1 Objectives

This lab has been constructed to introduce the development of dedicated embedded system based on NIOS-II CPU core and other interface IPs from Altera. DE2 Altera board having a Cyclone-II FPGA is used to implement the target embedded computer system. This lab introduces you to the system development flow for the Nios II processor. This lab-tutorial is a good starting point to the general concept of building embedded systems in FPGAs. In this lab you will build a Nios II hardware system and create a software program to run on the Nios II system. This lab also presents an introduction to Altera’s SOPC Builder software, which is used to implement a system that uses the Nios II processor on an Altera FPGA device.

2 Introduction

The students will develop and implement a Nios-II based embedded computer system by employing a Cyclone-II FPGA based DE2 board from Altera. Quartus II CAD tools along with SOPC builder and Nios IDE software development environment is used throughout in this lab.

Nios II CPU is a soft processor, defined in a hardware description language (e.g. Verilog or VHDL), which can be implemented in Altera’s FPGA devices by using the Quartus II CAD system. To implement a useful embedded system it is necessary to add other functional units such as memories, input/output interfaces, timers, communications interfaces, etc. To facilitate the implementation of such systems, it is useful to have computer-aided-design (CAD) software for implementing an embedded system-on-a-programmable-chip (SOPC). You are going to build a NIOS CPU based system shown as Quartus block diagram. The NIOS system will read 8 binary switches and show their values on 8 LEDs given on the DE2 board.

Figure 1: Example Nios II System with I/O

In this lab, the system development flow is illustrated by giving step-by-step instructions for using the SOPC Builder in conjunction with the Quartus II software to implement a simple embedded computer system. The last step in the development process involves configuring the designed circuit in an actual FPGA device, and running an application program. The screen captures in the tutorial were obtained using the Quartus II version 5.1; for 5.0 version of the software, some of the images may be slightly different.
3 NIOS II System

In this section the complete design flow for creating a Nios II system and prototyping it on a target board is discussed. Figure 2 shows the Nios II system development flow consists of three types of development: hardware design steps, software design steps, and system design steps, involving both hardware and software. For simpler Nios II systems, one person might perform all the steps. For more complex systems, separate hardware and software designers might be responsible for different steps. System design steps involve both the hardware and software, and might require input from both sides. In the case of separate hardware and software teams, it is important to know exactly what files and information must be passed between teams at the points of intersection in the design flow.

Figure 2: Nios II System Development Flow
A simple Nios II based embedded system that will be implemented in this lab/tutorial on the DE2 board is shown in Figure 3. The Nios II processor and the interfaces needed to connect to other ICs on the DE2 board are implemented over the Cyclone II FPGA. These components are interconnected by means of an interconnection network called the Avalon Switch Fabric. The memory blocks in the Cyclone II device can be used to provide an on-chip memory for the Nios II processor. The SRAM, SDRAM and Flash memory chips on the DE2 board are accessed through appropriate interfaces. Parallel and serial input/output interfaces provide typical I/O ports used in embedded computer systems. A special JTAG UART interface is used to connect to the circuitry that provides a Universal Serial Bus (USB) link to the host computer to which the DE2 board is connected. This circuitry and the associated software is known as USB-Blaster. Another module, JTAG Debug module, is provided to allow the host computer to control the Nios II system. It makes it possible to perform operations such as downloading programs into memory, starting and stopping execution, setting breakpoints, and collecting real-time execution trace data. All parts of the Nios II system implemented on the FPGA are defined by using a hardware description language. One can also use the SOPC Builder tool to implement the desired embedded system simply by choosing the required components and specifying the parameters needed to make each component fit the overall requirements of the system. In this lab/tutorial, we will also illustrate the capability of the SOPC Builder by designing the simple embedded system of Figure 3. The same approach can be used to design large and complex embedded computer systems.

