# Using the DDR3 SDRAM on Intel's DE5 Board with VHDL Designs

*For Quartus® Prime 18.1*

## 1 Introduction

This tutorial explains how DDR3 memory modules connected to Intel's DE5 Development and Education board can be used with a Nios® II system implemented by using the Intel® Platform Designer tool. The discussion is based on the assumption that the reader has access to a DE5 board and is familiar with the material in the tutorial *Introduction to the Intel Platform Designer Tool* and the tutorial *Intel FPGA Monitor Program Tutorial for Nios II*.

The screen captures in the tutorial were obtained using the Quartus® Prime version 18.1; if other versions of the software are used, some of the images may be slightly different.

**Contents**:

## 2 Background

The introductory tutorial *Introduction to the Intel Platform Designer Tool* explains how the memory in an 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 DE5 board contains two DDR3 SODIMM (Small outline dual inline memory modules) slots that can be used to expand the amount of memory available to the FPGA. To provide access to the DDR3 SODIMMs, the Platform Designer tool implements a *DDR3 SDRAM Controller with UniPHY* circuit that is compatible with both DDR3 and DDR3L memory modules. This circuit generates the signals needed to interface with the DDR3 SODIMMs. The DDR3 standard requires careful timing between the memory modules and the system, so the *DDR3 SDRAM Controller with UniPHY* circuit uses a reference clock signal to produce two clock signals: one for the system and one for the memory module.

## 3 Example Nios® II System

As an illustrative example, we will add the DDR3 SDRAM to the Nios II system described in the *Introduction to the Intel Platform Designer Tool* tutorial. Figure 1 gives the block diagram of our example system.

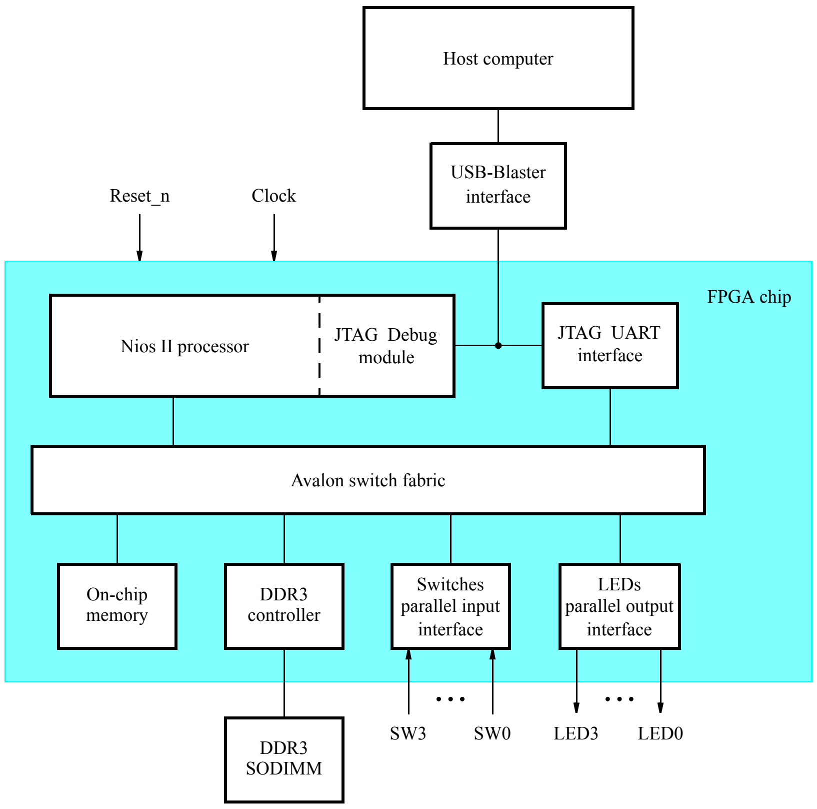


Figure 1. Example Nios II system implemented on the DE5 board.

The system realizes a trivial task. Four toggle switches on the DE5 board, $SW3-0$, are used to turn on or off the four green LEDs, $LED3-0$. 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 four-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 Platform Designer 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 LEDs accordingly is loaded into a memory block in the FPGA chip. In this tutorial, we will explain how DDR3 SODIMMs on the DE5 can be included in the system in Figure 1, so that our application program can be run from the DDR3 SDRAM rather than from the on-chip memory.

Doing this tutorial, the reader will learn about:

- Using the Platform Designer tool to include a DDR3 SDRAM Interface for a Nios II-based system
- Interfacing components clocked by different frequency signals on the DE5 board

## 4 The DDR3 SDRAM Interface

The signals needed to communicate with the DDR3 SODIMMs are shown in Figure 2. All of the signals can be provided by the DDR3 SDRAM Controller that can be generated by using the Platform Designer tool.

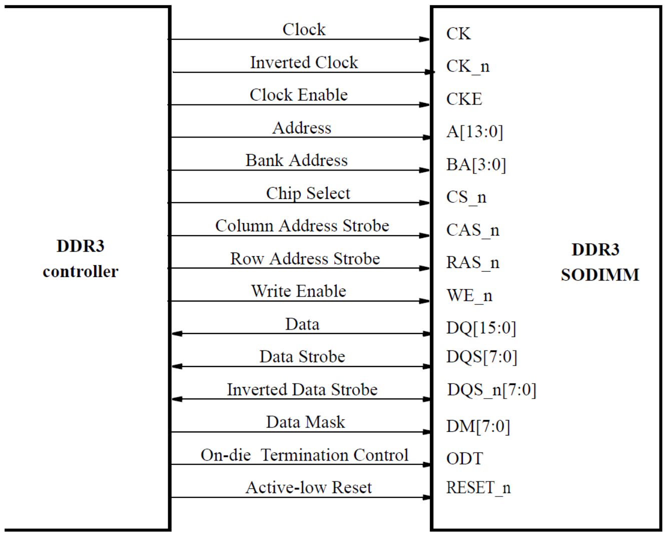
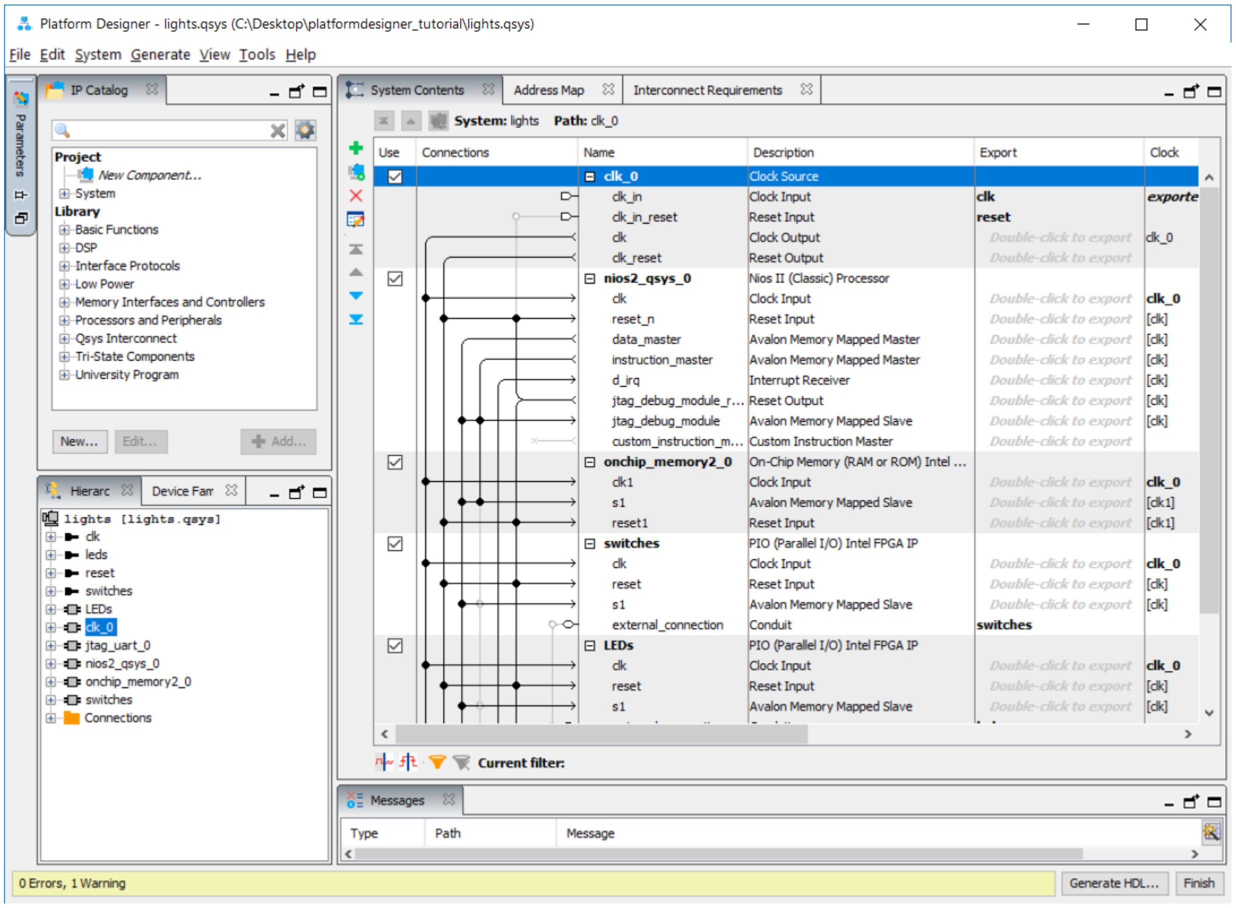


Figure 2. The DDR3 SDRAM signals.

