, DRAM_LDQM : BUFFER STD_LOGIC);

END lights;

```
ARCHITECTURE Structure OF lights IS
```

COMPONENT nios_system

PORT (clk : IN STD_LOGIC; reset n: IN STD LOGIC; out_port_from_the_LEDs : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); in_port_to_the_Switches : IN STD_LOGIC_VECTOR(7 DOWNTO 0) zs_addr_from_the_sdram : OUT STD_LOGIC_VECTOR(11 DOWNTO 0); zs ba from the sdram : BUFFER STD LOGIC VECTOR(1 DOWNTO 0); zs cas n from the sdram : OUT STD LOGIC; zs cke from the sdram : OUT STD LOGIC: zs cs n from the sdram : OUT STD LOGIC; zs dq to and from the sdram : INOUT STD LOGIC VECTOR(15 DOWNTO 0); zs_dqm_from_the_sdram : BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0); zs_ras_n_from_the_sdram : OUT STD_LOGIC; zs_we_n_from_the_sdram : OUT STD_LOGIC); END COMPONENT: SIGNAL BA : STD_LOGIC_VECTOR(1 DOWNTO 0); SIGNAL DQM : STD_LOGIC_VECTOR(1 DOWNTO 0); BEGIN $DRAM_BA_1 \le BA(1); DRAM_BA_0 \le BA(0);$

```
DRAM_UDQM \le DQM(1); DRAM_LDQM \le DQM(0);
```

-- Instantiate the Nios II system entity generated by the SOPC Builder.

NiosII: nios_system PORT MAP (CLOCK_50, KEY(0), LEDG, SW,

DRAM_ADDR, BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N,

DRAM_DQ, DQM, DRAM_RAS_N, DRAM_WE_N);

 $DRAM_CLK \leq CLOCK_50;$

END Structure;

Figure 8. A first attempt at instantiating the expanded Nios II system.

As an experiment, you can enter the code in Figure 8 into a file called *lights.vhd*. Add this file and all the *.vhd files produced by the SOPC Builder to your Quartus II project. Compile the code and download the design into the Cyclone II FPGA on the DE2 board. Use the application program from the tutorial *Introduction to the Altera SOPC Builder Using VHDL Design*, which is shown in Figure 9. Notice in our expanded system, the addresses assigned by the SOPC Builder are 0x01003000 for Switches and 0x01003010 for LEDs, which are different from the original system. These changes are already reflected in the program in Figure 9.

.include "nios_macros.s" Switches, 0x01003000 .equ .equ LEDs, 0x01003010 .global _start _start: r2, Switches movia r3, LEDs movia r4, 0(r2)loop: ldbio stbio r4, 0(r3) br loop

Figure 9. Assembly language code to control the lights.

Use the Altera Monitor Program, which is described in the tutorial *Altera Monitor Program*, to assemble, download, and run this application program. If successful, the lights on the DE2 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 Altera Monitor Program, which may display the message depicted in Figure 10. To solve the problem, it is necessary to modify the design as indicated in the next section.

Info & Errors	_ ×
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00	4
Resetting and pausing target processor: OK	
Initializing CPU cache (if present)	
OK	
Downloading 00800000 (0%)	
Downloaded 1KB in 0.0s	
Verifying 00800000 (0%)	
Verify failed between address 0x800000 and 0x80001B	
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 10. Error message in the Altera Monitor Program that may be due to the SDRAM clock skew problem.

5 Using a Phase-Locked Loop

The clock skew depends on physical characteristics of the DE2 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. There exists a Quartus II Megafunction, called *ALTPLL*, which can be used to generate the desired circuit. The circuit can be created, by using the Quartus II MegaWizard Plug-In Manager, as follows:

1. Select Tools > MegaWizard Plug-In Manager. This leads to the window in Figure 11. Choose the action Create a new custom megafunction variation and click Next.