![Figure 3: A Simple Nios II based Embedded Computer System.](image)

Our example system of Figure 3 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. The Nios II CPU will perform this by executing a program stored in the on-chip memory. Continuous operation is required, such that as the switches are toggled the lights change accordingly. You will use the SOPC Builder to design the hardware depicted in Figure 3. Next, you will also assign the Cyclone II pins to realize the connections between...
the parallel interfaces and the switches and LEDs, which act as I/O devices. Then, you will configure the FPGA to implement the designed system. Finally, you will use the IDE software development environment to compile, download and execute a Nios II program that performs the desired task. Doing this lab, the students will learn about:

- Quartus II CAD system
- Using the SOPC Builder to design a Nios II-based embedded computer system
- Integrating the designed Nios II computer system into a Quartus II project
- Implementing the designed system on the DE2 board
- Running an application program on the Nios II processor

### 4 Main Steps for System Configuration by using SOPC Builder

SOPC builder allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters. To implement the system of Figure 2, you have to instantiate the following functional units:

- Nios II processor, which is referred to as a Central Processing Unit (CPU)
- On-chip memory, which consists of the memory blocks in the Cyclone II chip; we will specify a 24-Kbyte memory arranged in 32-bit words
- Two parallel I/O interfaces
- JTAG UART interface for communication with the host computer

To define the target embedded computer system, start the Quartus II software and perform the following steps: Create a new Quartus II project for your system. As shown in Figure 4, we stored our project in a directory called `sopc_builder_tutorial`, and we assigned the name `lights` to both the project and its top-level design entity.

You can choose a different directory or project name. In your project, choose the EP2C35F672C6 chip as the target device, because this is the FPGA on the DE2 board. You may also attach a preliminary block diagram/schematic file “lights.bdf” for your NIOS system while creating the project.

You also need to set the frequency value of "Default required fmax" at 50Mhz by going to Assignments => Timing Settings and the Clock setting of "Default required fmax" to 50MHz.

![Figure 4. Create a new project.](image)
2. Select Tools > SOPC Builder, which leads to the pop-up box in Figure 5. Enter nios_system as the system name; this will be the name of the system that the SOPC Builder will generate. Choose VHDL as the target HDL, in which the system module will be specified. Click OK to reach the window in Figure 6.

![Figure 5. Create a new nios II system.](image)

3. Figure 6 displays the System Contents tab of the SOPC Builder, which is used to add components to the system and configure the selected components to meet the design requirements. The available components are listed on the left side of the window. Before choosing our components, examine the area in the figure labeled Target. A drop-down list is provided that allows some available Altera boards to be selected. It is not necessary to select a board, and since the DE2 board is not included in the list leave the selection as Unspecified board. Then, check the setting for the Device Family and ensure that Cyclone II is selected.

4. The Nios II processor runs under the control of a clock. For this lab we will make use of the 50-MHz clock that is provided on the DE2 board. It is possible to specify the names and frequency of clock signals in the SOPC Builder. If not already included in this list, specify a clock named \textit{clk} with the source designated as External and the frequency set to 50.0 MHz.
5. In this step, specify the Nios processor as follows:
   • On the left side of the window in Figure 6 select Avalon Components > Nios II Processor - Altera Corporation and click Add, which leads to the window in Figure 7.
   • Choose Nios II/e which is the simplest version of the processor. Click Finish to return to the window in Figure 6, which now shows the Nios II processor specified as indicated in Figure 8. There may be some warnings or error messages displayed in the SOPC Builder Messages window (at the bottom of the screen), because some parameters have not yet been specified. Ignore these messages as we will provide the necessary data later. Observe also that a new tab called Nios II More “cpu_0” Settings appears, which allows further configuration of the processor - we will not use it.

![Altera Nios II - cpu_0](image)

Figure 7. Create a Nios II processor.

6. To specify the on-chip memory perform the following:
   • Select Avalon Components > Memory > On-Chip Memory (RAM or ROM) and click Add
   • In the On-Chip Memory Configuration Wizard window, shown in Figure 9, set the memory width to 32 bits and the total memory size to 20 Kbytes
   • Do not change the other default settings
   • Click Finish, which returns to the System Contents tab as indicated in Figure 10

7. Specify the input parallel I/O interface as follows:
   • Select Avalon Components > Other > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard in Figure 11
   • Specify the width of the port to be 8 bits and choose the direction of the port to be Input, as shown in this figure.
   • Click Finish to return to the System Contents tab as given in Figure 12
Figure 8. The defined Nios processor.