## 5 Using the Platform Designer tool to Generate the Nios® II System

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



System: lights Path: clk_0

| Use | Connections | Name | Description | Export | Clock |
|---|---|---|---|---|---|
| ✓ | | clk_0 | Clock Source | | |
| | | clk_in | Clock Input | clk | exporte |
| | | clk_in_reset | Reset Input | reset | |
| | | clk | Clock Output | Double-click to export | clk_0 |
| | | clk_reset | Reset Output | Double-click to export | |
| ✓ | | nios2_qsys_0 | Nios II (Classic) Processor | | |
| | | clk | Clock Input | Double-click to export | clk_0 |
| | | reset_n | Reset Input | Double-click to export | [clk] |
| | | data_master | Avalon Memory Mapped Master | Double-click to export | [clk] |
| | | instruction_master | Avalon Memory Mapped Master | Double-click to export | [clk] |
| | | d_irq | Interrupt Receiver | Double-click to export | [clk] |
| | | jtag_debug_module_r... | Reset Output | Double-click to export | [clk] |
| | | jtag_debug_module | Avalon Memory Mapped Slave | Double-click to export | [clk] |
| | | custom_instruction_m... | Custom Instruction Master | Double-click to export | |
| ✓ | | onchip_memory2_0 | On-Chip Memory (RAM or ROM) Intel ... | | |
| | | clk1 | Clock Input | Double-click to export | clk_0 |
| | | s1 | Avalon Memory Mapped Slave | Double-click to export | [clk1] |
| | | reset1 | Reset Input | Double-click to export | [clk1] |
| ✓ | | switches | PIO (Parallel I/O) Intel FPGA IP | | |
| | | clk | Clock Input | Double-click to export | clk_0 |
| | | reset | Reset Input | Double-click to export | [clk] |
| | | s1 | Avalon Memory Mapped Slave | Double-click to export | [clk] |
| | | external_connection | Conduit | switches | |
| ✓ | | LEDs | PIO (Parallel I/O) Intel FPGA IP | | |
| | | clk | Clock Input | Double-click to export | clk_0 |
| | | reset | Reset Input | Double-click to export | [clk] |
| | | s1 | Avalon Memory Mapped Slave | Double-click to export | [clk] |

0 Errors, 1 Warning

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

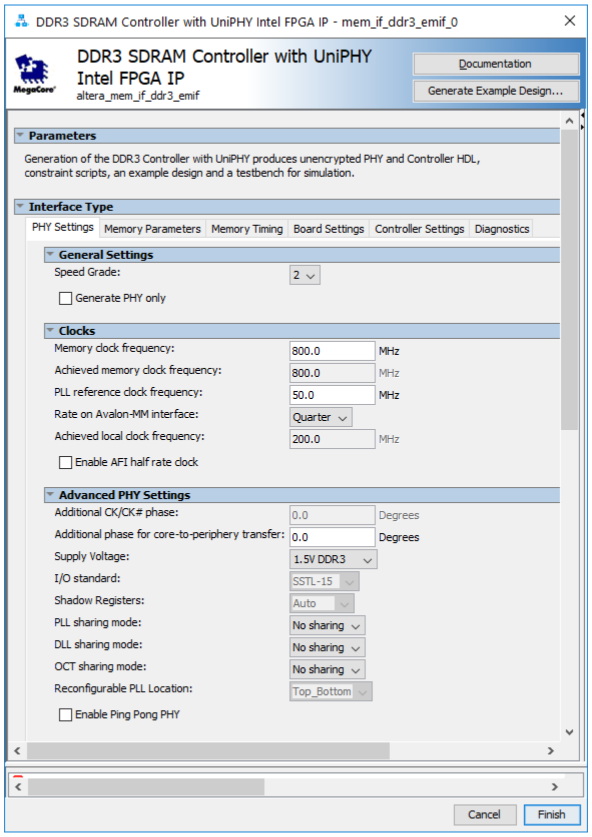
If you saved the *lights* project, then open this project in the Quartus Prime software and then open the Platform Designer tool. Otherwise, you need to create and implement the project, as explained in the introductory tutorial, to obtain the system shown in the figure.

The DDR3 controller requires you to specify the parameters of your particular DDR3 SODIMM for it to function correctly. A list of necessary parameters are given in Table 1. The tutorial provides the parameters for the K4B2G0846C-HCK0 SODIMM manufactured by Samsung Electronics and supplied with the DE5 at the time of writing. If you use a different memory module, you will have search your module's datasheet for the parameters listed in Table 1.

| **Memory Parameter** | **K4B2G0846C-HCK0 Timing Values** |
|---|---|
| Memory device speed | 800 MHz |
| CAS Latency | 11 cycles |
| Row address width | 14 bits |
| Column address width | 10 bits |
| Bank address width | 3 bits |
| Address and control setup to CK clock rise [**tIS**] | 170 ps |
| Address and control hold after CK clock rise [**tIH**] | 120 ps |
| Data setup to clock (DQS) rise [**tDS**] | 10 ps |
| Data hold after clock (DQS) rise [**tDH**] | 45 ps |
| DQS, DQS to DQ skew, per group, per access [**tDQSQ**] | 100 ps |
| DQ output hold time from DQS, DQS [**tQH**] | 0.38 cycles |
| DQS output access time from CK, CK [**tDQSCK**] | 255 ps |
| First latching edge of DQS to associated clock edge [**tDQSS**] | 0.27 cycles |
| DQS Differential High Pulse Width [**tQSH**] | 0.4 cycles |
| DQS falling edge hold time from CK [**tDSH**] | 0.18 cycles |
| DQS falling edge to CK setup time [**tDSS**] | 0.18 cycles |
| Memory initialization time at power-up [**tINIT**] | 500 us |
| Load mode register command period [**tMRD**] | 4 cycles |
| Active to precharge time [**tRAS**] | 35 ns |
| Active to read or write time [**tRCD**] | 13.75 ns |
| Precharge command period [**tRP**] | 13.75 ns |
| Refresh command interval [**tREFI**] | 7.8 us |
| Auto-refresh command interval [**tRFC**] | 160 ns |
| Write recovery time [**tWR**] | 15 ns |
| Write to read period [**tWTR**] | 4 cycles |
| Four active window time [**tFAW**] | 30 ns |
| RAS to RAS delay time [**tRRD**] | 6 ns |
| Read to precharge time [**tRTP**] | 7.5 ns |

Table 1. Parameters for the K4B2G0846C-HCK0 SODIMM supplied with the DE5

To add the DDR3 controller, in the window of Figure 3 select Memory Interfaces and Controllers > Memory Interfaces with UniPHY > DDR3 SDRAM Controller with UniPHY and click Add. A window depicted in Figure 4 appears.



Generation of the DDR3 Controller with UniPHY produces unencrypted PHY and Controller HDL, constraint scripts, an example design and a testbench for simulation.