MegaWizard	Plug-In Manager [page 1]	×
×	The MegaWizard Plug-In Manager helps you create or modify design files that contain custom variations of megafunctions. Which action do you want to perform? © <u>Create a new custom megafunction variation</u> © <u>E</u> dit an existing custom megafunction variation © Copy an existing custom megafunction variation Copyright © 1991-2008 Altera Corporation	
	Cancel < <u>B</u> ack. <u>N</u> ext > Einis	h

Figure 11. The MegaWizard.

2. In the window in Figure 12, specify that Cyclone II is the device family used and that the circuit should be defined in VHDL. Also, specify that the generated output (VHDL) file should be called *sdram_pll.vhd*. From the list of megafunctions in the left box select I/O > ALTPLL. Click Next.

MegaWizard Plug-In Manager [page 2a]	
Which megafunction would you like to customize? Select a megafunction from the list below	Which device family will you be Cyclone II
Arithmetic Communications Solution Altracks Altracks Altracks Altracks Altracks Altracks AltrackTrl AltrackTrl	Which type of output file do you want to create? AHDL VHDL Verilog HDL What name do you want for the output file? Browse D:\DE2_sdram_tutorial\sdram_pil.vhd
ALTDQS ALTOSS ALTOSUF ALTIOBUF ALTIUVDS ALTNEMPHY ALTOCT ALTPLL_RECONFIG ALTPLL_RECONFIG ALTREMOTE_UPDATE MAXII oscillator	Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory, in the global user libraries specified in the User Libraries page of the Settings dialog box (Assignments menu). Your current user library directories are:

Figure 12. Select the megafunction and name the output file.

3. In Figure 13, specify that the frequency of the *inclock0* input is 50 MHz. Leave the other parameters as given by default. Click Next to reach the window in Figure 14.

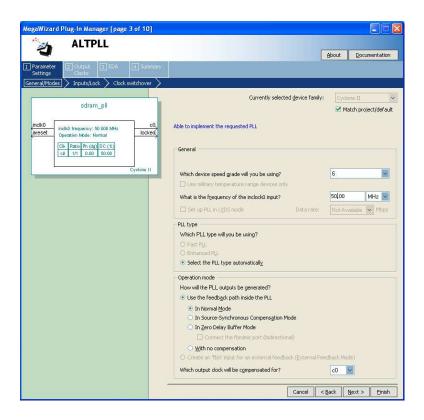


Figure 13. Define the clock frequency.

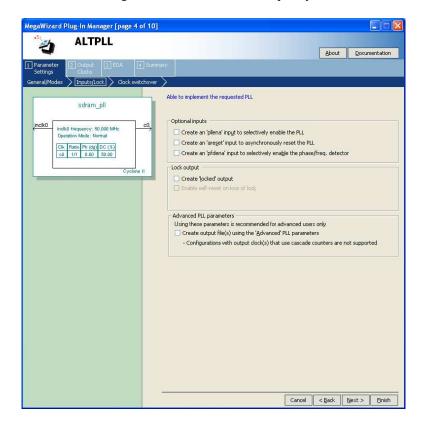


Figure 14. Remove unnecessary signals.

4. We are interested only in the input signal *inclock0* and the output signal *c0*. Remove the other two signals shown in the block diagram in the figure by de-selecting the optional input areset as well as the locked output, as indicated in the figure. Click Next on this page as well as on page 5, until you reach page 6 which is shown in Figure 15.

MegaWizard Plug-In Manager [page 6 o	f 10]		
ALTPLL		About	Documentation
Parameter Z Output Settings Clocks	Summary		
dkc0 > dkc1 > dkc2 >			
sdram_pil	c0 - Core/External Output Clock Able to implement the requested PLL		
inclk0 inclk0 frequency: 50.000 MHz	으. 🖸 Use this clock		
Operation Mode: Normal	Clock Tap Settings	Requested settings	Actual settings
Clk Ratio Ph (dg) DC (%) c0 1/1 -54.00 50.00	Enter output clock frequency:	50,00000000 MHz V	50.000000
and the setter setter set			
Cyclone II	Clock <u>multiplication</u> factor	1	1
	Clock division factor	1 × << Copy	1
	Clock phase shift	-3.00 ns 💌	-3.00
	Clock d <u>u</u> ty cycle (%)	50.00 😂	50.00
	More Details >>		
		Per Clock Feasib	
		c0 c1 c2	2
		Cancel < <u>B</u> ack	Next > Einish

Figure 15. Specify the phase shift.

