CPET-561 Embedded Systems Design 1

Lab1

Signoffs and Grade:

Name:____

Component	Signoff	Date	Time
Blinking LED –VHDL only			
Switches control LEDs			

===

Component	Received	Possible
Signoffs		100
Penalties • after the first 15 minutes of lab session 2: -10 • after the first 15 minutes of lab session 3: -25 no signoffs after the first 15 minutes of lab session 4:	-	
Total		100

Introduction: An embedded System on a Programmable Chip (SOPC) consists of a microprocessor and peripheral hardware modules implemented on a Field Programmable Gate Array (FPGA). For CPET-561, the Altera Cyclone V FPGA on the DE1-SoC board will be used for labs and activities. Altera provides a suite of design and verification tools to simplify the development of embedded systems on the Cyclone V.

Objective: The purpose of this lab is to introduce students to the Altera tools they will be using throughout the semester to design and verify embedded systems. At the completion of this tutorial, students will be able to:

- Use the Platform Designer tool to design a Nios II-based system
- Integrate the Nios II system into a Quartus Prime project
- Implement the designed system on the DE1-SoC board
- Run an application on the Nios II processor

Procedure:

The system for this tutorial is shown in Figure1. The system realizes a trivial task. Eight toggle switches on the DE1 board, SW7−0, are used to turn on or off the eight red LEDs, LEDR 7−0. The switches are connected to the Nios II Software Build Tools by means of a parallel I/O configured to act as an input port. The LEDs are driven by the signals from another parallel I/O 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, Platform Designer, and Eclipse software tools will be used to design the hardware depicted in Figure1. 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.

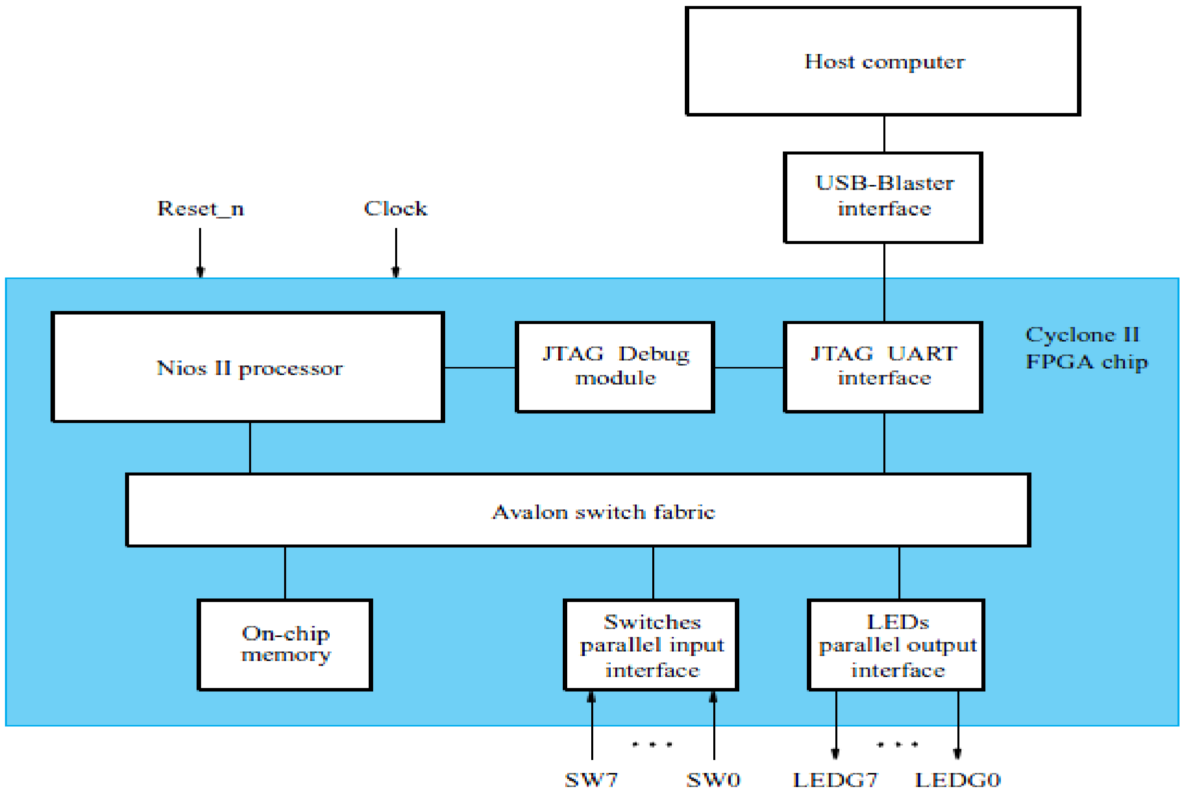


Figure 1: The simple Nios system that will be used for this lab.