Figure 9. Define the on-chip memory.
Figure 10. Including the on-chip Memory

Figure 11. Defining a parallel input interface.
8. In the same way, specify the output parallel I/O interface:
   • Select Avalon Components > Other > PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard again
   • Specify the width of the port to be 8 bits and choose the direction of the port to be Output
   • Click Finish to return to the System Contents tab.

9. We wish to connect to a host computer and provide a means for communication between the Nios II system and the host computer. This can be accomplished by instantiating the JTAG UART interface as follows:
   • Select Avalon Components > Communication > JTAG UART and click Add to reach the JTAG UART Configuration Wizard in Figure 13
   • Do not change the default settings
   • Click Finish to return to the System Contents tab

10. The complete system is depicted in Figure 14. Note that the SOPC Builder automatically chooses names for the various components. The names are not necessarily descriptive enough to be easily associated with the target design, but they can be changed. In Figure 3, we use the names Switches and LEDs for the parallel input and output interfaces, respectively. These names can be used in the implemented system. Right-click on the pio_0 name and then select Rename. Change the name to Switches. Similarly, change pio_1 to LEDs.

11. The base and end addresses of the various components in the designed system can be assigned by the user, but the SOPC Builder can also assign them automatically. We will choose the latter possibility. So, select the command (using the menus at the top of the SOPC Builder window) System > Auto-Assign Base Addresses, which produces the assignment shown in Figure 15.
Figure 13. Defining the JTAG UART interface.

Figure 14. The complete nios- system.
12. Having specified all the components needed to implement the desired embedded system, it can now be generated. Select the System Generation tab, which leads to the window of Figure 16. Turn off Simulation – Create simulator project files, because in this lab we will not deal with the simulation of hardware. Click Generate on the bottom of the SOPC Builder window. The generation process produces the messages displayed in the figure. When the message “SUCCESS: SYSTEMGENERATION COMPLETED” appears, click Exit. This returns to the main Quartus II window.

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

Now we introduce to the nios system development and flow for its application software. This portion program/configure a Nios II CPU based system on an FPGA and create a software program to run on the Nios II system. The simple Nios II based embedded system that is implemented is shown in Figure 3. Our example system of Figure 3 realizes a trivial task. Eight toggle switches on the DE2 board, SW[7..0] are used to turn on or off the eight green LEDs, LEDG[7..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. The Nios II CPU will perform this by executing a program stored in the on-chip memory. Continuous operation is required, such that as the switches are toggled the lights change accordingly. You have used the SOPC Builder to design the hardware depicted in Figure 3. Next, you will also assign the Cyclone II pins to realize the connections between the parallel interfaces and the switches and LEDs, which act as I/O devices. Then, you will configure the FPGA to implement the designed system. Finally, you will use the IDE software development environment to compile, download and execute a Nios II program that performs the desired task.
5. Integration of the Nios II System into a Quartus II Project

To complete the hardware design, we need to perform the following steps:
In this section you will perform the following steps to complete the hardware design:
- Instantiate the SOPC Builder nios-system module in the Quartus II project.
- Assign FPGA pins.
- Compile the Quartus II project
- Verify timing

5.1 Instantiation of the Module Generated by the SOPC Builder

The instantiation of the generated module depends on the design entry method chosen for the overall Quartus II project. We have chosen to use BDF (schematic), but the approach is similar for both Verilog and VHDL entry methods. Normally, the Nios II module is likely to be a part of a larger design. However, in the case of our simple example there is no other circuitry needed. All we need to do is instantiate the Nios II system in our top-level BDF file, and connect inputs and outputs of the parallel I/O ports, as well as the clock and reset inputs, to the appropriate pins on the Cyclone II device. You can use the initial “lights.bdf” file provided in project directory of the course web page. At this stage, you make a note of the base addresses for SW and LED pio of
the nios_system you have built with the SOPC builder. We will use these addresses to read from switches and write to LEDs in the C/C++ code for Nios II CPU at the end of this lab part.

