Using the DDR2 SDRAM on Intel's DE4 Board with VHDL Designs

For Quartus® Prime 18.1

1 Introduction

This tutorial explains how DDR2 memory modules connected to Intel's DE4 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 DE4 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:

- Example Nios II System
- The DDR2 SDRAM Interface
- Using the Platform Designer tool to Generate the Nios II System
- Integration of the Nios II System into the Quartus Prime Project
- Using the Clock Crossing Bridge IP Core

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 DE4 board contains two DDR2 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 DDR2 SODIMMs, the Platform Designer tool implements a *DDR2 SDRAM Controller with UniPHY* circuit that generates the signals needed to interface with DDR2 SODIMMs. The DDR2 standard requires careful timing between the memory modules and the system, so the *DDR2 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 DDR2 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.

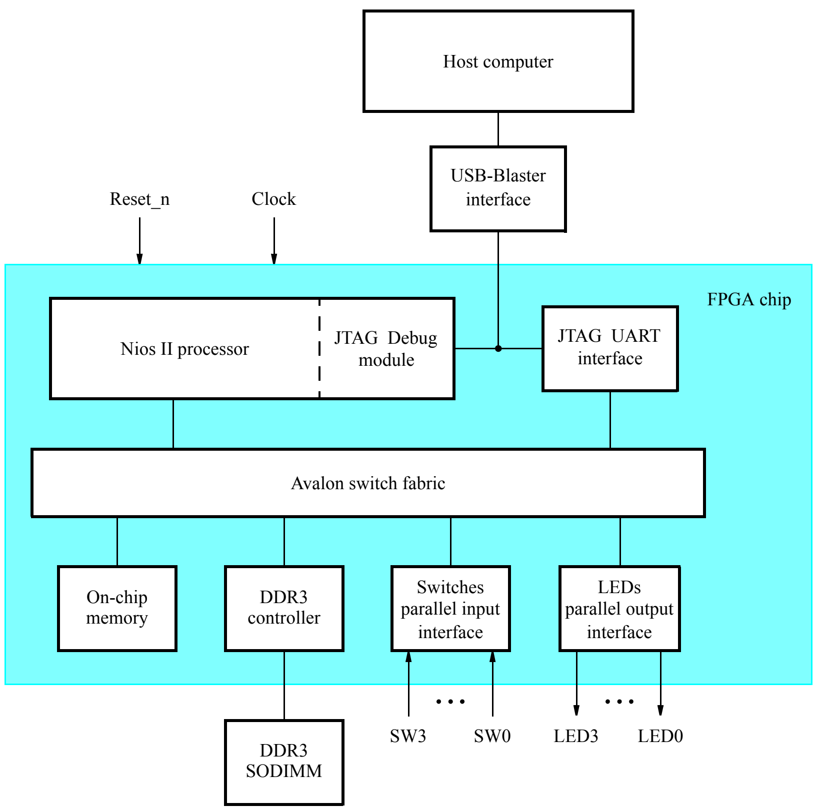

Figure 1. Example Nios II system implemented on the DE4 board.

The system realizes a trivial task. Four toggle switches on the DE4 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 DDR2 SODIMMs on the DE4 can be included in the system in Figure 1, so that our application program can be run from the DDR2 SDRAM rather than from the on-chip memory.

Doing this tutorial, the reader will learn about:

- Using the Platform Designer tool to include a DDR2 SDRAM Interface for a Nios II-based system
- Interfacing components clocked by different frequency signals on the DE4 board

4 The DDR2 SDRAM Interface

The signals needed to communicate with the DDR2 SODIMMs are shown in Figure 2. All of the signals can be provided by the DDR2 SDRAM Controller that can be generated by using the Platform Designer tool.

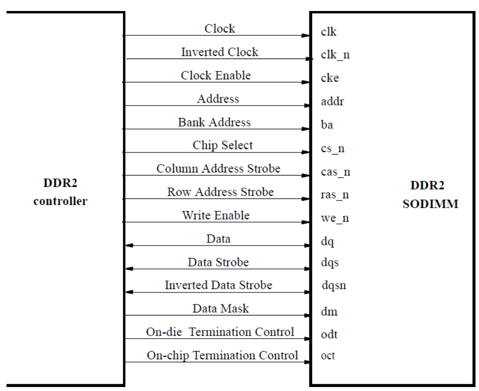

Figure 2. The DDR2 SDRAM signals.

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.