Create Quartus Project

1. Create a directory named DE1_SoC_Lab1/part1. This directory will contain all of your work for this demo.
2. Start Quartus Prime 18.1.
3. Select File->New Project Wizard from the Quartus menu.
4. On the first page of the new project wizard, click on Next >
5. On the second page enter the following parameters:
    a. Working Directory = navigate to the DE1_SoC_Lab1/part1 directory
    b. Name of project = lightsNoNios
    c. Top level design = lightsNoNios
6. Click next to get to the Project Type page and select Empty Project.
7. 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…
    a. Family = Cyclone V
    b. Name Filter = **5CSEMA5F31C6**
    c. After you enter the above filter name, there should be only one device left in the list. Select this device and click next.
8. 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, put it in the project directory DE1_SoC_Lab1/part1.
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 lightsNoNios.vhd file that you added in step one above and select it.
4. 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 DE1_SoC.qsf file from myCourses. There will be many more pins assigned than needed.
2. Select Assignment->Import Assignments from the Quartus menus.
3. Navigate to the DE1_SoC.qsf file you downloaded in step 1.
4. Check “Copy Existing Assignments”.
5. Click OK.
6. **Note:** You can use this pin file for all of your designs provided your entity port names match exactly with the pin names in the file.

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. Fix any syntax/compilation errors and review all provided code for functional errors.
3. Verify that the process completed with no errors or latch warnings.

Program the Device

At this point you are ready to program the Cyclone V FPGA. To do this you will use the lightsNoNios.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. Turn on the board.
2. When the programmer application opens…
   a. Click Auto Detect
   b. Select the 5CSEMA5 from the devices presented in the pop-up menu.
   c. There should now be two devices shown in the JTAG chain.
   d. Click on **5CSEMA5** in the list of devices (not the processor).
   e. Click on “Change File”.
   f. Navigate to ../DE1_Soc_Lab1/part1/output_file/lightsNoNios.sof and select it. This file should now show up in the list next to that device.
   g. 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.
   h. Click Start. The progress bar at the type right should turn green and indicate success. Also the LED should now be blinking.

i. **Obtain a signoff from the lab instructor or TA.**

Create the Nios Processor

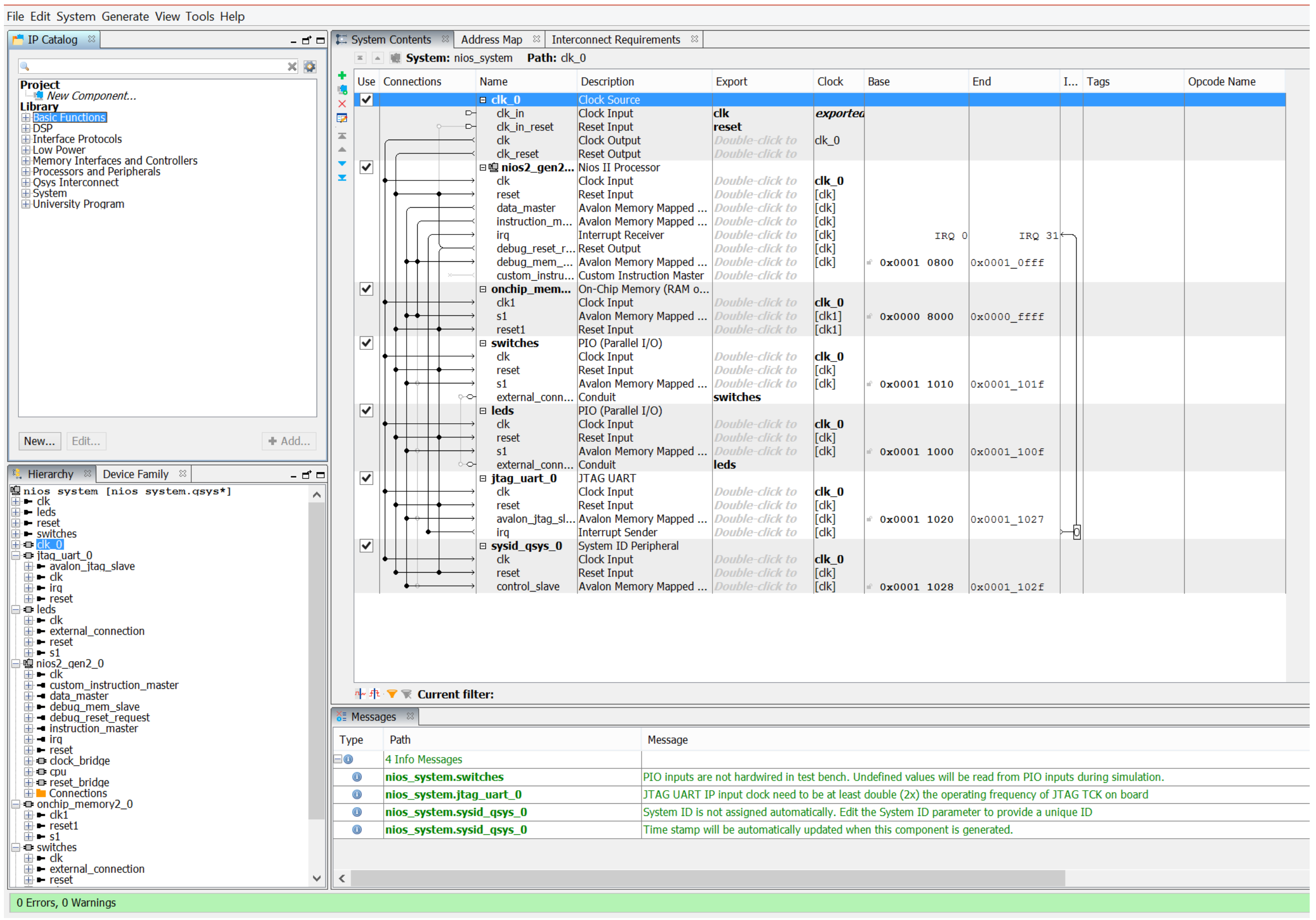
Platform Designer is a tool used in conjunction with the Quartus software. It allows the user to easily create a system based on the Nios II processor, by simply selecting the desired functional units and specifying their parameters.