You will instantiate a system module symbol **nios_system** into the BDF. To instantiate the system module in the BDF, perform the following steps:

1. Double click in the empty space between the input and output pins. The Symbol dialog box appears.
2. Under Libraries:, expand Project.
3. Click nios_system. The symbol dialog box displays the nios_system symbol.
4. Click OK. You return to the BDF. The nios_system symbol tracks with your mouse pointer.
5. Connect the inputs on the symbol with the wires on the left-hand side of the BDF.
6. Click the left mouse button to drop the symbol in place. Figure 17 shows the complete BDF using the LED pins.

![BDF schematic](image.png)

**Figure 17: The Example Nios II System with I/O**

### 5.2 PIN Assignments

There are two options for PIN assignments of the DE2 board. The easiest one is to import the DE2 pin assignment file “DE2_pin_assignments.csv” by using the Assignments > Import Assignment. The assignment file “DE2_pin_assignments.csv” is available at the course web in the ../projects/nios2 directory. However, by importing this file you will only use 17 pin assignments all the other assignments will appear with question marks.

As a 2nd option, perform the Start Analysis and Synthesis. Then Choose Pins in the Assignment menu of Quartus II Assignment Editor. You can sort the pin assignments by name by clicking the To column in the Assignment Editor as shown in Figure 18. Scroll down until the pin LEDG[0]–LEDG[7] appears in the Assignment Editor. Double click in the location cell for pin LEDG[0] and select the appropriate FPGA pin. The FPGA pins can be copies from the excel file “DE2_pin_assignments.csv” provided in the lab3 directory. You can also copy and paste multiple pin assignments. Similarly SW[0]–SW[7] and CLOCK_50 pins can be found for assignment. Finally save the pin assignment by selecting save in the File menu and close the assignment editor.

After proper pin assignments, they should appear in your BDF file when opened in Quartus II.

### 5.3 Compile the Quartus II Project

At this point you are ready to compile the Quartus II project and verify that the resulting design meets timing requirements. You must compile the hardware design to create an FPGA configuration file that you can download to the board. After the compilation completes, you may like to analyze the timing performance of FPGA design to verify that the design will work in hardware.

Perform the following steps:
1. On the Processing menu, click **Start Compilation**
2. The Quartus II Status utility window displays progress. The compilation process can take several minutes.
   When compilation completes, a dialog box displays the message "Full compilation was successful."
3. Click **OK**. The Quartus II software displays the Compilation Report window.

You have finished integrating the Nios II system into the Quartus II project. You are ready to download the FPGA configuration file to the target board.

---

**Figure 18: Assigning Pins with the Quartus II Assignment Editor**

---

**5.4 Download Hardware Design to Target FPGA**

Program and configure the Cyclone II FPGA in the JTAG programming mode as follows:

1. Connect the DE2 board to the host computer by means of a USB cable plugged into the USB-Blaster port. Turn on the power to the DE2 board. Ensure that the RUN/PROG switch is in the RUN position.
2. Select Tools > Programmer to reach the window in Figure 19.
3. If not already chosen by default, select JTAG in the Mode box. Also, if the USB-Blaster is not chosen by default, press the Hardware Setup... button and select the USB-Blaster 3/2 depending on your computer system hardware in the window that pops up.
4. The configuration file `lights.sof` should be listed in the window. If the file is not already listed, then click Add File and select it.
5. Click the box under Program/Configure to select this action.
6. At this point the window settings should appear as indicated in Figure 19. Press Start to configure the FPGA.
6. Development and Execution of the Application Program

Having configured the required hardware in the FPGA device, it is now necessary to create and execute an application program that performs the desired operation of reading from switches and writing to LEDs. This can be done by writing the required program either in the Nios II assembly language or in a high-level language such as C. We will illustrate the C/C++ programming approach. The parallel I/O interfaces (SW and LEDs) generated by the SOPC Builder is accessible by means of registers in the interface.

Depending on how the PIO is configured, there may be as many as four registers. One of these registers is called the Data register. In a PIO configured as an input interface, the data read from the Data register is the data currently present on the PIO input lines. In a PIO configured as an output interface, the data written (by the Nios II processor) into the Data register drives the PIO output lines. If a PIO is configured as a bi-directional interface, then the PIO inputs and outputs use the same physical lines. In this case there is a Data Direction register included, which determines the direction of the input/output transfer. In our unidirectional PIOs, it is only necessary to have the Data register. The addresses assigned by the SOPC Builder in the example are 0x00000080 for the Data register in the PIO called Switches and 0x00000810 for the Data register in the PIO called LED. You should check the base register addresses in your SOPC builder window as shown in Figure 20. You can also find the full description of the PIO interface by opening the SOPC Builder window in Figure 20 and right-clicking on the module name of a PIO (either Switches or LEDs). Then, in the pop-up box select Data Sheet to open the document PIO Core with Avalon Interface that gives a full description of the interface. To use this facility you may need to be connected to the Internet.