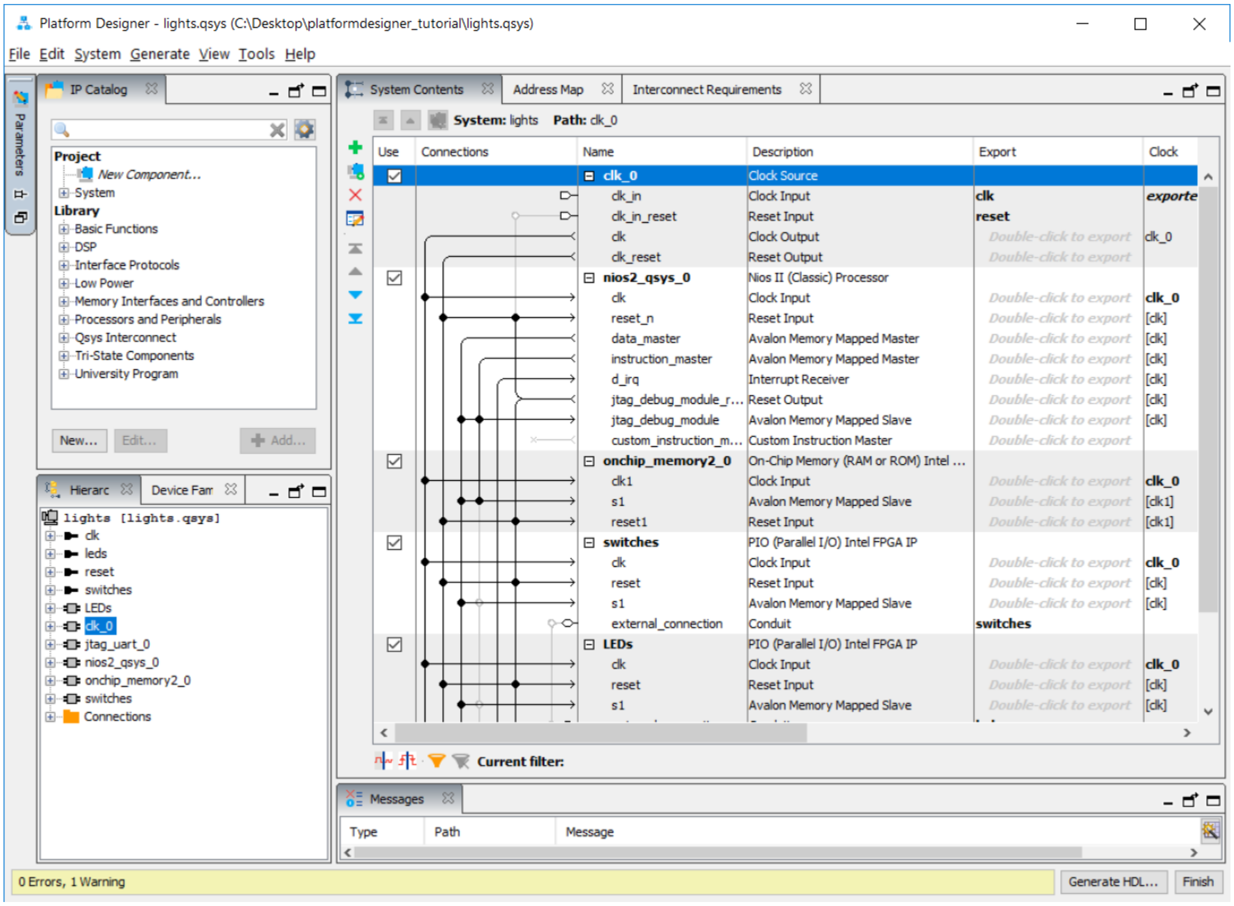

Figure 3. The Nios II system defined in the introductory tutorial.

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 DDR2 controller requires you to specify the parameters of your particular DDR2 SODIMM for it to function correctly. A list of necessary parameters are given in Table 1. The tutorial provides the parameters for the D2SH28081XH25AA SODIMM manufactured by DSL Memory Specialties and supplied with the DE4 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	400 MHz
CAS Latency	6 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**]	375 ps
Address and control hold after CK clock rise [**tIH**]	375 ps
Data setup to clock (DQS) rise [**tDS**]	250 ps
Data hold after clock (DQS) rise [**tDH**]	250 ps
DQS, DQS to DQ skew, per group, per access [**tDQSQ**]	200 ps
DQ output hold time from DQS, DQS [**tQHS**]	300 ps
DQS output access time from CK, CK [**tDQSCK**]	350 ps
First latching edge of DQS to associated clock edge [**tDQSS**]	0.25 cycles
DQS Differential High Pulse Width [**tDQSH**]	0.35 cycles
DQS falling edge hold time from CK [**tDSH**]	0.2 cycles
DQS falling edge to CK setup time [**tDSS**]	0.2 cycles
Memory initialization time at power-up [**tINIT**]	200 us
Load mode register command period [**tMRD**]	5 cycles
Active to precharge time [**tRAS**]	40.0 ns
Active to read or write time [**tRCD**]	15.0 ns
Precharge command period [**tRP**]	15.0 ns
Refresh command interval [**tREFI**]	7.8 us
Auto-refresh command interval [**tRFC**]	127.5 ns
Write recovery time [**tWR**]	15.0 ns
Write to read period [**tWTR**]	3 cycles
Four active window time [**tFAW**]	37.5 ns
RAS to RAS delay time [**tRRD**]	7.5 ns
Read to precharge time [**tRTP**]	7.5 ns

Table 1. Parameters for the D2SH28081XH25AA SODIMM supplied with the DE4

To add the DDR2 controller, in the window of Figure 3 select Memory Interfaces and Controllers > Memory Interfaces with UniPHY > DDR2 SDRAM Controller with UniPHY and click Add. A window depicted in Figure 4 appears.

