University of Massachusetts Amherst
Computer Systems Lab 1 (ECE 354)

LAB 1 Reference Manual

Lab 1: Using NIOS II processor for code execution on FPGA

Objectives:

1. Understand the typical design flow in an Embedded System design as illustrated by the Altera CAD tools: Quartus Prime, Qsys, NIOS II IDE.
2. Understand differences between traditional microcontrollers and soft core processors.
3. Learn how to use on chip memory and SDRAM in your hardware designs.
4. Gain more insight into hardware and software development flow using Quartus Prime, assigning pins of FPGA etc.

This lab consists of two parts. First you will **echo "Hello from NIOS II"** on your PC screen in the NIOS II IDE console window. Next, you will **implement a counter program** and display the value of count on LEDs or the 7-segment hexadecimal displays on DE1-SoC.

These labs will primarily teach you the use of System on Programmable Chip Development tool (Qsys). Before starting this lab, it is **mandatory** that you read the Quartus II introduction PDF document (available as a resource file) which describes a typical Quartus Prime design. Use this document to refresh your understanding in: Creating a new project, compiling it, assigning pins to FPGA and programming the FPGA.

Part 1: Hello from NIOS II.

1. Create a new project in Quartus Prime, taking care to select the proper FPGA (**5CSEMA5F31C6**). Do not add any files at this stage to project. Please store the project in a new directory. Also, note that the path names shouldn't have any spaces in them.

2. Go to Tools -> Qsys. Save the Qsys system with the same name as the top-level entity name.

3. On the left-hand side of screen, you will find several hardware components present under the "IP Catalog" tab. These components are available to you for use. Double clicking on any individual component adds it your hardware. We will be adding various components to design our system (You can search for components by just typing their names). Note that the Clock for our system is automatically added initially. You can rename the module if you wish (Don't include spaces).

4. To add processor, under embedded processors tab double click on NIOS II processor. In the window, which opens, you will see two flavors, resource optimized (NIOS II/e) and performance optimized (NIOS II/f) processors. Select NIOS II/e for this project as the task at hand is quite trivial.

5. The System Contents tab now contains NIOS II processor. Observe that the Qsys automatically assigned a base address to it. Also, you will see that IRQ's are also determined automatically by the Qsys.

   Note: As you add components and make connections in your Qsys system, error and warning messages appear in the Qsys Messages tab, indicating steps that you must perform before the system is complete. Some error messages appear between steps and are not resolved immediately; as you progress through the tutorial, errors are resolved, and the error messages disappear.

6. Next, we will add some On-Chip memory to store our program. This memory will be instantiated on the FPGA. To do this, select On-Chip Memory (RAM or ROM) present under the Memory components.

You will first have to expand the memory tab to see what other components are present under it. Use 20 KB of On-Chip memory. Leave the other settings unaltered and click on finish.

7. Next, we will add a JTAG UART which is needed to talk to the DE1-SoC by the PC. This is present under the communications components. Click on finish in the pop up box and leave all the settings unaltered. Add the Interval Timer from the IP catalog. NIOS II HAL (Hardware Abstraction layer) requires a timer for creating clock tick which can be used to generate interrupts. However, this is optional for this part.

8. Now you will need to connect the components. The Clock Output (clk) and Reset Output (clk_reset) of the system clock needs to be connected to Clock Input and Reset Input of other components. To connect the Clock Output to the Clock Input of the NIOS II, click to fill the connection dot between them under the connections tab. Similarly connect all the clock and reset interfaces. Now connect data_master of the NIOS II to the Avalon slave (s1) interfaces of the components. Connect the instruction_master to the Slave of On-Chip memory as it stores the program code.

9. Select On-Chip memory for "Reset vector" and "Exception vector" in the vectors tab of the NIOS II processor. Also, select Auto-assign base address under System tab to remove any errors due to base address. Assign the interrupt numbers to the components which support interrupts. To do this click on the dot under the IRQ tab of respective components. Assign highest priority to Interval Timer.

10. You are now ready to generate the system, click on the "generate" button on the bottom of the screen. This operation will take a few seconds to complete.

11. After the system is generated, go back to Quartus Prime where your project is still open. Do not close the QSYS builder window.

12. We will now work in Quartus Prime for some time. The Qsys creates a QIP file for the generated system which contains the Verilog files for all the system components. In the project Navigator, go to Files, right click to find Add/remove files in project. Then browse for the QIP file in the project directory, add it.

13. Click on "Start Analysis and Synthesis" button present on the task bar where you also have the buttons for "Start Compilation", "Settings", "Assignment Editor" etc.

14. Next, use the assignment editor for performing pin assignments to your design. You must read the pin assignment section of the document from Altera which explains it in detail (The link is given at the beginning of this document). Assign any toggle switch to the reset_n pin and assign the 50 MHz clock to clk pin.

15. Now do a full compilation by clicking the "Start Compilation" button on the task bar.

16. Program the FPGA with the SOF file contained in our project directory. To do this, click on the "Programmer" button on the taskbar. A pop up tells you about the time limited feature of our design. Click on OK. Next, check the program/configure option in the programmer window and click on start. Make sure that you have USB blaster installed and you are using JTAG mode. DO NOT click on the cancel button in the "time remaining" pop up window which appears after programming is successful.

