# Educational Objective

The acronym SoC stands for “System on a Chip”, and generally is used to describe a system that contains programmable logic, a microprocessor, and in some cases analog components on a single chip. Given the large amount of logic available on FPGAs, one way to implement such a system is to define a microprocessor using HDL (ex. Verilog or VHDL) and to synthesize this along with the rest of your HDL project. This is called a “soft-core” processor, since it is implemented in HDL “software”. Writing the HDL for a microprocessor is complex, but fortunately there are tools that can generate this system for us, in such a way that we fairly easily instantiate a soft-core processor in our HDL code.

In this lab you will how to navigate through the Altera toolchain for creating, programming, and debugging a Nios II processor on an Altera Cyclone V FPGA.

Table of Contents

[Educational Objective 1](#_Toc472803370)

[Deliverables 2](#_Toc472803371)

[Lab-Procedure 2](#_Toc472803372)

[Create Quartus Project 3](#_Toc472803373)

[Add the Top Level VHDL Module 4](#_Toc472803374)

[Make the Pin Assignments 4](#_Toc472803375)

[Compile the Design 4](#_Toc472803376)

[Program the Device 4](#_Toc472803377)

[Create the Nios Processor 5](#_Toc472803378)

[Instantiate the Nios System in your VHDL and re-program the FPGA 8](#_Toc472803379)

[Compile and Download the C-Code 9](#_Toc472803380)

[Run and Debug Your Code on the DE1-SoC Board 10](#_Toc472803381)

# Deliverables

Create a directory named cpet561/lab1. This directory will contain all of your work for lab 1.

For this lab, you will submit the following files in the drop-box on myCourses/.

* All source files including VHDL (\*.vhd), C code (\*.c), and assembly code (\*.s)
* Within Quartus you will select Project -> Archive Project to create an archive of your design. You will submit the archive file along with your source files.

# Lab-Procedure

A Nios processor can be implemented on the DE1-SoC board as shown in Figure 1.



Figure : Typical Nios Processor Implemented on the DE1-SoC board.

The system for this tutorial is given in Figure 2. The system realizes a trivial task. Eight toggle switches on the DE1 board, SW7−0, are used to turn on or off the eight green LEDs, LEDG7−0. The switches are connected to the Nios II Software Build Tools 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 a program stored in the on‐chip memory. Continuous operation is required, such that when the switches are toggled the lights change accordingly.

Altera's Quartus, QSYS, and Eclipse software tools will be used to design the hardware depicted in Figure 2. Next, Cyclone pins will be assigned to realize the connections between the FPGA and the I/O devices. Then, the FPGA will be configured to implement the designed system. Finally, the Nios Software Build Tools will be used to assemble, download and execute a program on the Nios processor that will perform the desired task.



Figure : The simple Nios system that will be used for this lab.

## Create Quartus Project

1. Select File->Create New Project from the Quartus menu.
2. On the first page of the new project wizard enter the following parameters…
   1. Working Directory = //cpet561/lab1
   2. Name of project = lights
   3. Top level design = lights
3. Click next to get to the Project Type page and select Empty Project.
4. Click next to get to the Add Files page. For now we will not add any files, so just click next to get to the Family and Device Settings page. On the Family and Device Settings page you will enter the following parameters…
   1. Family = Cyclone V
   2. Name Filter = 5CSEMA5F31C6
   3. After you enter the above filter name, there should be only one device left in the list. Select this device and click next.
5. Continue to click next until you get to the last page, then click Finish.

## Add the Top Level VHDL Module

1. Get the file lightsNoNios.vhd from myCourses, rename it to lights.vhd, and put it in the project directory //cpet561/lab1.
2. Now select Project->Add/Remove Files in Project from the Quartus menu.
3. Click on the navigate button (with the three dots) next to the file name entry box. Navigate to the lights.vhd file that you added in step one above and select it.
4. Click Add
5. Click OK

## Make the Pin Assignments

1. All the inputs and outputs in your top level VHDL file must be mapped to pins on the Cyclone V FPGA device on the DE1-SoC board. In order to do this, you will need to get the assignments.qsf file from myCourses.
2. Select Assignment->Import Assignments from the Quartus menus.
3. Navigate to the assignments.qsf file you downloaded in step 1.
4. Check “Copy Existing Assignments”
5. Click OK

## Compile the Design

1. Select Processing->Start Compilation from the Quartus menus (or use the shortcut on the toolbar, it is a green arrow).
2. Verify that the process completed with no errors or warnings.