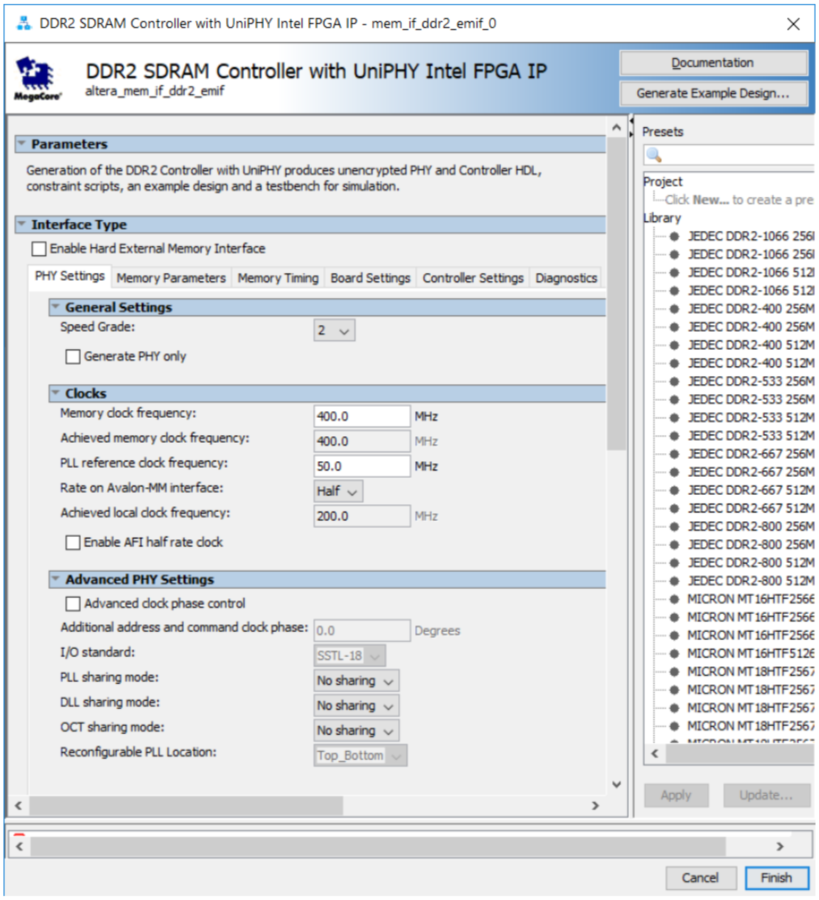

Figure 4. Add the DDR2 Controller.

Set the Speed Grade to 2, the Memory clock frequency parameter to 400.0 MHz, the PLL reference clock frequency to 50.0 MHz, Rate on Avalon-MM interface to Half, and leave other settings in PHY Settings as default. Click Memory Parameters to show the window in Figure 5