**General Settings**

| | |
|---|---|
| Speed Grade: | 2 |
| ☐ Generate PHY only | |

**Clocks**

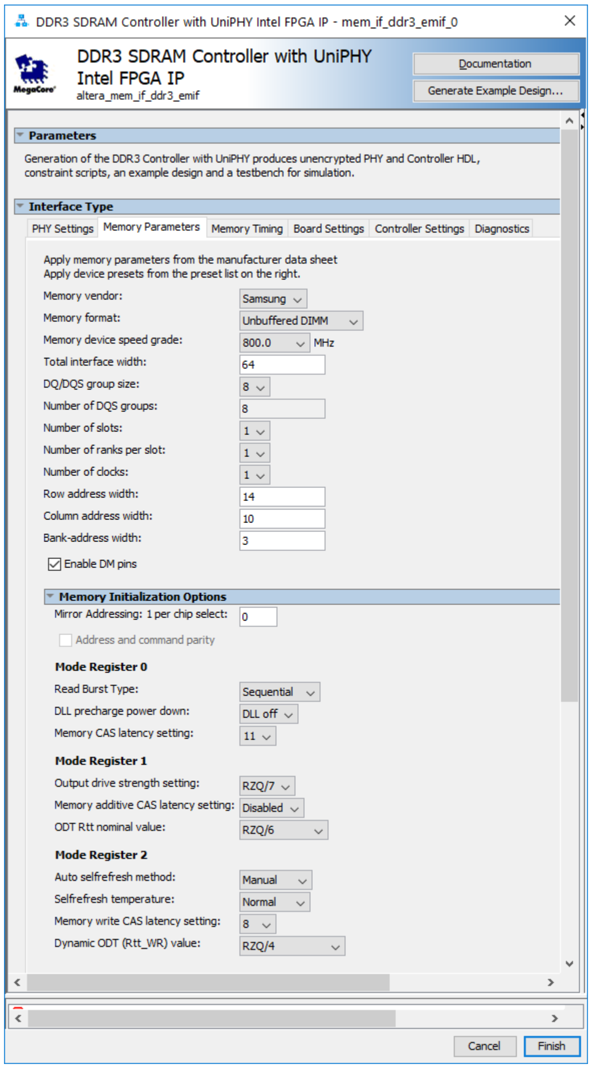
| | |
|---|---|
| Memory clock frequency: | 800.0 MHz |
| Achieved memory clock frequency: | 800.0 MHz |
| PLL reference clock frequency: | 50.0 MHz |
| Rate on Avalon-MM interface: | Quarter |
| Achieved local clock frequency: | 200.0 MHz |
| ☐ Enable AFI half rate clock | |

**Advanced PHY Settings**

| | |
|---|---|
| Additional CK/CK# phase: | 0.0 Degrees |
| Additional phase for core-to-periphery transfer: | 0.0 Degrees |
| Supply Voltage: | 1.5V DDR3 |
| I/O standard: | SSTL-15 |
| Shadow Registers: | Auto |
| PLL sharing mode: | No sharing |
| DLL sharing mode: | No sharing |
| OCT sharing mode: | No sharing |
| Reconfigurable PLL Location: | Top_Bottom |
| ☐ Enable Ping Pong PHY | |

Figure 4. Add the DDR3 Controller.

Set the Speed Grade to 2, the Memory clock frequency parameter to 800.0 MHz, the PLL reference clock frequency to 50.0 MHz, Rate on Avalon-MM interface to Quarter, and leave other settings in PHY Settings as default. Click Memory Parameters to show the window in Figure 5


DDR3 SDRAM Controller with UniPHY Intel FPGA IP - mem_if_ddr3_emif_0

DDR3 SDRAM Controller with UniPHY Intel FPGA IP
altera_mem_if_ddr3_emif

Documentation
Generate Example Design...

**Parameters**

Generation of the DDR3 Controller with UniPHY produces unencrypted PHY and Controller HDL, constraint scripts, an example design and a testbench for simulation.

**Interface Type**

PHY Settings | Memory Parameters | Memory Timing | Board Settings | Controller Settings | Diagnostics

Apply memory parameters from the manufacturer data sheet
Apply device presets from the preset list on the right.

| | |
|---|---|
| Memory vendor: | Samsung |
| Memory format: | Unbuffered DIMM |
| Memory device speed grade: | 800.0 MHz |
| Total interface width: | 64 |
| DQ/DQS group size: | 8 |
| Number of DQS groups: | 8 |
| Number of slots: | 1 |
| Number of ranks per slot: | 1 |
| Number of clocks: | 1 |
| Row address width: | 14 |
| Column address width: | 10 |
| Bank-address width: | 3 |

☑ Enable DM pins

**Memory Initialization Options**

| | |
|---|---|
| Mirror Addressing: 1 per chip select: | 0 |

☐ Address and command parity

**Mode Register 0**

| | |
|---|---|
| Read Burst Type: | Sequential |
| DLL precharge power down: | DLL off |
| Memory CAS latency setting: | 11 |

**Mode Register 1**

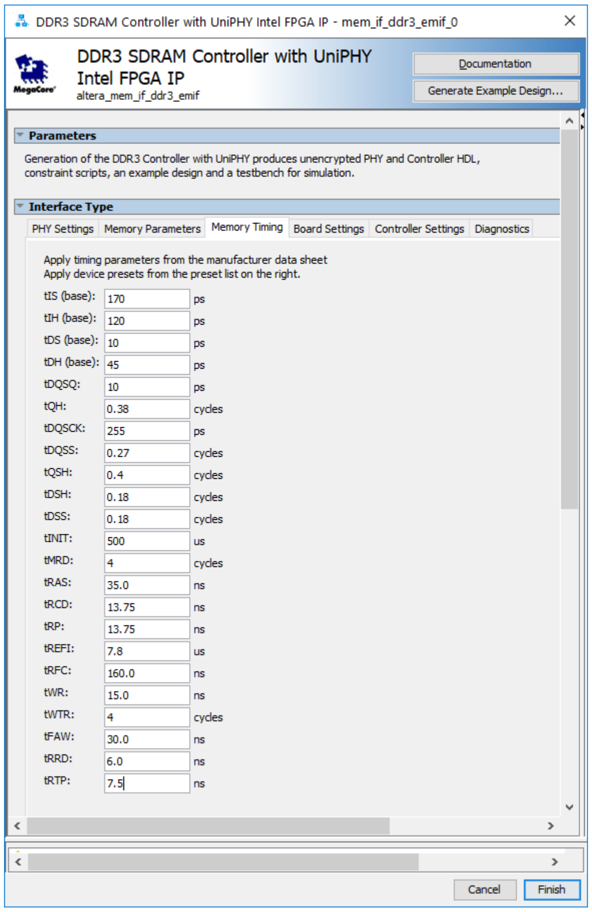
| | |
|---|---|
| Output drive strength setting: | RZQ/7 |
| Memory additive CAS latency setting: | Disabled |
| ODT Rtt nominal value: | RZQ/6 |

**Mode Register 2**

| | |
|---|---|
| Auto selfrefresh method: | Manual |
| Selfrefresh temperature: | Normal |
| Memory write CAS latency setting: | 8 |
| Dynamic ODT (Rtt_WR) value: | RZQ/4 |

Figure 5. DDR3 Controller Memory Parameters Window.

Set the Memory vendor to Samsung, the Memory format to Unbuffered DIMM, the Memory device speed grade to 800.0 MHz, theTotal interface width to 64, the Row address width to 14, and the Column address width to 10. In the Memory Initialization Options, set the Memory CAS latency setting to 11, Output drive strength setting to RZQ/7, ODT Rtt nominal value to RZQ/6, Memory write CAS latency setting to 8, and Dynamic ODT (Rtt_WR) value to RZQ/4. The other settings should be left at their default values. Click Memory Timing to get to the window shown in Figure 6.