## Program the Device

At this point you are ready to program the Cyclone V FPGA. To do this you will use the lights.sof file generated during compilation. This is a large file that determines the state of all the programmable logic resources and block RAM on the FPGA. Since the top level VHDL causes one of the LEDs to blink, the blinking LED will be an indication that the FPGA is indeed programmed.

1. Connect the DE1-SoC board and your host PC using the provided USB cable.
2. Turn the DE1-SoC on.
3. When the programmer application opens…
   1. Click Auto Detect
   2. Select the 5CSEMA5F31 from the devices presented in the pop-up menu.
   3. There should now be two devices shown in the JTAG chain.
   4. Click on 5CSEMA5F31 in the list of devices.
   5. Click on “Change File”
   6. Navigate to //cpet561/lab1/output\_file/lights.sof and select it. This file should now show up in the list next to that device.
   7. Click on the Program/Configure button for the 5CSEMA5F31 device. The 5CSEMA5F31 should have a check now in the Program/Configure box, the other device SHOULD NOT.
   8. Click Start.
   9. The progress bar at the type right should turn green and indicate success. Also the LED should now be blinking.

## Create the Nios Processor

QSYS is a tool used in conjunction with the Quartus software. It 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.

In this step we will specify the configuration of Nios soft-core processor we would like, and also what peripherals we would like to include. The Altera QSYS software will then generate all the VHDL code and other files required to implement our processor system.

To implement the system in Figure 2, the following functional units need to be created:

* Nios II processor
* On-chip memory, which consists of the memory blocks in the Cyclone V chip.
* Two parallel I/O interfaces
* JTAG UART interface for communication with the host computer.

To create the Nios system in QSYS, follow the following steps. When you have completed all the steps, the **System Contents** tab of QSYS should look like the one shown in Figure 3.