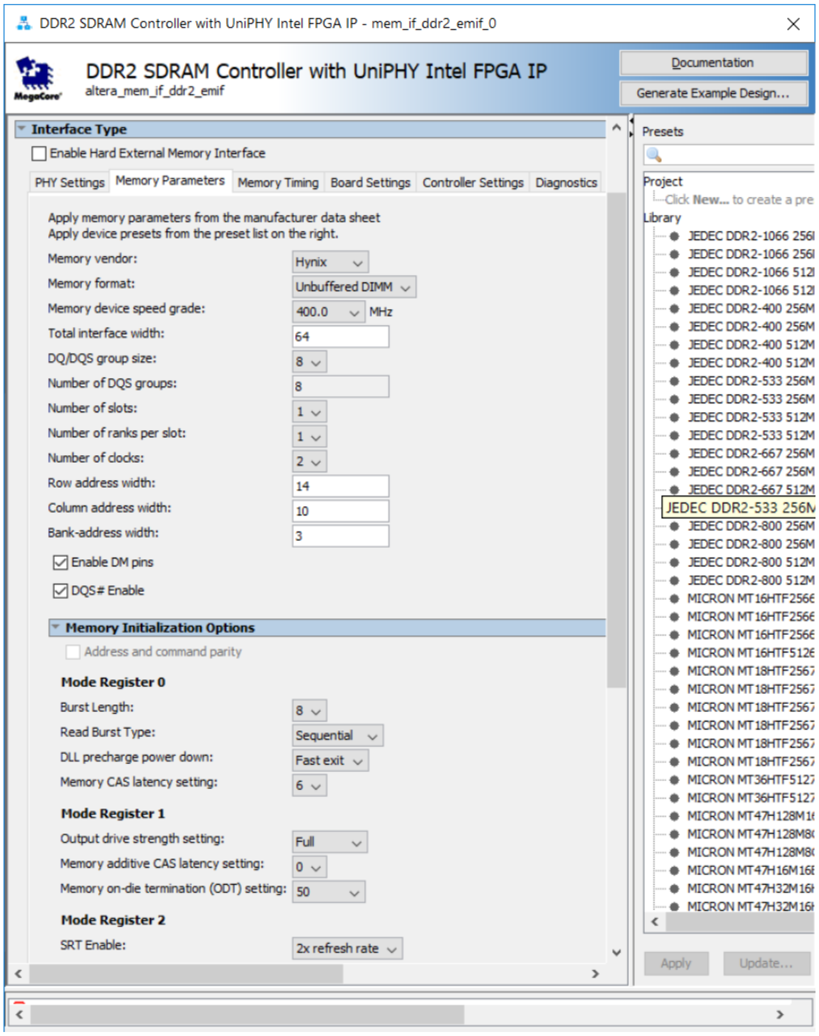

Figure 5. DDR2 Controller Memory Parameters Window.

Set the Memory vendor to Hynix, the Memory format to Unbuffered DIMM, the Memory device speed grade to 400.0 MHz, theTotal interface width to 64, the Number of Clocks to 2, 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 6, Memory on-die termination (ODT) setting to 50. The other settings should be left at their default values. Click Memory Timing to get to the window shown in Figure 6.

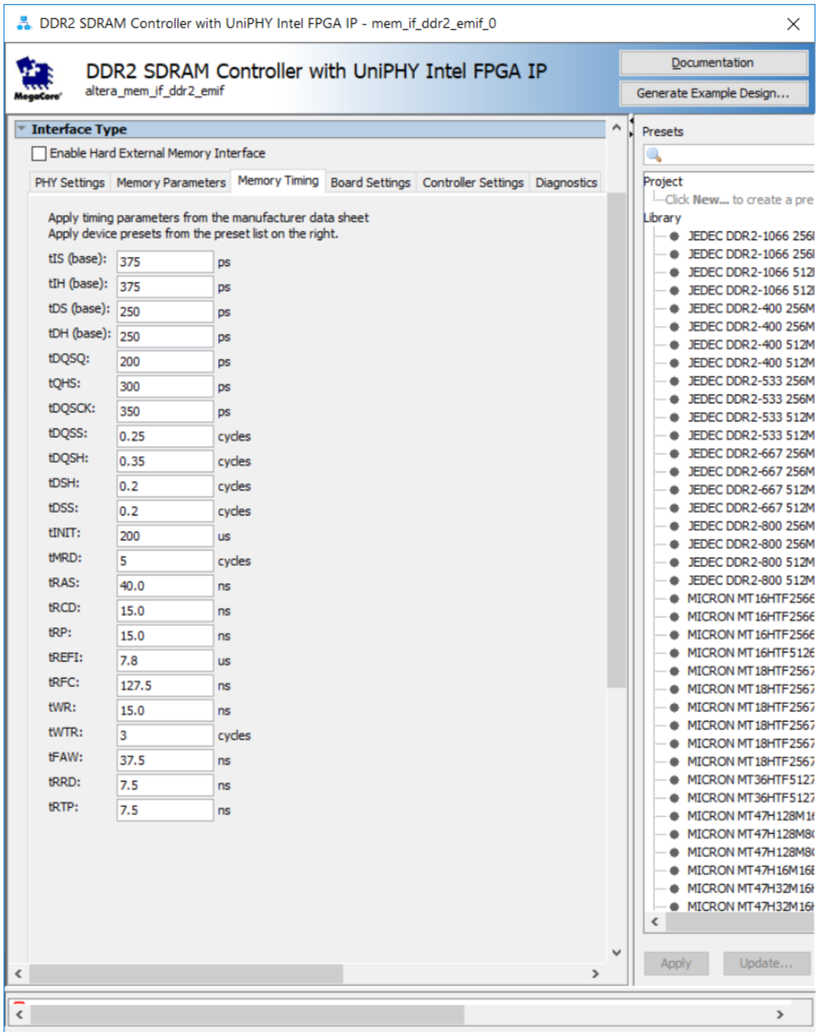

Figure 6. DDR2 Controller Memory Timing Window.

Set the timing parameters to the values shown in Table 1 then click on Finish.

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 7 press Finish to add the component to Platform Designer. Right-click on the component and rename it to *DDR2_Controller*. You should now have the system shown in Figure 7.