DDR3 SDRAM Controller with UniPHY Intel FPGA IP - mem_if_ddr3_emif_0

DDR3 SDRAM Controller with UniPHY Intel FPGA IP
altera_mem_if_ddr3_emif

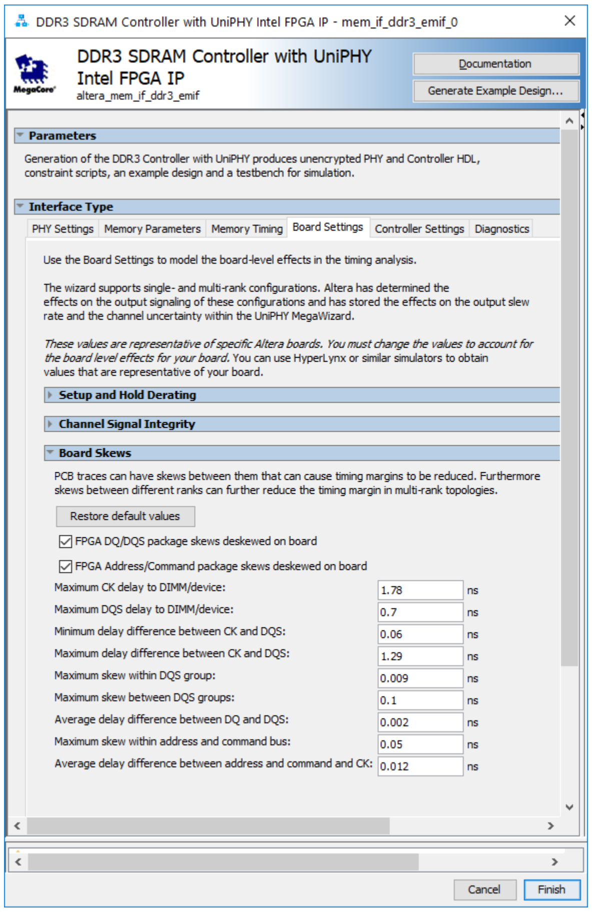
Generation of the DDR3 Controller with UniPHY produces unencrypted PHY and Controller HDL, constraint scripts, an example design and a testbench for simulation.

Apply timing parameters from the manufacturer data sheet
Apply device presets from the preset list on the right.

| | | |
|---|---|---|
| tIS (base): | 170 | ps |
| tIH (base): | 120 | ps |
| tDS (base): | 10 | ps |
| tDH (base): | 45 | ps |
| tDQSQ: | 10 | ps |
| tQH: | 0.38 | cycles |
| tDQSCK: | 255 | ps |
| tDQSS: | 0.27 | cycles |
| tQSH: | 0.4 | cycles |
| tDSH: | 0.18 | cycles |
| tDSS: | 0.18 | cycles |
| tINIT: | 500 | us |
| tMRD: | 4 | cycles |
| tRAS: | 35.0 | ns |
| tRCD: | 13.75 | ns |
| tRP: | 13.75 | ns |
| tREFI: | 7.8 | us |
| tRFC: | 160.0 | ns |
| tWR: | 15.0 | ns |
| tWTR: | 4 | cycles |
| tFAW: | 30.0 | ns |
| tRRD: | 6.0 | ns |
| tRTP: | 7.5 | ns |

Figure 6. DDR3 Controller Memory Timing Window.

Set the timing parameters to the values shown in Table 1 then click on Board Settings to get to the window shown in Figure 7.


DDR3 SDRAM Controller with UniPHY Intel FPGA IP - mem_if_ddr3_emif_0

DDR3 SDRAM Controller with UniPHY Intel FPGA IP
altera_mem_if_ddr3_emif

Documentation
Generate Example Design...

**Parameters**

Generation of the DDR3 Controller with UniPHY produces unencrypted PHY and Controller HDL, constraint scripts, an example design and a testbench for simulation.

**Interface Type**

PHY Settings | Memory Parameters | Memory Timing | Board Settings | Controller Settings | Diagnostics

Use the Board Settings to model the board-level effects in the timing analysis.

The wizard supports single- and multi-rank configurations. Altera has determined the effects on the output signaling of these configurations and has stored the effects on the output slew rate and the channel uncertainty within the UniPHY MegaWizard.

*These values are representative of specific Altera boards. You must change the values to account for the board level effects for your board.* You can use HyperLynx or similar simulators to obtain values that are representative of your board.

**Setup and Hold Derating**

**Channel Signal Integrity**

**Board Skews**

PCB traces can have skews between them that can cause timing margins to be reduced. Furthermore skews between different ranks can further reduce the timing margin in multi-rank topologies.

[✓] FPGA DQ/DQS package skews deskewed on board

[✓] FPGA Address/Command package skews deskewed on board

| | | |
|---|---|---|
| Maximum CK delay to DIMM/device: | 1.78 | ns |
| Maximum DQS delay to DIMM/device: | 0.7 | ns |
| Minimum delay difference between CK and DQS: | 0.06 | ns |
| Maximum delay difference between CK and DQS: | 1.29 | ns |
| Maximum skew within DQS group: | 0.009 | ns |
| Maximum skew between DQS groups: | 0.1 | ns |
| Average delay difference between DQ and DQS: | 0.002 | ns |
| Maximum skew within address and command bus: | 0.05 | ns |
| Average delay difference between address and command and CK: | 0.012 | ns |

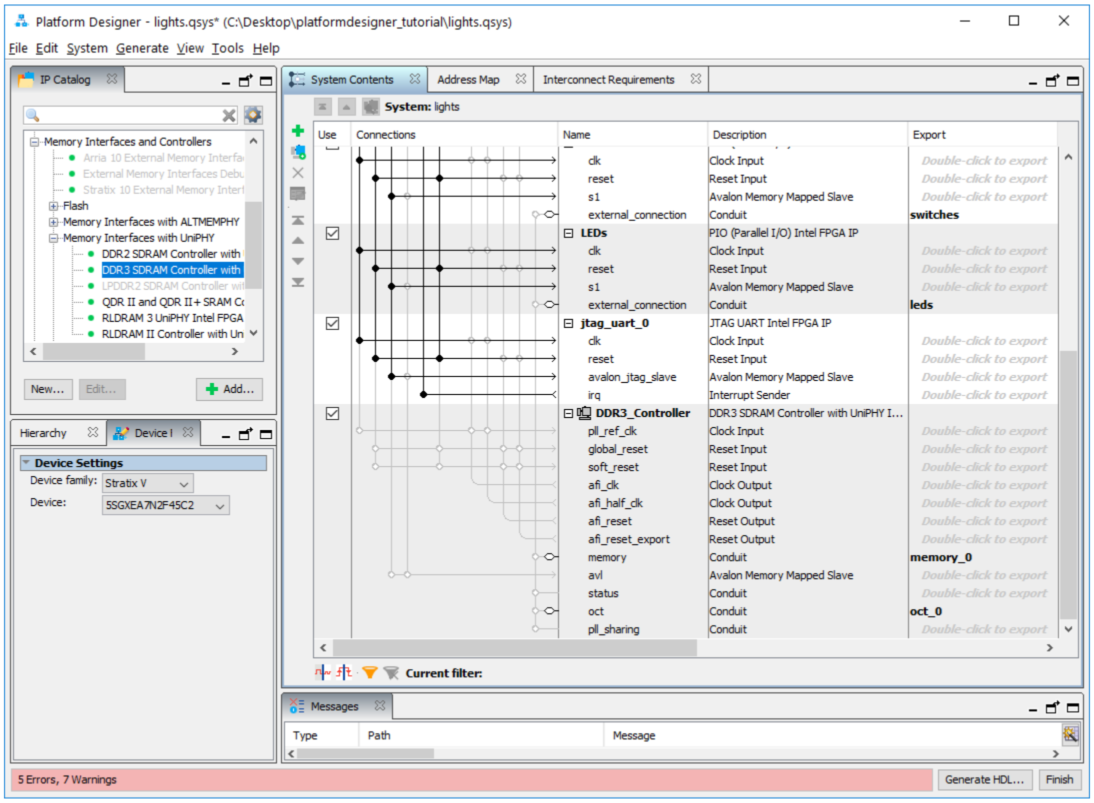
Figure 7. DDR3 Controller Board Settings Window.

In the Board Skews sub-menu, check FPGA DQ/DQS package skews deskewed on board and FPGA Address/Command package skews deskewed on board then set the other board skew parameters according to the values in Table 2.

| Board Skew Parameter | Value (ns) |
|---|---|
| Maximum CK delay to DIMM/device | 1.78 |
| Maximum DQS delay to DIMM/device | 0.7 |
| Minimum delay difference between CK and DQS | 0.06 |
| Maximum delay difference between CK and DQS | 1.29 |
| Maximum skew within DQS group | 0.009 |
| Maximum skew between DQS groups | 0.1 |
| Average delay difference between DQ and DQS | 0.002 |
| Maximum skew within address and command bus | 0.05 |
| Average delay difference between address and command and CK | 0.012 |

Table 2. Board Skew Parameters

If you wish to save the settings of this controller to save time when making another system, press New in the lower-right of the window shown in Figure 7. This will open up a dialog that allows you to give your preset a name and then save it. Now in Figure 8 press Finish to add the component to Platform Designer. Right-click on the component and rename it to *DDR3_Controller*. You should now have the system shown in Figure 8.