- 5. The shifted clock signal is called c0. Specify that the output clock frequency is 50 MHz. Also, specify that a phase shift of -3 ns is required, as indicated in the figure. Click Next to reach the window in figure 16.
- 6. In order to ensure that the phase shift is exactly -3 ns, we will drive the original (non-shifted) clock through the PLL as well, but without any modifications. It will be called c1. Select Use this clock and specify that the output clock frequency is 50 MHz. Leave all the other settings unchanged and click Finish, which advances to page 10.
- 7. In the summary window in Figure 17 click Finish to complete the process. At this point, a box may pop up asking you to add the newly generated files to the project. In this case, select No, since this is not needed when using VHDL.

MegaWizard Plug-In Manager [page 7 o	of 10]		
ALTPLL Parameter 2 Output 3 EDA G Settings] Summary	About	
dkc0 $dkc1$ $dkc2$			
sdram_pll inclk0 Operation Mode: Normal Cik Ratio Phr (dg) DC (%) c0 17/1 - 34.00 50.00 c1 17/1 0.00 50.00 Cyclone	c1 - Core/External Output Clock Able to implement the requested PLL ✓ Use this clock Clock Tap Settings Enter output clock frequency: Enter output clock parameters: Clock <u>multiplication factor</u> Clock <u>glvision factor</u> Clock <u>glvision factor</u> Clock <u>phase shift</u>	Requested settings 50 MHz V 1 0 1 0 0.00 0 deg V	Actual settings 50.000000 1 1 0.00
	Clock dyty cycle (%)	50.00 CO CL C2	
		Cancel < Back	Next > Einish

Figure 16. Drive the original clock signal through the PLL.

gaWizard Plug-In Manager [page 10 of 10	<u>[</u>	
		About Documentation
Parameter 2 Output 3 EDA 4 Summ Settings Clocks	ary	
sdram_pll inclk0 frequency: 50.000 MHz Operation Mode: Normal Clik Ratio Ph (dg) DC (%)	automatically generated, and a rec Finish to generate the selected file subsequent MegaWizard Plug-In M The MegaWizard Plug-In Manager directory:	rate. A gray checkmark indicates a file that is d checkmark indicates an optional file. Click s. The state of each checkbox is maintained in anager sessions. creates the selected files in the following
c0 1/1 -54.00 50.00	D:\DE2_sdram_tutorial\	
	File ✓ sdram_pll.vhd ✓ sdram_pll.ppí sdram_pll.cmp sdram_pll.st sdram_pll.isst //d ✓ sdram_pll_waveforms.html isdram_pll_wave*.jpg	Description Variation file PinPlanner ports PPF file AHDL Include file VHDL component declaration file Quartus II symbol file Instantiation template file Sample waveforms in summary Sample waveform file(s)

Figure 17. The summary page.

The desired PLL circuit is now defined as a VHDL entity in the file *sdram_pll.vhd*, which is placed in the project directory. Figure 18 shows the entity ports, consisting of signals *inclk0*, *c0*, and *c1*.

🕸 sdram_pll	.vhd		
	42 43	ENTITY sdram_pl1 IS PORT	^
₩ 🎲 📅	44	■ (_
te te	45 46	<pre>inclk0 : IN STD_LOGIC := '0'; c0 : OUT STD LOGIC ;</pre>	
<u> 16</u> % %	47 48	c1 : OUT STD_LOGIC	
Z 0 🕺	49	END sdram_pll;	
2	50 51		
267 268 ab/	52 53	ARCHITECTURE SYN OF sdram_p11 IS	
	54	SIGNAL sub_wire0 : STD_LOGIC_VECTOR (5 DOWNTO 0);	
	55 56	SIGNAL sub_wire1 : STD_LOGIC ; SIGNAL sub_wire2 : STD_LOGIC ;	~
	<		>

Figure 18. The generated PLL entity.