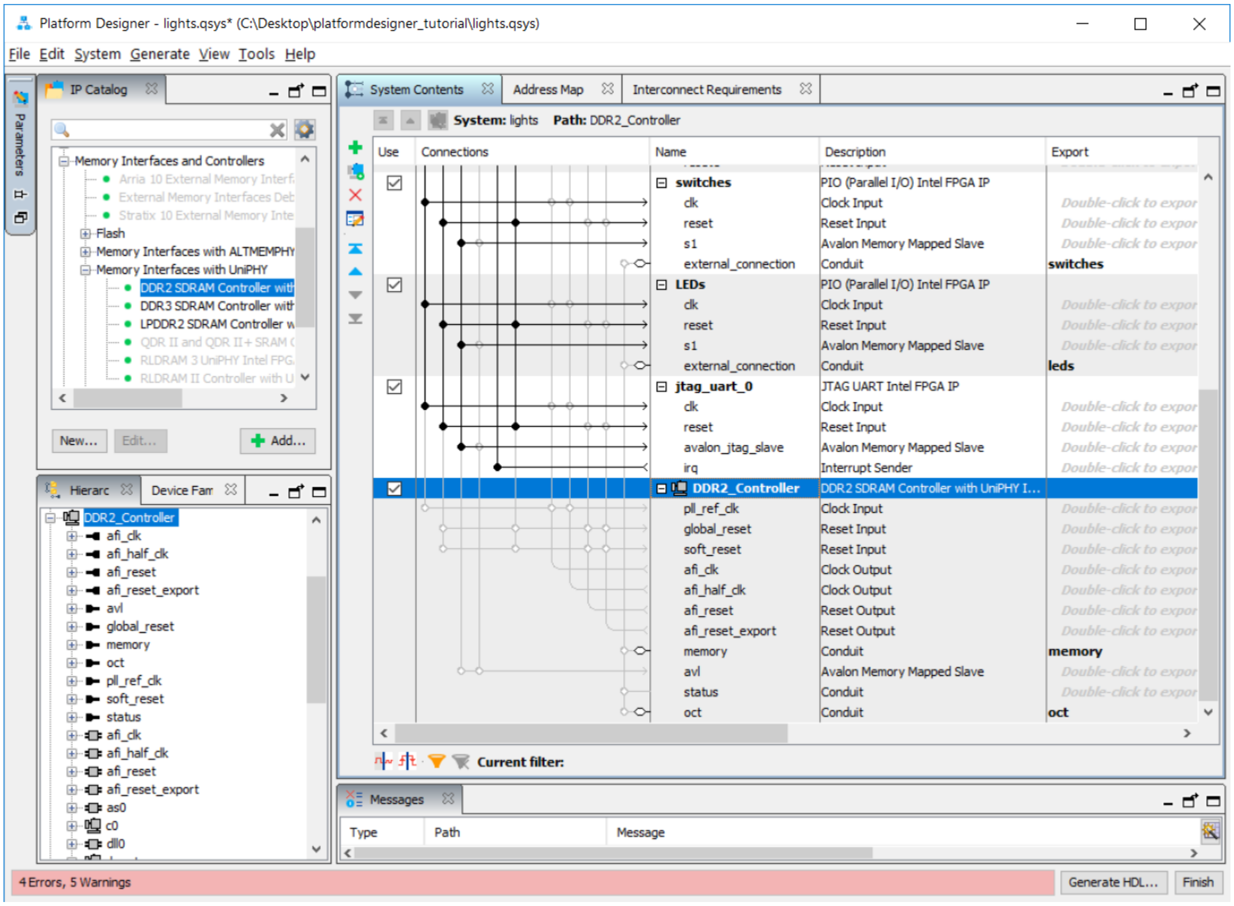

Figure 7. Platform Designer system with the new DDR2 Controller.

Make the following connections:

- Connect the *pll_ref_clk* port of *DDR2_Controller* to the *clk* port of *clk_0*.
- Connect *global_reset* and *soft_reset* ports of *DDR2_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 *DDR2_Controller*.
- Connect the clock input of the *nios2_processor*, *onchip_memory*, *switches*, *LEDs*, and *jtag_uart* to the clock output, *afi_clk* of *DDR2_Controller*.
- Connect the *data_master* and *instruction_master* ports of the NIOS II processor to the *avl* port of *DDR2_Controller*.

Double click on the *avl* base address of the *DDR2_Controller* and set it to **0x4000_0000**. Your system should now look similar to the one in Figure 8. Right-click on the Nios II processor component to get to the window in Figure 9. Set the Reset Vector memory and Exception Vector memory to DDR2_Controller.avl and press Finish to return to the window in Figure 8. Click on Generate HDL... > Generate to generate your system and then close Platform Designer.