Platform Designer - lights.qsys* (C:\Desktop\platformdesigner_tutorial\lights.qsys)
File Edit System Generate View Tools Help
IP Catalog
Memory Interfaces and Controllers
Arria 10 External Memory Interfa
External Memory Interfaces Debu
Stratix 10 External Memory Inter
Flash
Memory Interfaces with ALTMEMPHY
Memory Interfaces with UniPHY
DDR2 SDRAM Controller with
DDR3 SDRAM Controller with
LPDDR2 SDRAM Controller wi
QDR II and QDR II+ SRAM Co
RLDRAM 3 UniPHY Intel FPGA
RLDRAM II Controller with Un
New...
Edit...
Add...
Hierarchy
Device I
System Contents
Address Map
Interconnect Requirements
Current filter:
Messages
Type
Path
Message
Generate HDL...
Finish

Device Settings
Device family: Stratix V
Device: 5SGXEA7N2F45C2

System: lights

| Use | Name | Description | Export |
|---|---|---|---|
| | clk | Clock Input | Double-click to export |
| | reset | Reset Input | Double-click to export |
| | s1 | Avalon Memory Mapped Slave | Double-click to export |
| | external_connection | Conduit | **switches** |
| ☑ | **LEDs** | PIO (Parallel I/O) Intel FPGA IP | |
| | clk | Clock Input | Double-click to export |
| | reset | Reset Input | Double-click to export |
| | s1 | Avalon Memory Mapped Slave | Double-click to export |
| | external_connection | Conduit | **leds** |
| ☑ | **jtag_uart_0** | JTAG UART Intel FPGA IP | |
| | clk | Clock Input | Double-click to export |
| | reset | Reset Input | Double-click to export |
| | avalon_jtag_slave | Avalon Memory Mapped Slave | Double-click to export |
| | irq | Interrupt Sender | Double-click to export |
| ☑ | **DDR3_Controller** | DDR3 SDRAM Controller with UniPHY I... | |
| | pll_ref_clk | Clock Input | Double-click to export |
| | global_reset | Reset Input | Double-click to export |
| | soft_reset | Reset Input | Double-click to export |
| | afi_clk | Clock Output | Double-click to export |
| | afi_half_clk | Clock Output | Double-click to export |
| | afi_reset | Reset Output | Double-click to export |
| | afi_reset_export | Reset Output | Double-click to export |
| | memory | Conduit | **memory_0** |
| | avl | Avalon Memory Mapped Slave | Double-click to export |
| | status | Conduit | Double-click to export |
| | oct | Conduit | **oct_0** |
| | pll_sharing | Conduit | Double-click to export |

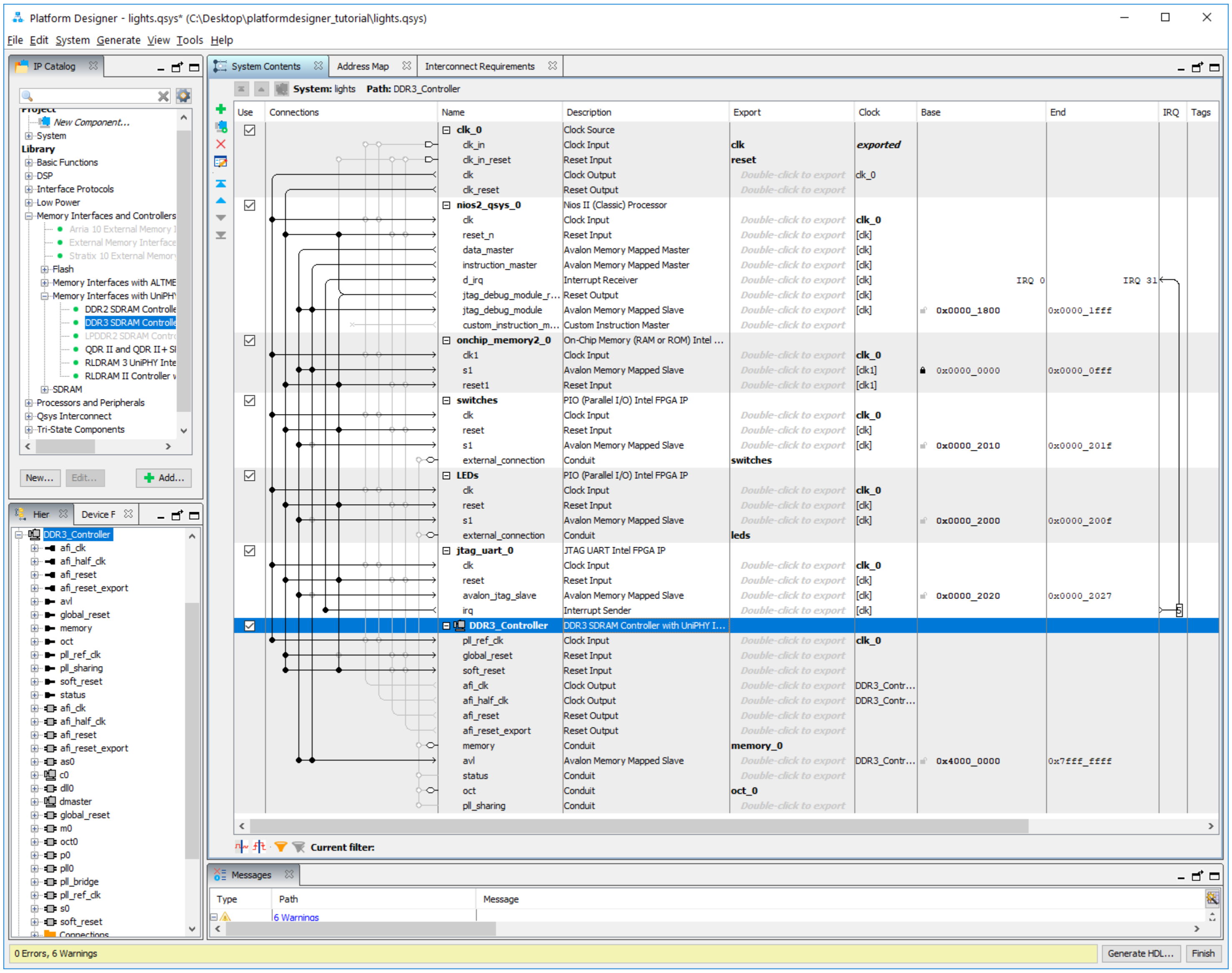
5 Errors, 7 Warnings

Figure 8. Platform Designer system with the new DDR3 Controller.

Make the following connections:

- Connect the *pll_ref_clk* port of *DDR3_Controller* to the *clk* port of *clk_0*.
- Connect *global_reset* and *soft_reset* ports of *DDR3_Controller* to *clk_reset* port of *clk_0*.
- Connect the *jtag_debug_module_reset* port of the Nios II processor to the *soft_reset* port of *DDR3_Controller*.
- Connect the clock input of the *nios2_processor*, *onchip_memory*, *switches*, *LEDs*, and *jtag_uart* to the clock output, *afi_clk* of *DDR3_Controller*.
- Connect the *data_master* and *instruction_master* ports of the NIOS II processor to the *avl* port of *DDR3_Controller*.

Double click on the *avl* base address of the *DDR3_Controller* and set it to **0x4000_0000**. Your system should now look similar to the one in Figure 9. Right-click on the Nios II processor component to get to the window in Figure 10. Set the Reset Vector memory and Exception Vector memory to DDR3_Controller.avl and press Finish to return to the window in Figure 9. Click on Generate HDL... > Generate to generate your system and then close Platform Designer.