Next, we have to fix the top-level VHDL entity, given in Figure 8, to include the PLL circuit. The desired code is shown in Figure 19. The PLL circuit connects the shifted clock output *c0* to the pin *DRAM_CLK*, and the unmodified clock signal *c1* to the clock signal required to drive the Nios II system.

- -- Implements a simple Nios II system for the DE2 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.
- -- SDRAM ports correspond to the signals in Figure 2; their names are those
- -- used in the DE2 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_CLK, DRAM_CKE : OUT STD_LOGIC; DRAM_ADDR : OUT STD_LOGIC_VECTOR(11 DOWNTO 0); DRAM_BA_1, DRAM_BA_0 : BUFFER STD_LOGIC; DRAM_CS_N, DRAM_CAS_N, DRAM_RAS_N, DRAM_WE_N : OUT STD_LOGIC; DRAM_DQ : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0); DRAM_UDQM, DRAM_LDQM : BUFFER STD_LOGIC);

END lights;

ARCHITECTURE Structure OF lights IS

COMPONENT nios_system

```
PORT ( clk : IN STD_LOGIC;
    reset_n : IN STD_LOGIC;
    out_port_from_the_LEDs : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    in_port_to_the_Switches : IN STD_LOGIC_VECTOR(7 DOWNTO 0)
    zs_addr_from_the_sdram : OUT STD_LOGIC_VECTOR(11 DOWNTO 0);
    zs_ba_from_the_sdram : BUFFER STD_LOGIC_VECTOR(1 DOWNTO 0);
    zs_cas_n_from_the_sdram : OUT STD_LOGIC;
    zs_cke_from_the_sdram : OUT STD_LOGIC;
    zs_cs_n_from_the_sdram : OUT STD_LOGIC;
    zs_dq_to_and_from_the_sdram : INOUT STD_LOGIC_VECTOR(15 DOWNTO 0);
    zs_ras_n_from_the_sdram : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    zs_ras_n_from_the_sdram : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    zs_ras_n_from_the_sdram : OUT STD_LOGIC];
    zs_we_n_from_the_sdram : OUT STD_LOGIC;
    zs_we_n_from_the_sdram : OUT STD_LOGIC;
    zs_we_n_from_the_sdram : OUT STD_LOGIC;
    zs_we_n_from_the_sdram : OUT STD_LOGIC );
    END COMPONENT;
```

...continued in Part b

Figure 19. Proper instantiation of the expanded Nios II system (Part a).

COMPONENT sdram_pll PORT (inclk0 : IN STD LOGIC; c0 : OUT STD_LOGIC); END COMPONENT SIGNAL BA : STD_LOGIC_VECTOR(1 DOWNTO 0); SIGNAL DQM : STD_LOGIC_VECTOR(1 DOWNTO 0); -- This signal is used to connect the unmodified clock signal c1 from the PLL to the -- NIOS II system SIGNAL pll c1 : STD LOGIC; BEGIN DRAM BA $1 \leq BA(1)$; $DRAM_BA_0 \le BA(0);$ DRAM UDQM $\leq =$ DQM(1); DRAM LDQM $\leq =$ DQM(0); -- Instantiate the Nios II system entity generated by the SOPC Builder. NiosII: nios system PORT MAP (pll c1, KEY(0), LEDG, SW, DRAM_ADDR, BA, DRAM_CAS_N, DRAM_CKE, DRAM_CS_N, DRAM_DQ, DQM, DRAM_RAS_N, DRAM_WE_N);

Instantiate the entity sdram_pll (inclk0, c0).
 neg_3ns: sdram_pll PORT MAP (CLOCK_50, DRAM_CLK, pll_c1);

END Structure;

Figure 19. Proper instantiation of the expanded Nios II system (Part b).

Compile the code and download the design into the Cyclone II FPGA on the DE2 board. Use the application program in Figure 9 to test the circuit.

Copyright ©2008 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, 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.