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 Platform Designer software will then generate all the VHDL code and other files required to implement our processor system.

To implement the system in Figure 1, 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 Platform Designer, follow the following steps. When you have completed all the steps, the **System Contents** tab of Platform Designer should look like the one shown in Figure .



System: nios_system **Path:** clk_0

Use	Connections	Name	Description	Export	Clock	Base	End	I...	Tags	Opcode Name
✓		**clk_0**	Clock Source							
		clk_in	Clock Input	**clk**	***exported***					
		clk_in_reset	Reset Input	**reset**						
		clk	Clock Output	*Double-click to*	clk_0					
		clk_reset	Reset Output	*Double-click to*						
✓		**nios2_gen2...**	Nios II Processor							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		data_master	Avalon Memory Mapped ...	*Double-click to*	[clk]					
		instruction_m...	Avalon Memory Mapped ...	*Double-click to*	[clk]					
		irq	Interrupt Receiver	*Double-click to*	[clk]	IRQ 0	IRQ 31			
		debug_reset_r...	Reset Output	*Double-click to*	[clk]					
		debug_mem_...	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0001 0800	0x0001_0fff			
		custom_instru...	Custom Instruction Master	*Double-click to*						
✓		**onchip_mem...**	On-Chip Memory (RAM o...							
		clk1	Clock Input	*Double-click to*	**clk_0**					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk1]	0x0000 8000	0x0000_ffff			
		reset1	Reset Input	*Double-click to*	[clk1]					
✓		**switches**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0001 1010	0x0001_101f			
		external_conn...	Conduit	**switches**						
✓		**leds**	PIO (Parallel I/O)							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		s1	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0001 1000	0x0001_100f			
		external_conn...	Conduit	**leds**						
✓		**jtag_uart_0**	JTAG UART							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		avalon_jtag_sl...	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0001 1020	0x0001_1027			
		irq	Interrupt Sender	*Double-click to*	[clk]			0		
✓		**sysid_qsys_0**	System ID Peripheral							
		clk	Clock Input	*Double-click to*	**clk_0**					
		reset	Reset Input	*Double-click to*	[clk]					
		control_slave	Avalon Memory Mapped ...	*Double-click to*	[clk]	0x0001 1028	0x0001_102f			

Type	Path	Message
	4 Info Messages	
●	**nios_system.switches**	PIO inputs are not hardwired in test bench. Undefined values will be read from PIO inputs during simulation.
●	**nios_system.jtag_uart_0**	JTAG UART IP input clock need to be at least double (2x) the operating frequency of JTAG TCK on board
●	**nios_system.sysid_qsys_0**	System ID is not assigned automatically. Edit the System ID parameter to provide a unique ID
●	**nios_system.sysid_qsys_0**	Time stamp will be automatically updated when this component is generated.

0 Errors, 0 Warnings

Figure 2: The finished system after all the following steps are completed.