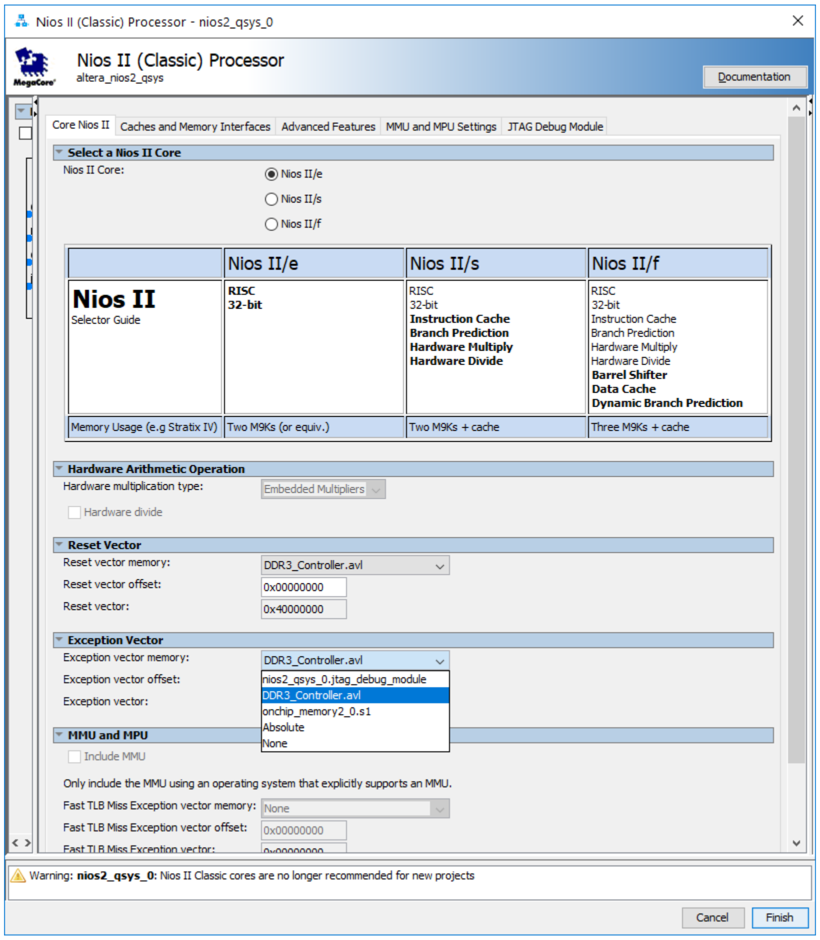
**System:** lights **Path:** DDR3_Controller

| Use | Connections | Name | Description | Export | Clock | Base | End | IRQ | Tags |
|---|---|---|---|---|---|---|---|---|---|
| ☑ | | ⊟ **clk_0** | Clock Source | | | | | | |
| | | clk_in | Clock Input | **clk** | *exported* | | | | |
| | | clk_in_reset | Reset Input | **reset** | | | | | |
| | | clk | Clock Output | *Double-click to export* | clk_0 | | | | |
| | | clk_reset | Reset Output | *Double-click to export* | | | | | |
| ☑ | | ⊟ **nios2_qsys_0** | Nios II (Classic) Processor | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset_n | Reset Input | *Double-click to export* | [clk] | | | | |
| | | data_master | Avalon Memory Mapped Master | *Double-click to export* | [clk] | | | | |
| | | instruction_master | Avalon Memory Mapped Master | *Double-click to export* | [clk] | | | | |
| | | d_irq | Interrupt Receiver | *Double-click to export* | [clk] | IRQ 0 | IRQ 31 | | |
| | | jtag_debug_module_r... | Reset Output | *Double-click to export* | [clk] | | | | |
| | | jtag_debug_module | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0000_1800 | 0x0000_1fff | | |
| | | custom_instruction_m... | Custom Instruction Master | *Double-click to export* | | | | | |
| ☑ | | ⊟ **onchip_memory2_0** | On-Chip Memory (RAM or ROM) Intel ... | | | | | | |
| | | clk1 | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk1] | 0x0000_0000 | 0x0000_0fff | | |
| | | reset1 | Reset Input | *Double-click to export* | [clk1] | | | | |
| ☑ | | ⊟ **switches** | PIO (Parallel I/O) Intel FPGA IP | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0000_2010 | 0x0000_201f | | |
| | | external_connection | Conduit | **switches** | | | | | |
| ☑ | | ⊟ **LEDs** | PIO (Parallel I/O) Intel FPGA IP | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | s1 | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0000_2000 | 0x0000_200f | | |
| | | external_connection | Conduit | **leds** | | | | | |
| ☑ | | ⊟ **jtag_uart_0** | JTAG UART Intel FPGA IP | | | | | | |
| | | clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | reset | Reset Input | *Double-click to export* | [clk] | | | | |
| | | avalon_jtag_slave | Avalon Memory Mapped Slave | *Double-click to export* | [clk] | 0x0000_2020 | 0x0000_2027 | | |
| | | irq | Interrupt Sender | *Double-click to export* | [clk] | | | 5 | |
| ☑ | | ⊟ **DDR3_Controller** | DDR3 SDRAM Controller with UniPHY I... | | | | | | |
| | | pll_ref_clk | Clock Input | *Double-click to export* | **clk_0** | | | | |
| | | global_reset | Reset Input | *Double-click to export* | | | | | |
| | | soft_reset | Reset Input | *Double-click to export* | | | | | |
| | | afi_clk | Clock Output | *Double-click to export* | DDR3_Contr... | | | | |
| | | afi_half_clk | Clock Output | *Double-click to export* | DDR3_Contr... | | | | |
| | | afi_reset | Reset Output | *Double-click to export* | | | | | |
| | | afi_reset_export | Reset Output | *Double-click to export* | | | | | |
| | | memory | Conduit | **memory_0** | | | | | |
| | | avl | Avalon Memory Mapped Slave | *Double-click to export* | DDR3_Contr... | 0x4000_0000 | 0x7fff_ffff | | |
| | | status | Conduit | *Double-click to export* | | | | | |
| | | oct | Conduit | **oct_0** | | | | | |
| | | pll_sharing | Conduit | *Double-click to export* | | | | | |

| Type | Path | Message |
|---|---|---|
| ⚠ | 6 Warnings | |

0 Errors, 6 Warnings

Figure 9. Final Platform Designer system with DDR3 Controller.



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: DDR3_Controller.avl

Reset vector offset: 0x00000000

Reset vector: 0x40000000

Exception Vector

Exception vector memory: DDR3_Controller.avl

Exception vector offset:

Exception vector:

nios2_qsys_0.jtag_debug_module
DDR3_Controller.avl
onchip_memory2_0.s1
Absolute
None

MMU and MPU

Include MMU

Only include the MMU using an operating system that explicitly supports an MMU.

Fast TLB Miss Exception vector memory: None

Fast TLB Miss Exception vector offset: 0x00000000

Fast TLB Miss Exception vector: 0x00000000

Warning: **nios2_qsys_0:** Nios II Classic cores are no longer recommended for new projects

Figure 10. Changing the reset and exception vectors of the NIOS II processor.

## 6 Integration of the Nios® II System into the Quartus® Prime 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 Platform Designer Tool*. The entity is named *lights*, because this is the name of the top-level design entity in our Quartus Prime project.

The new top-level entity is presented in Figure 11. 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 LEDs, *LED*, as used in our original design. They also use the pin names *DDR3A_A*, *DDR3A_BA*, *DDR3A_CAS_n*, *DDR3A_CK*, *DDR3A_CKE*, *DDR3A_CK_n*, *DDR3A_CS_n*, *DDR3A_DM*, *DDR3A_DQ*, *DDR3A_DQS*, *DDR3A_DQS_n*, *DDR3A_ODT*, *DDR3A_RAS_n*, *DDR3A_RESET_n*, *DDR3A_WE_n*, and *RZQ_4*, which correspond to the DDR3 SDRAM signals indicated in Figure 2. The names are the ones cited in the DE5 User Manual and are included in the file called *DE5.qsf*. The QSF file is included in the design files for this tutorial.

```
// Implements the augmented Nios II system for the DE5 board.
// Inputs:    SW3-0 are parallel port inputs to the Nios II system.
//            BUTTON0 is the active-low system reset.
// Outputs:   LEDR3-0 are parallel port outputs from the Nios II system.
//            DDR3 ports correspond to the signals in Figure 2; their names
//            are those used in the DE5 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);
        BUTTON : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
        OSC_50_B7D, RZQ_4 : IN STD_LOGIC;
        LED : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        DDR3A_A : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
        DDR3A_BA : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
        DDR3A_CAS_n, DDR3A_RAS_n, DDR3A_WE_n : OUT STD_LOGIC_VECTOR (0 DOWNTO 0);
        DDR3A_RESET_n : OUT STD_LOGIC;
        DDR3A_CK, DDR3A_CK_n : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
        DDR3A_CKE, DDR3A_CS_n, DDR3A_ODT : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
        DDR3A_DM : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        DDR3A_DQ : INOUT STD_LOGIC_VECTOR (63 DOWNTO 0);
        DDR3A_DQS, DDR3A_DQS_n : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END lights;
ARCHITECTURE Structure OF lights IS
    COMPONENT nios_system
        PORT (
            clk_clk : IN STD_LOGIC;
            reset_reset_n : IN STD_LOGIC;
            switches_export: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
            leds_export : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
            memory_mem_a : OUT   std_logic_vector(13 DOWNTO 0);
            memory_mem_ba : OUT   std_logic_vector(2 DOWNTO 0);
            memory_mem_ck : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_ck_n : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_cke : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_cs_n : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_dm : OUT   std_logic_vector(7 DOWNTO 0);
            memory_mem_ras_n : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_cas_n : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_we_n : OUT   std_logic_vector(0 DOWNTO 0);
            memory_mem_reset_n : OUT   std_logic;
            memory_mem_dq : INOUT std_logic_vector(63 DOWNTO 0);
            memory_mem_dqs : INOUT std_logic_vector(7 DOWNTO 0);
            memory_mem_dqs_n : INOUT std_logic_vector(7 DOWNTO 0);
            memory_mem_odt : OUT   std_logic_vector(0 DOWNTO 0);
            oct_rzqin : IN std_logic);
    END COMPONENT;
```