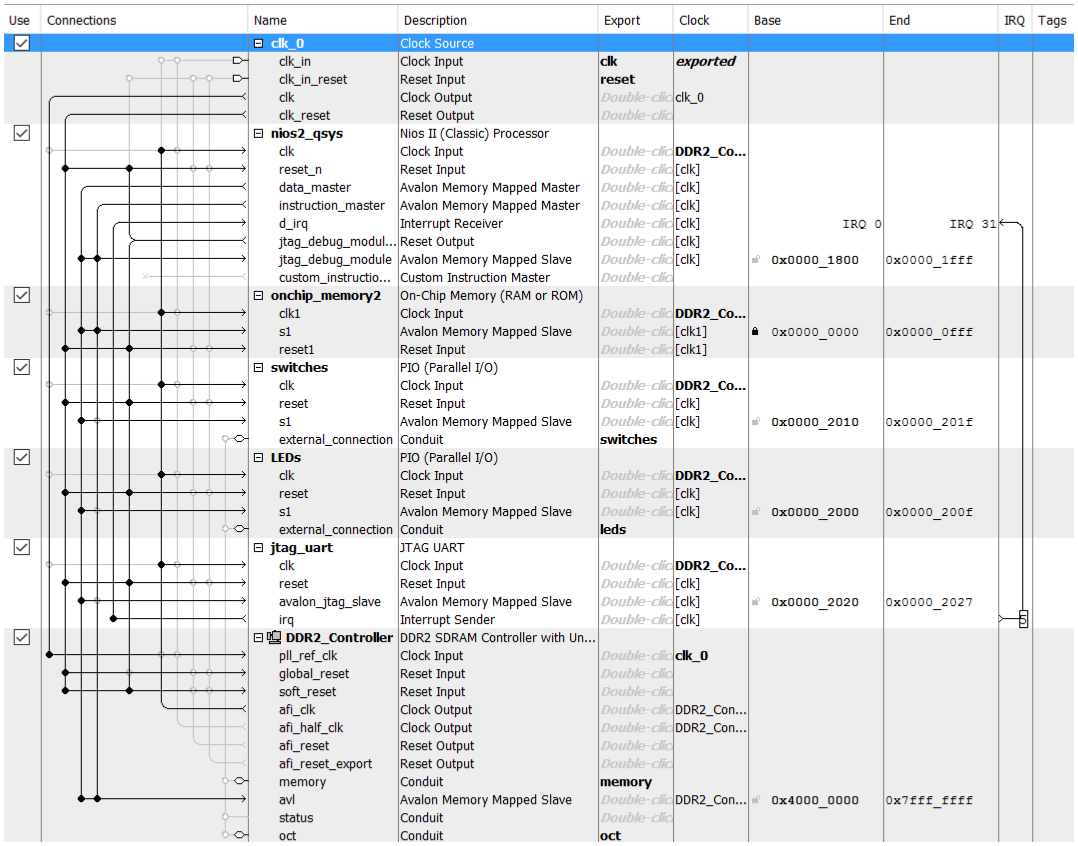

Figure 8. Final Platform Designer system with DDR2 Controller.

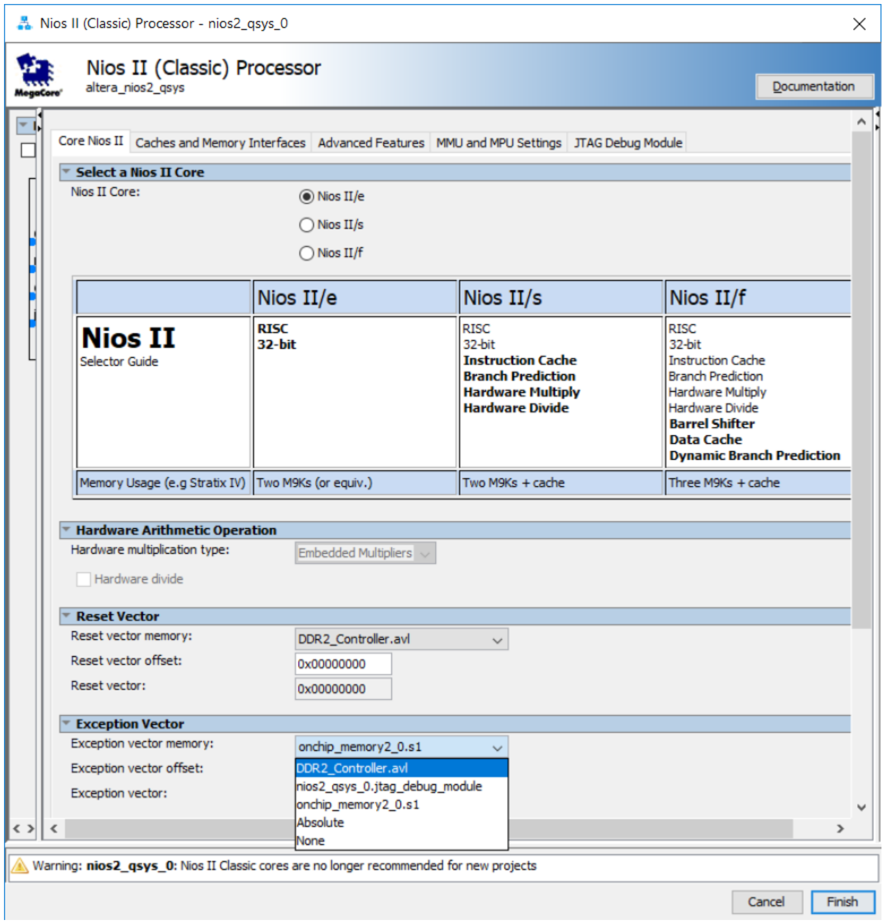

Figure 9. Changing the reset and exception vectors of the NIOS II processor.