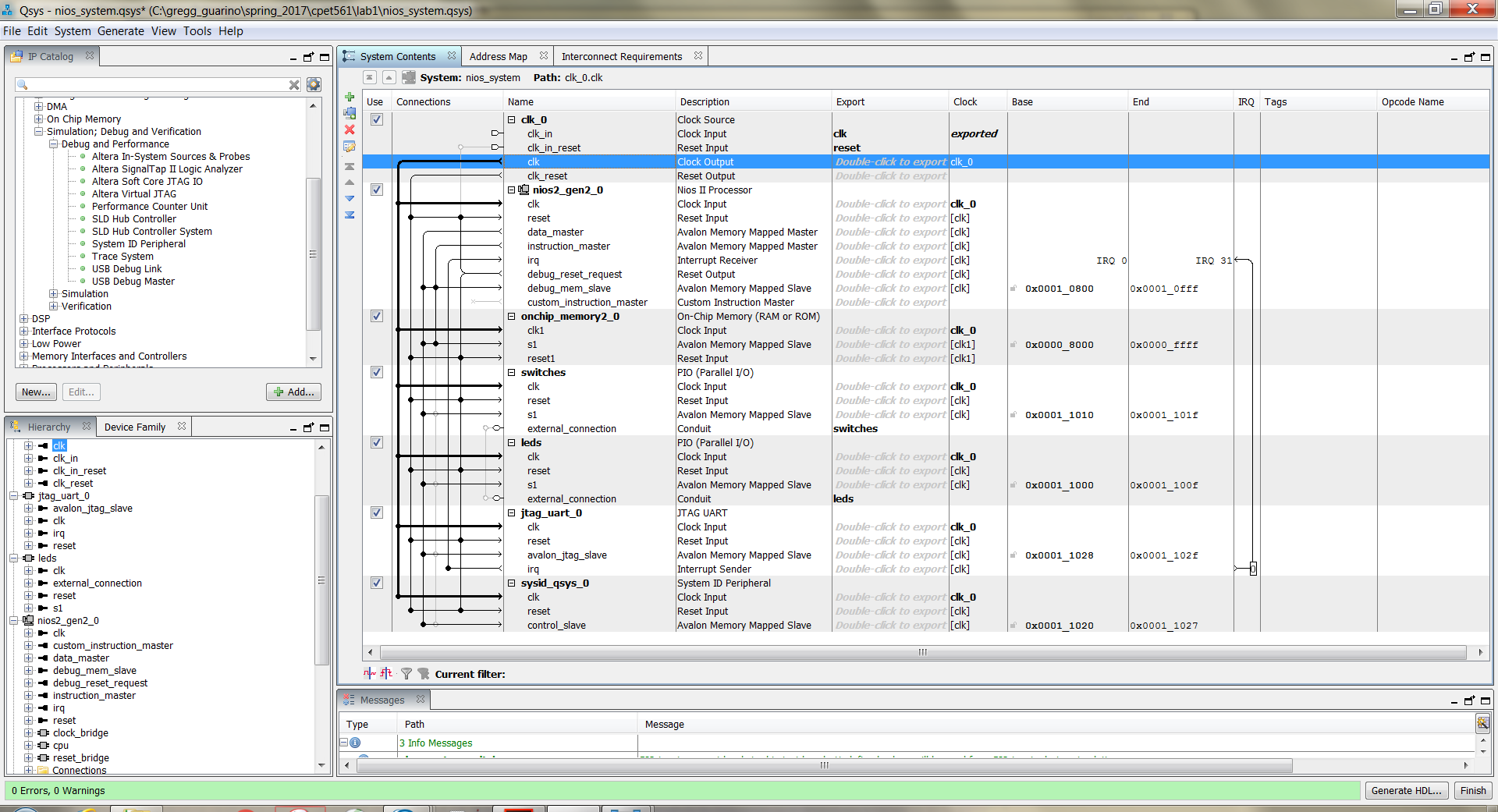


Figure 3: The finished system after all the following steps are completed.