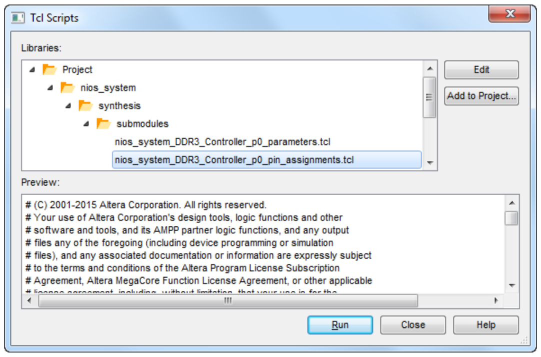
Figure 11. The top-level entity that instantiates the expanded Nios II system. (Part *a*)

```
BEGIN
-- Instantiate the Nios II system entity generated by the Qsys tool.
    NiosII: nios_system
    PORT MAP (
        clk_clk             => OSC_50_B7D,
        reset_reset_n       => BUTTON(0),
        switches_export     => SW,
        leds_export     => LED,
        memory_mem_a        => DDR3A_A,
        memory_mem_ba       => DDR3A_BA,
        memory_mem_ck       => DDR3A_CK,
        memory_mem_ck_n     => DDR3A_CK_n,
        memory_mem_cke      => DDR3A_CKE,
        memory_mem_cs_n     => DDR3A_CS_n,
        memory_mem_dm       => DDR3A_DM,
        memory_mem_ras_n    => DDR3A_RAS_n,
        memory_mem_cas_n    => DDR3A_CAS_n,
        memory_mem_we_n     => DDR3A_WE_n,
        memory_mem_reset_n  => DDR3A_RESET_n,
        memory_mem_dq       => DDR3A_DQ,
        memory_mem_dqs      => DDR3A_DQS,
        memory_mem_dqs_n    => DDR3A_DQS_n,
        memory_mem_odt      => DDR3A_ODT,
        oct_rzqin           => RZQ_4);
END Structure;
```

Figure 11. The top-level entity that instantiates the expanded Nios II system. (Part *b*).

Perform the following:

- Enter the code in Figure 11 into a file called *lights.vhd*. Add this file and the *nios_system.qip* file produced by the Platform Designer tool to your Quartus Prime project.
- Import the pin assignments from the QSF file included in the design files for this tutorial.
- Perform analysis and synthesis of the design by clicking Processing > Start > Analysis and Synthesis
- Click Tools > Tcl Scripts... to open the window in Figure 12. Select and run the script Project > nios_system > synthesis > submodules > nios_system_DDR3_Controller_p0_pin_assignments.tcl; the script is required to correctly set the differential pins needed by the DDR3 SDRAM Interface.
- Compile the project.
- Use the Intel FPGA Monitor Program, which is described in the tutorial *Intel FPGA Monitor Program Tutorial for Nios II* to download and test the system on the DE5 board. Use the assembly program from the tutorial *Intel FPGA Monitor Program Tutorial for Nios II* to test the system; it has been reproduced for you in Figure 13.



nios_system_DDR3_Controller_p0_parameters.tcl
nios_system_DDR3_Controller_p0_pin_assignments.tcl

```
# (C) 2001-2015 Altera Corporation. All rights reserved.
# Your use of Altera Corporation's design tools, logic functions and other
# software and tools, and its AMPP partner logic functions, and any output
# files any of the foregoing (including device programming or simulation
# files), and any associated documentation or information are expressly subject
# to the terms and conditions of the Altera Program License Subscription
# Agreement, Altera MegaCore Function License Agreement, or other applicable
```

Figure 12. TCL Scripts window in Quartus Prime.

If successful, the lights on the DE5 board will respond to the operation of the toggle switches.

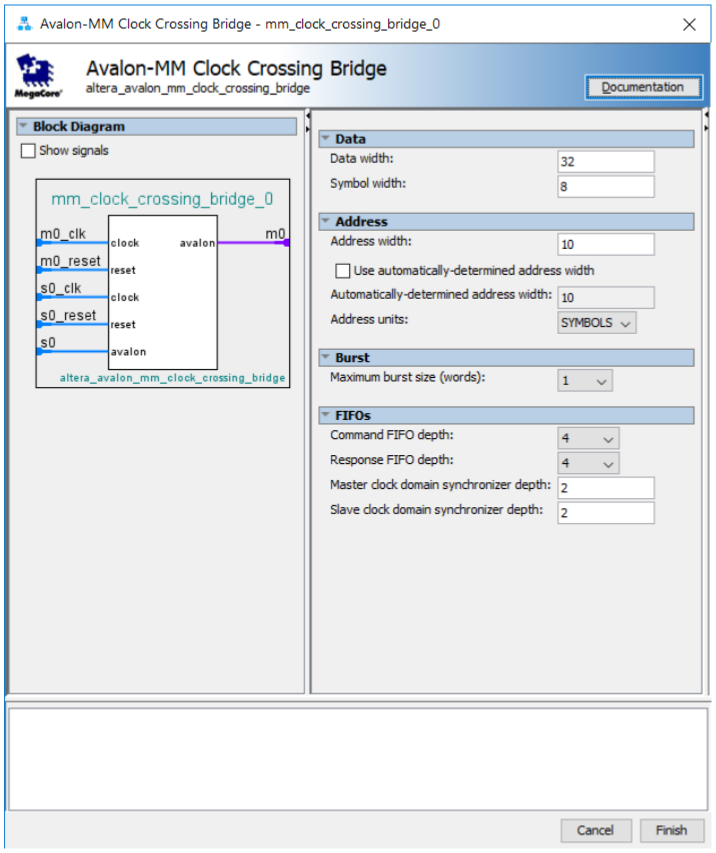
```
.equ      Switches, 0x00002010
.equ      LEDs, 0x00002000
.global   _start
_start:
          movia    r2, Switches
          movia    r3, LEDs
loop:     ldbio    r4, 0(r2)
          stbio    r4, 0(r3)
          br       loop
```

Figure 13. Assembly language code to control the lights.

## 7 Using the Clock Crossing Bridge IP Core

A clock crossing bridge allows components clocked by different frequency clock signals to interface and work with each other. This allows you to have low-speed and high-speed components in the same system without compromising the performance of your high-speed components. We will now modify our Nios system so that the human interface components (LEDs and switches) are run by a 50 MHz clock and the other components are run by the clock generated by the DDR3 SDRAM controller component.

Add the clock crossing bridge component to your Platform Designer system by selection Basic Functions > Bridges and Adaptors > Memory Mapped > Avalon-MM Clock Crossing Bridge. The window in Figure 14 will appear. Accept the default settings and press Finish. Right-click on the component and rename it *Clock_Crossing_Bridge*.