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 VHDL entity is presented in Figure 10. The input and output ports of the entity use the pin names for the 50-MHz clock, *OSC_50_BANK3*, pushbutton switches, *BUTTON*, toggle switches, *SLIDE_SW*, and LEDs, *LED*, as used in our original design. They also use the pin names *M1_DDR2_addr*, *M1_DDR2_ba* , *M1_DDR2_cas_n*, *M1_DDR2_cke*, *M1_DDR2_clk*, *M1_DDR2_clk_n*, *M1_DDR2_cs_n*, *M1_DDR2_dm*, *M1_DDR2_dq*, *M1_DDR2_dqs*, *M1_DDR2_dqsn*, *M1_DDR2_odt*, *M1_DDR2_ras_n*, *M1_DDR2_we_n*, *M1_DDR2_oct_rdn*, *M1_DDR2_oct_r* which correspond to the DDR2 SDRAM signals indicated in Figure 2. The pin assignments are included in the file *DE4.qsf* that is included with the design files for this tutorial.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY lights IS
    PORT (
        SLIDE_SW : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        BUTTON, M1_DDR2_oct_rdn, M1_DDR2_oct_rup : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
        OSC_50_BANK3 : IN STD_LOGIC;
        LED : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        M1_DDR2_addr : OUT STD_LOGIC_VECTOR (13 DOWNTO 0);
        M1_DDR2_ba : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
        M1_DDR2_cas_n, M1_DDR2_ras_n, M1_DDR2_we_n : OUT STD_LOGIC_VECTOR (0 DOWNTO
            0);
        M1_DDR2_clk, M1_DDR2_clk_n : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        M1_DDR2_cke, M1_DDR2_cs_n, M1_DDR2_odt : OUT STD_LOGIC_VECTOR(0 DOWNTO 0);
        M1_DDR2_dm : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        M1_DDR2_dq : INOUT STD_LOGIC_VECTOR (63 DOWNTO 0);
        M1_DDR2_dqs, M1_DDR2_dqsn : 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(1 DOWNTO 0);
            memory_mem_ck_n : OUT   STD_LOGIC_VECTOR(1 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_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_rdn : IN STD_LOGIC;
            oct_rup : IN STD_LOGIC);
    END COMPONENT;
```

Figure 10. 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_BANK3,
        reset_reset_n       => BUTTON(0),
        switches_export     => SLIDE_SW,
        leds_export         => LED,
        memory_mem_a        => M1_DDR2_addr,
        memory_mem_ba       => M1_DDR2_ba,
        memory_mem_ck       => M1_DDR2_clk,
        memory_mem_ck_n     => M1_DDR2_clk_n,
        memory_mem_cke      => M1_DDR2_cke,
        memory_mem_cs_n     => M1_DDR2_cs_n,
        memory_mem_dm       => M1_DDR2_dm,
        memory_mem_ras_n    => M1_DDR2_ras_n,
        memory_mem_cas_n    => M1_DDR2_cas_n,
        memory_mem_we_n     => M1_DDR2_we_n,
        memory_mem_dq       => M1_DDR2_dq,
        memory_mem_dqs      => M1_DDR2_dqs,
        memory_mem_dqs_n    => M1_DDR2_dqsn,
        memory_mem_odt      => M1_DDR2_odt,
        oct_rdn             => M1_DDR2_oct_rdn(0),
        oct_rup             => M1_DDR2_oct_rup(0));
END Structure;
```

Figure 10. The top-level VHDL entity that instantiates the expanded Nios II system. (Part *b*).

Perform the following:

- Enter the code in Figure 10 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 11. Select and run the script Project > nios_system > synthesis > submodules > nios_system_DDR2_Controller_p0_pin_assignments.tcl; the script is required to correctly set the differential pins needed by the DDR2 SDRAM Interface.

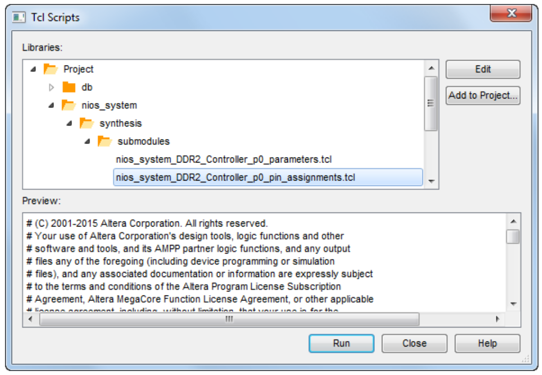

Figure 11. TCL Scripts window in Quartus Prime.

- 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 DE4 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 12.

If successful, the lights on the DE4 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 12. 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 DDR2 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 13 will appear. Accept the default settings and press Finish. Right-click on the component and rename it *Clock_Crossing_Bridge*.