1. Select Tools > QYS, from the Quartus menus. This will bring up the QSYS application, 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.
2. Before you go too far, take the time to save your design and assign a name. Select File > Save from the QSYS menu and enter nios\_system as the system name.
3. The first step in building a system is to specify the processor as follows:
   1. On the left side of QSYS, in the Component Library tab, select Processors & Peripherals->Embedded Processors->Nios II Processor, and click Add. This will bring up the configuration window for the Nios processor.
   2. Choose Nios II/e processor, which is the simplest version. Click Finish to return to the main QSYS window, which now shows the Nios II processor specified. There may be some warnings or error messages displayed in the QSYS Messages window (at the bottom of the screen), because some parameters have not yet been specified. Ignore these messages for now.
4. To specify the on-chip memory perform the following:
   1. Select Basic Functions->On Chip Memory->On Chip Memory (RAM or ROM) and click Add
   2. In the On-Chip Memory Configuration Wizard, set the memory width to 32 bits and the total memory size to 32 Kbytes (i.e. 32768 bytes).
   3. Do not change the other default settings
   4. Click Finish, which returns to the QSYS application.
5. Our system needs a mechanism to get signals or information into the system (in this case it will be the input from the user switches). To do this we need to define an input parallel I/O interface using the procedure defined below.
   1. Select Processors & Peripherals->Peripherals->PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard.
   2. Specify the width of the port to be 8 bits and choose the direction of the port to be Input.
   3. Click Finish to return to the QSYS application.
6. Our system needs a mechanism to get signals or information out of the system (in this case it will be the output signals that drive the LEDs. We can create an output parallel I/O interface in the same way we created the input interface.
   1. Select Processors & Peripherals->Peripherals->PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard again
   2. Specify the width of the port to be 8 bits and choose the direction of the port to be Output
   3. Click Finish, which returns to the QSYS application.
7. In order to connect to a host computer, and provide a means of communication with the Nios II Software Build Tools, a JTAG UART interface needs to be instantiated as follows:
   1. Select Interface Protocols > Serial > JTAG UART and click Add to reach the JTAG UART Configuration Wizard.
   2. Do not change the default settings
   3. Click Finish, which returns to the QSYS application.
8. The last component we need to add to our system is System ID Peripheral. The System ID Peripheral component does not affect the functionality of the design directly, but instead safeguards against accidentally downloading software compiled for a different Nios II system.
   1. Select Basic Functions->Simulation; Debug & Verification->Debug & Performance->System ID Peripheral and click Add to reach the System ID Peripheral Wizard.
   2. Enter a unique value of System ID. The value is not important as long as it is unique. It may be useful to enter a HEX number that reflects the lab number.
   3. Click Finish, which returns to the QSYS application.
9. QSYS 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. To change the component names, right-click on the pio\_0 name and then select Rename. Change the name to switches. Similarly, change pio\_1 to leds.
10. At this point we have defined the components required in our system, but we have not defined how they are connected. We do this first by connecting the clock and resets. When we opened QSYS there was a clock component already present in the design. This component takes in an external clock and reset, and outputs a copy of the clock and a reset synchronized to the clock. We will connect these signals to the clock and reset port of the other components. Start by clicking the clk output of the clk\_0 component and then moving the mouse in the connection column. By clicking on the net we are working on first, the clock net is highlighted making it easier to follow. With the mouse over the connection column, a series of lines will appear. Open circles are places where connections can be made, and filled in dots are where connections are present. Click on all the open circles on the clock net to make connections to the other components.
11. Repeat the process from step 10 above to connect the reset signal (clk\_reset output port) from clk\_0 component to the other components.
12. To have control of the system, the JTAG debug module needs to be able to reset the system, except for the clk\_0 component. To allow this, we need to connect the **debug\_reset\_request** output port of the JTAG module to all the resets, except the reset into the clk\_0 component. We use the same procedure above.
13. Next we need to connect the instruction bus (i.e.: instruction\_master port) of the Nios II processor to the on-chip memory, so it can read the instructions from memory. The Nios II processor’s bus is called Avalon, and the processor is the master of the bus. Select the instruction\_master and repeat the procedure above to connect the Nios II’s instruction\_master port to the on-chip memory’s S1 (Avalon slave). Remember the processor’s instruction bus is **only** connected to the on-chip memory.
14. Next we need to connect the data bus (ie: data\_master port) of the Nios II processor to the Avalon slave ports of the on-chip memory **and other components**. The on-chip memory is connected to both the instruction and data ports because our on-chip memory will be used for both instructions and data. For other components, we only need to connect the Nios II processor data\_master port. Connect the data\_master port of the Nios II processor to all the other component’s S1 (Avalon slave) ports.
15. The Nios II processor runs under the control of a clock. For this tutorial, the 50 MHz clock that is provided on the DE2 board will be used. It is possible to specify the names and frequency of clock signals in the QSYS. Select the Clock Setting tab. You should see clock defined from an external source with a frequency of 50MHz. Since this is what we are using we will leave the settings as they are.
16. Now that our Nios II processor is connected to the components, we can edit the Nios II to define where the reset vector and exception vector will be stored. The behavior of the Nios II processor when it is reset is defined by its reset vector. The reset vector is the location in a memory device where the processor fetches the first instruction when it is reset. Similarly, the exception vector is the memory address the processor goes to when an interrupt is raised. To specify these two parameters, perform the following. This will clean up some of the error messages.
    1. Right-click on the Nios II Processor component and then select Edit or double click on Nios II Processor component.
    2. Find the reset vector and exception vector tab and select **onchip\_memory2\_0.s1 in the** drop-down for both the reset vector and the exception vector.
    3. Do **not** change any other settings.
    4. Click Finish, which returns to the QSYS application.
17. The base address and end addresses of the various components in the system can be assigned by the user, but they can also be assigned automatically by QSYS. These are the addresses you will use in your c-code or assembly code to communicate with the various peripherals. To have QSYS automatically assign these addresses, select the command System > Assign Base Addresses from the QSYS menus. Notice this removes the errors about overlapping addresses.
18. We are going to instantiate the system we designed in QSYS as a component in the top level VHDL of our Quartus project. To make the connections from the switches and LEDs to the top-level VHDL, we need to define the ports that will be exported from the Nios component. To do this, double click where it says **double-click to export,** and enter the name. In order to be consistant with the code provided on myCourses you should use the name **switches** for the input port, and **leds** for the output port.
19. The next step is to connect the interrupt from the JTAG UART to the Nios II IRQ. Select the JTAG UART and move the mouse to the IRQ column (last column on the right in the System Contents tab). A zero will appear saying that the interrupt will be connected to the Nios II processor’s IRQ0 signal.
20. Your completed system should now look like Figure 3. The last step is to tell QSYS to go generate all the VHDL code, and other files required to implement your microprocessor system. Click the Generate button, on the bottom right of the QSYS window. The process may take a minute or two to complete. There should be no errors.