Avalon-MM Clock Crossing Bridge - mm_clock_crossing_bridge_0
Avalon-MM Clock Crossing Bridge
altera_avalon_mm_clock_crossing_bridge
Documentation
Block Diagram
Show signals
mm_clock_crossing_bridge_0
m0_clk
m0_reset
s0_clk
s0_reset
s0
clock
reset
clock
reset
avalon
avalon
m0
altera_avalon_mm_clock_crossing_bridge
Cancel
Finish

**Data**

| | |
|---|---|
| Data width: | 32 |
| Symbol width: | 8 |

**Address**

| | |
|---|---|
| Address width: | 10 |
| ☐ Use automatically-determined address width | |
| Automatically-determined address width: | 10 |
| Address units: | SYMBOLS |

**Burst**

| | |
|---|---|
| Maximum burst size (words): | 1 |

**FIFOs**

| | |
|---|---|
| Command FIFO depth: | 4 |
| Response FIFO depth: | 4 |
| Master clock domain synchronizer depth: | 2 |
| Slave clock domain synchronizer depth: | 2 |

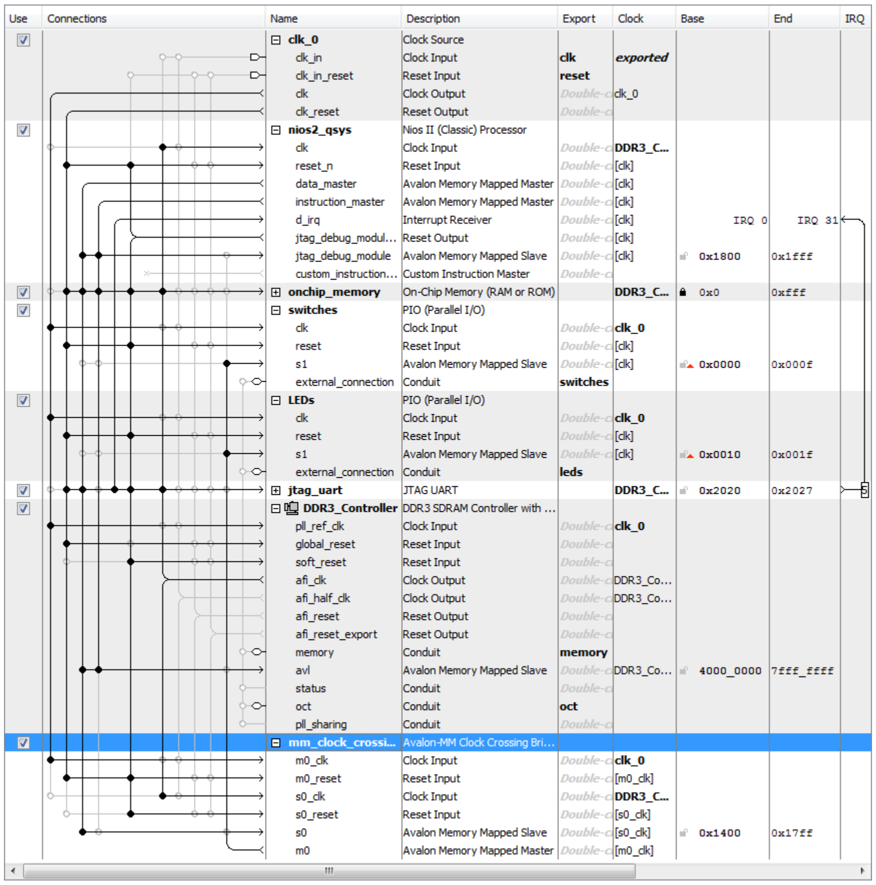
Figure 14

Make the following changes to your Nios system:

- Remove the connections from the *data_master* port of *nios2_processor* and the *s1* ports of *switches* and *LEDs*.
- Connect the *m0* port of *Clock_Crossing_Bridge* to the *s1* ports of *switches* and *LEDs*.
- Connect the *data_master* port of *nios2_processor* to the *s0* port of *Clock_Crossing_Bridge*.
- Connect the *clk* port of *clk_0* to the *s0_clk* port of *Clock_Crossing_Bridge* and the *clk* ports of *switches* and *LEDs*.
- Connect the *afi_clk* port of *DDR3_Controller* to the *m0_clk* port of *Clock_Crossing_Bridge*.
- Connect the *jtag_debug_module_reset* of *nios2_processor* to the *m0_reset* and *s0_reset* ports of *Clock_Crossing_Bridge*.
- Connect the *clk_reset* port of *clk_0* to the *m0_reset* port of *Clock_Crossing_Bridge*.

Double-click on the base address of *Clock_Crossing_Bridge* and set it to `0x1400`. Similarly, set the base address of *switches* to `0x0000` and *LEDs* to `0x0010`. The Nios processor will access these components at the address:

bridge base address + component base address. For *switches* this will be address **0x1400** and for *LEDs* this will be **0x1410**. The complete system is shown in Figure 15. Regenerate the system then recompile the top-level VHDL entity. Finally use the updated assembly program given in Figure 16 to test your system with the Intel FPGA Monitor Program.



| 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-c | clk_0 | | | |
| | | clk_reset | Reset Output | Double-c | | | | |
| ✓ | | ⊟ **nios2_qsys** | Nios II (Classic) Processor | | | | | |
| | | clk | Clock Input | Double-c | **DDR3_C...** | | | |
| | | reset_n | Reset Input | Double-c | [clk] | | | |
| | | data_master | Avalon Memory Mapped Master | Double-c | [clk] | | | |
| | | instruction_master | Avalon Memory Mapped Master | Double-c | [clk] | | | |
| | | d_irq | Interrupt Receiver | Double-c | [clk] | IRQ 0 | IRQ 31 | |
| | | jtag_debug_modul... | Reset Output | Double-c | [clk] | | | |
| | | jtag_debug_module | Avalon Memory Mapped Slave | Double-c | [clk] | 0x1800 | 0x1fff | |
| | | custom_instruction... | Custom Instruction Master | Double-c | | | | |
| ✓ | | ⊞ **onchip_memory** | On-Chip Memory (RAM or ROM) | | **DDR3_C...** | 0x0 | 0xfff | |
| ✓ | | ⊟ **switches** | PIO (Parallel I/O) | | | | | |
| | | clk | Clock Input | Double-c | **clk_0** | | | |
| | | reset | Reset Input | Double-c | [clk] | | | |
| | | s1 | Avalon Memory Mapped Slave | Double-c | [clk] | 0x0000 | 0x000f | |
| | | external_connection | Conduit | **switches** | | | | |
| ✓ | | ⊟ **LEDs** | PIO (Parallel I/O) | | | | | |
| | | clk | Clock Input | Double-c | **clk_0** | | | |
| | | reset | Reset Input | Double-c | [clk] | | | |
| | | s1 | Avalon Memory Mapped Slave | Double-c | [clk] | 0x0010 | 0x001f | |
| | | external_connection | Conduit | **leds** | | | | |
| ✓ | | ⊞ **jtag_uart** | JTAG UART | | **DDR3_C...** | 0x2020 | 0x2027 | 5 |
| ✓ | | ⊟ **DDR3_Controller** | DDR3 SDRAM Controller with ... | | | | | |
| | | pll_ref_clk | Clock Input | Double-c | **clk_0** | | | |
| | | global_reset | Reset Input | Double-c | | | | |
| | | soft_reset | Reset Input | Double-c | | | | |
| | | afi_clk | Clock Output | Double-c | DDR3_Co... | | | |
| | | afi_half_clk | Clock Output | Double-c | DDR3_Co... | | | |
| | | afi_reset | Reset Output | Double-c | | | | |
| | | afi_reset_export | Reset Output | Double-c | | | | |
| | | memory | Conduit | **memory** | | | | |
| | | avl | Avalon Memory Mapped Slave | Double-c | DDR3_Co... | 4000_0000 | 7fff_ffff | |
| | | status | Conduit | Double-c | | | | |
| | | oct | Conduit | **oct** | | | | |
| | | pll_sharing | Conduit | Double-c | | | | |
| ✓ | | ⊟ **mm_clock_crossi...** | Avalon-MM Clock Crossing Bri... | | | | | |
| | | m0_clk | Clock Input | Double-c | **clk_0** | | | |
| | | m0_reset | Reset Input | Double-c | [m0_clk] | | | |
| | | s0_clk | Clock Input | Double-c | **DDR3_C...** | | | |
| | | s0_reset | Reset Input | Double-c | [s0_clk] | | | |
| | | s0 | Avalon Memory Mapped Slave | Double-c | [s0_clk] | 0x1400 | 0x17ff | |
| | | m0 | Avalon Memory Mapped Master | Double-c | [m0_clk] | | | |

Figure 15. The final Nios II system.

```
.equ      Switches, 0x00001400
.equ      LEDs, 0x00001410
.global   _start
_start:
          movia     r2, Switches
          movia     r3, LEDs
loop:     ldbio     r4, 0(r2)
          stbio     r4, 0(r3)
          br        loop
```

Figure 16. New Assembly language code to control the lights.