1. Select Tools > Platform Designer, from the Quartus menus. This will bring up the Platform Designer 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 Platform Designer menu and enter nios_system as the system name.
3. The first step in building a system is to specify the processor as follows:
    a. On the left side of Platform Designer, 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.
    b. Choose Nios II/e processor, which is the simplest version. Click Finish to return to the main Platform Designer window, which now shows the Nios II processor specified. There may be some warnings or error messages displayed in the Platform Designer 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:
    a. Select Basic Functions->On Chip Memory->On Chip Memory (RAM or ROM) and click Add
    b. 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).
    c. Do not change the other default settings
    d. Click Finish, which returns to the Platform Designer 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.
    a. Select Processors & Peripherals->Peripherals->PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard.
    b. Specify the width of the port to be 8 bits and choose the direction of the port to be Input.
    c. Click Finish to return to the Platform Designer 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.
    a. Select Processors & Peripherals->Peripherals->PIO (Parallel I/O) and click Add to reach the PIO Configuration Wizard again
    b. Specify the width of the port to be 8 bits and choose the direction of the port to be Output
    c. Click Finish, which returns to the Platform Designer 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:
    a. Select Interface Protocols > Serial > JTAG UART and click Add to reach the JTAG UART Configuration Wizard.

   b. Do not change the default settings.
   c. Click Finish, which returns to the Platform Designer 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.
   a. Select Basic Functions->Simulation; Debug & Verification->Debug & Performance->System ID Peripheral and click Add to reach the System ID Peripheral Wizard.
   b. 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.
   c. Click Finish, which returns to the Platform Designer application.
9. Platform Designer 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 Platform Designer 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, <u>except</u> for the clk_0 component. To allow this, we need to connect the **debug_reset_request** output port of the **nios2_gen2_0** to all the resets, <u>except</u> the reset into the clk_0 component.
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. Connect the Nios II (**nios2_gen2_0**) 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. 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.

a. Right-click on the Nios II Processor component and then select Edit or double click on Nios II Processor component (**nios2_gen2_0**).
b. Find the vectors tab and select **onchip_memory2_0.s1 in the** drop-down for both the reset vector and the exception vector.
c. Do **not** change any other settings.

16. 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 Platform Designer. These are the addresses you will use in your c-code or assembly code to communicate with the various peripherals. To have Platform Designer automatically assign these addresses, select the command System > Assign Base Addresses from the Platform Designer menus. Notice this removes the errors about overlapping addresses.
17. We are going to instantiate the system we designed in Platform Designer 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.
    a. Click on the **external connection** port of the LEDs PIO (Parallel IO) component.
    b. Double click on **Double Click to Export**, and enter the name **leds**.
    c. Click on the **external connection** port of the switches PIO (Parallel IO) component.
    d. Double click on **Double Click to Export**, and enter the name **switches**.
18. 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). Click on the open connector circle in the IRQ column. A zero will appear saying that the interrupt will be connected to the Nios II processor's IRQ0 signal.
19. Your completed system should now look like 2. The last step is to tell Platform Designer 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 Platform Designer window. Change the synthesis drop-down box to VHDL and generate your system. Platform Designer will now generate all the VHDL code to implement your system. 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. Create a new project in DE1_SoC_Lab1/part2 and download and add the lightWithNios.vhd file from myCourses.
2. NOTE: The provided VHDL file will have VHDL errors in it that you will need to fix to get this project to work! Be sure you review the ENTIRE VHDL file for syntax AND conceptual mistakes.
3. Once you have created your new project and updated the VHDL, under the Quartus menu item Project->Add/Remove Files add the processor you just created from DE1_SoC_Lab1/nios_system/synthesis/nios_system.qip.

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. If the programmer app is no longer running, then you may need to repeat the procedure previously outlined in the section "Program the Device". 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 /DE1_SoC_Lab1/part2/software.
2. From Quartus, select Tools->Nios II Software Build Tools for Eclipse.
3. In the window that pops up, navigate to /DE1_SoC_Lab1/part2/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 DE1_SoC_Lab1/part2 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 creation 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 DE1_SoC_Lab1/part2/software/nios_software directory.
10. Copy the file /DE1_SoC_Lab1/part2/software/nios_software_bsp/system.h to the directory /DE1_SoC_Lab1/part2/software/nios_software/.
11. Right click on **nios_software_bsp** in the project explorer tab on the left side of the Eclipse screen. Select NIOSII -> Generate BSP.
12. 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.
13. Right click on **nios_software** in the project explorer tab on the left side of the Eclipse screen. Select Build Project from the menu.
14. 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 lightsWithNios.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 lightsWithNios.sof.

15. Returning now to the Eclipse application, select Run->Debug Configurations from the menu. This will open the Debug Configurations window.
16. 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.
17. In the **project** tab of the new configuration, in the **project name** drop-down, select **nios_software**.
18. Click on the **Debug** button.
19. Click **Yes** to open the debug session.

NOTE: If you have a SysID error preventing you from running the debugger, recheck the VHDL file for errors. Specifically review the clock and reset connectivity.

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.
6. **Obtain a signoff from the lab instructor or TA.**