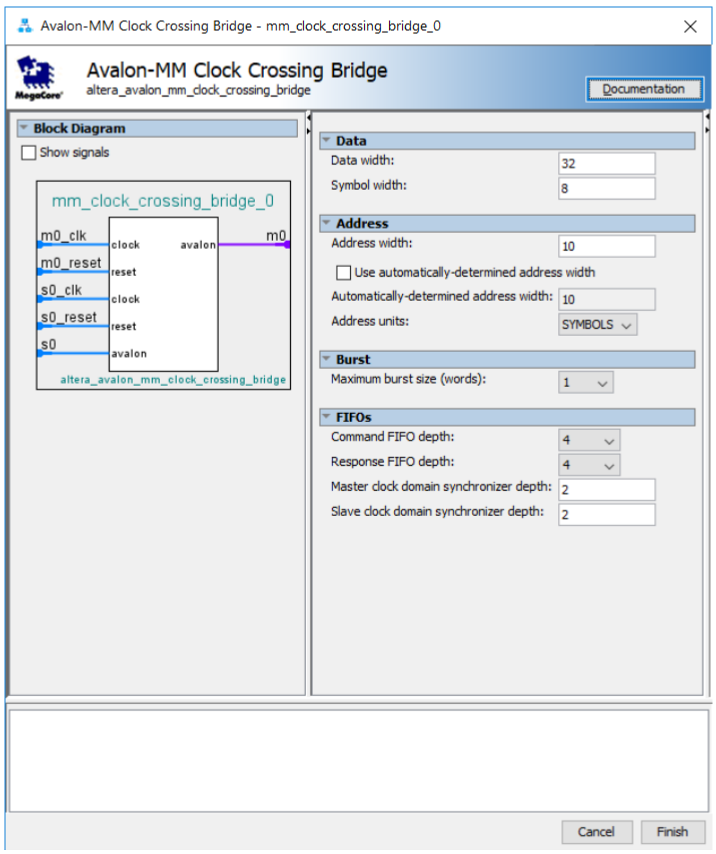

Figure 13. The Avalon-MM Clock Crossing Bridge

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 *DDR2_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 14. Regenerate the system then recompile the top-level VHDL entity. Finally use the updated assembly program given in Figure 15 to test your system with the Intel FPGA Monitor Program.

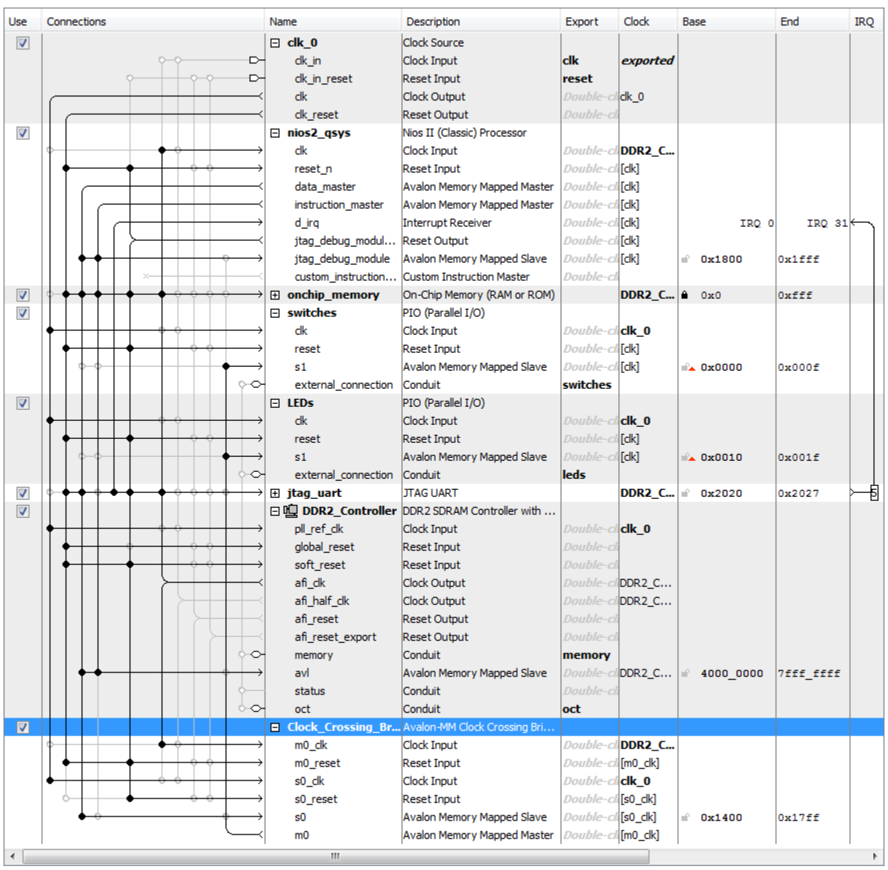

Figure 14. The final Nios II system.

```
.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 15. New Assembly language code to control the lights.