6.1 Create a New C/C++ Application Project

In this section you will start the Nios II integrated development environment (IDE) and compile a simple C/C++ language program. This section presents only the most basic software development steps to demonstrate software running on the hardware system you created in previous sections. You will create a new Nios II C/C++ Application Project. First of all start the Nios II IDE. Click Run Nios II IDE. Tab in the SOPC builder window already opened by you as shown in Figure 21. When the Workspace Launcher dialog box appears, click OK to accept the default workspace location or you can specify any other empty directory.
Perform the following steps for the application software development.
1. On the File menu, point to New, and then click C/C++ Application to open the New Project.
2. Click Browse under Select Target Hardware. The Select Target Hardware dialog box opens.
4. Select nios_system.ptf.
5. Click Open. You return to the New Project wizard, and the SOPC Builder System and CPU fields are now filled in.
6. Select Blank Project in the Select Project Template list. See Figure 22 for another project count_binary.
7. Click Finish.
We get a blank window for a C application code to be typed. The Nios II IDE displays two new projects in the C/C++ Projects view on the left-hand side of the workbench: blank_project_0 and blank_project_0_syslib. blank_project_0 is your C/C++ application project, and count_binary_0_syslib is a system library that encapsulates the details of the Nios II system hardware. The left-hand pane of the IDE workbench has two tabbed views: The C/C++ Projects view and the Navigator view. Click the C/C++ Projects tab to display the C/C++ Projects view. This view is appropriate for most C/C++ development activity. Then create a new C program file lights.c. Type your code given in Figure 23 or copy and paste.

Figure 21. Generated Nios_System.
#include <stdio.h>
int main()
{
    unsigned char * Switches;
    unsigned char * LEDs;

    Switches = (unsigned char *)0x0000800;
    LEDs = (unsigned char *)0x0000810;

    printf("Beginning.\n");

    while(1){
        *LEDs = *Switches;
    }

    return 0;
}

Figure 23. C language code to control the lights

You can make edits or type in any C/C++ program file in the IDE and repeat the compile and run steps described next.
6.2 Compile the Project

In this section you will compile the project to produce an executable software image. For the example tutorial design, you must first adjust the project settings to minimize the memory footprint of the software, because your Nios II hardware system contains only 20 Kbytes of memory.

Perform the following steps:
1. Right-click blank_project_0 and click System Library Properties.
2. The System Library page contains all settings related to how the program interacts with the underlying hardware. Therefore, the settings here reflect names you specified when creating the Nios II hardware in section "Define the System in SOPC Builder".
3. Change the following settings, which affect the size of the compiled executable (see Figure 24).
   a. Turn off Clean exit (flush buffers).
   b. Turn on Small C library.
4. Click OK to close the Properties dialog box and return to the IDE workbench.
5. Then Build the project

![Image](image.png)

Figure 24. NIOS II IDE New Project Wizard

6.3 Run the Program

To download the software executable to the DE2 board, perform the following steps:
1. Right-click the blank_project_0 project, point to Run As, and then click Nios II Hardware. The IDE downloads the program to the FPGA on the target board and starts execution.
   When the target hardware starts executing the program, the Console view displays the character I/O output “Beginning”. If you connected LEDs to the Nios II system during "Integrate the SOPC Builder System into Quartus II Project", then the LEDs show the switch positions.
2. Click Terminate (the red square) on the toolbar at the upper-right hand corner of the Console view to terminate the run session. When you click Terminate, the IDE disconnects from the target hardware and leaves the Nios II processor running.
References:
1. Introduction to the Altera SOPC Builder Using VHDL Design
2. Nios II Hardware Development Tutorial

Appendix: BDFs for Nios_System