## Instantiate the Nios System in your VHDL and re-program the FPGA

For this part of the lab, you will need to use the version of top-level VHDL code which instatiates the Nios processor.

1. Delete the /cpet561/lab1/lights.vhd file.
2. Copy lightsWithNios.vhd from myCourses into the /cpet561/lab1 directory.
3. Rename lightsWithNios to lights.vhd.
4. Re-compile the Quartus project and verify that there are no errors. There will be many (perhaps hundreds) of warnings, this is normal and can be ignored for now.
5. Go back to the Programmer application, and click Start to download the new FPGA configuration, which includes the Nios II soft-core processor. The rate of the LED blinking is twice as fast in this version, which will provide visual indication of which configuration is running.

## Compile and Download the C-Code

At this point, the Nios processor is not doing anything, because we have not yet downloaded a program. If you move the slide-switches it will not effect the LEDs. In this section we will download a C program that will read the switches from the parallel input port, and write the state of the LEDs through the parallel output port. To do this we will use yet another software tool, the **Nios II Software Build Tools for Eclipse**. The procedure is as follows:

1. First we will create a directory in our project to hold the software. Create the directory /cpet561/lab1/software.
2. From Quartus, select Tools->Nios II Software Build Tools for Eclipse.
3. In the window that pops up, navigate to /cpet561/lab1/software and select this for the Workspace.
4. Select File->New->Nios II Application and BSP from Template from the Eclipse menus.
5. On the first tab of the dialog, for the SOPC Information File, navigate to nios\_system.sopcinfo in you lab1 project directory. **Caution**: Be sure that you are selecting the file in the project directory you are currently working in. Sometimes it comes up with an sopc file from a previous project.
6. For the Project Name enter nios\_software.
7. Select Blank Project in the list of Template types.
8. Click Next to get to the next tab, then click Finish. The generation of the project may take a minute.
9. The C code source file is provided to you on myCourses. The file name is lights\_main.c. Download lights\_main.c into the cpet561/lab1/software/nios\_software directory.
10. Right click on **nios\_software\_bsp** in the project explorer tab on the left side of the Eclipse screen. Select Build Project from the menu.
11. Right click on **nios\_software** in the project explorer tab on the left side of the Eclipse screen. Select Build Project from the menu.
12. In step 5 of the section “Instantiate the Nios System in the Top-Level VHDL” you programmed the Cyclone V FPGA with the version of lights.sof which contains the Nios system. If this is still running, then you do not need to do anything in this step. If not, then return to the Quartus programmer, and program the Cyclone V FPGA now with the correct version of lights.sof.
13. Returning now to the Eclipse application, select Run->Debug Configurations from the menu. This will open the Debug Configurations window.
14. In the Debug Configuration window, on the left side, find **Nios II Hardware**. Right click on **Nios II Hardware** and select **New**. This will create **New\_configuration** on the right side of the window. You can change the name of the configuration, but it is not necessary to do so.
15. In the **project** tab of the new configuration, in the **project name** drop-down, select **nios\_software**.
16. In the **target connections** tab click on **Refresh Connections**.
17. Click on the **Debug** button.
18. Click **Yes** to open the debug session.

## Run and Debug Your Code on the DE1-SoC Board

1. When you started the debug session, all the “under the hood” startup and configuration code ran, and the application stopped on the first line of your **main()** function. To run your program click on **resume**, which is the green arrow in the tool bar at the top.
2. Now you should observe that the slide switches 0 through 7 control the LEDs as expected.
3. You can pause the program by clicking on the **pause** button in the toolbar.
4. You can also single step through your code. You will generally want to use the **step over** button, unless you want to drill down into the code of a function.
5. Play around with these features, noticing that the switches no longer operate to control the LEDs when the code is not running.