17. After programming the FPGA, go to start menu and find NIOS II Software Build Tools for Eclipse and start.

18. You will see that NIOS II IDE is launched and you are asked to specify a workspace. You can use this as your default workspace or specify another folder depending on your convenience. Go to file->new-> NIOS II Application and BSP from template to create a new program which we will run on our NIOS.

19. The "New Project" window presents you with several options. You can select any template from the ones present or create a new one. We will be using the default Hello World template which is already selected. This program in C has already been written and we will be running it on our board. We also need to specify the Qsys system which is a sopcinfo file the NIOS II toolchain uses to determine the hardware present. Browse for the sopcinfo file from the project directory. Select the **Hello World small** template.

20. Go to run-> run as->NIOS II hardware. Make sure that your reset switch is logic HIGH (Reset_n is active when it is low) otherwise your processor will be reset all the time. You should see "Hello from NIOS II" on your screen after sometime if everything is properly done.

To make your life simpler, you must note the following:

1. Give the same name for project in Quartus Prime and the system in Qsys.
2. You can add the components in Qsys in any order, it does not matter.
3. The pins must be assigned properly. Note what is logic High and Low for switches. Refer to DE1-SoC manual.
4. Specify the correct Qsys system and project template in NIOS II IDE.
5. If you don't see things working out, it is a good idea to switch the workspace in the NIOS II IDE.

After completing this part of the first lab, you are ready to design a more complex system.

Part 2: Implementing a counter and directing its output to LEDs and Seven Segment Displays.

In this lab, you have to implement a counter using a C program called "count binary" which is available in the NIOS II IDE as a template (like Hello world in previous part). The primary task is to create a Qsys system which can support this program, which includes SDRAM. This program generates a counter capable of counting from 0x00 to 0xFF on NIOS II. The user can see the count value either on the two Seven-segment displays or the 8 red LEDs or on both simultaneously. The selection of display is made by pressing on of the three blue keys which you see on DE1-SoC board per the following rule:

I. Key 0 pressed: LEDs display the count value in binary.
II. Key 1 pressed: Seven-segment displays show the count value in hexadecimal.
III. Key 2 pressed: All the above show count value.

To complete this lab, you are given some general guidelines and I assume that you have completed part 1 successfully. Before starting this part of the lab, try to understand some sections of the count binary program which is given as a template in NIOS II. Pay particular attention to the way the input is accepted by the switches.

1. In your hardware using QSYS system, include the following components: NIOS II, JTAG UART, Interval Timer as in part 1. You can optionally include System ID (SysID) for debugging. Remove the clock which is already there by default.
2. Include a PLL to provide the clock for your design. You can find it under University Program -> Clock -> System and SDRAM Clocks for DE series Boards. Export the Clock input and reset inputs by double clicking the respective fields under the Export tab. SDRAM requires a different clock frequency to operate than what is required by the NIOS II. Hence it is necessary to include the PLL which provides the required clock frequencies for the SDRAM and the NIOS II processor.

3. You'll be using external off chip SDRAM for both instruction and data memory for this part instead of the on-chip memory. SDRAM requires a controller which generates the control signals for its operation.

- Add SDRAM to System Control by going to the System Contents Tab > Memories and Memory Controllers > External Memory Interfaces > SDRAM Interfaces > SDRAM Controller.
- There are different settings to control SDRAM depending on the chip model available. We want to create a generic 8 MB chip in the SDRAM controller wizard, change bits to 16. Change the address width row to 13, column to 10.
- Leave timing to default settings and click Finish.
- Make the necessary connections to the controller.
- Modify the CPU reset and exception vectors to use SDRAM memory. Auto assign memory addresses, System > Assign Base Addresses

4. LEDs, Hexadecimal Seven Segment Displays and Keys are examples of parallel input output hardware, which can be found under "other" components. You will have to include parallel I/O thrice, one for LED, one for the Keys and another for the Seven Segment Display. While adding each parallel I/O, you can specify their width and whether they are input, output or both. There are other options for input type I/Os which let you control their triggering nature (edge vs level) and let you decide their interrupt generation. Think of how many LEDs you need to display the binary count value. You will have to instantiate the same number of LEDs, one for each bit. Hence the width of parallel I/O should be equal to the number of LEDs needed, one output for each bit. These parallel I/O will be connected to the actual LEDs by pin assignments later. In a similar manner, include another set of 4 parallel I/Os for keys which the user will press. Lastly, include support for Seven Segment Display by using a third set of parallel I/Os. You have to figure out the width in this case by reading the DE1-SoC manual which describes the 7 Segment Displays. Note that you will need two seven segment displays to show the count value. While using these I/Os, you may want to read the program **count binary** located in the Quartus installation more carefully to determine any special needs they might have.

5. Read the requirements given in the count binary program for the hardware. It expects the inputs and outputs to be named in a certain way. Rename the components in your hardware by right clicking them (after you have included them) and selecting rename. Generate the QSYS system. After that add the QIP file to the project.

6. After generating the hardware, you must follow an approach which is different from what you did in previous part. Also, pin assignment using Assignment editor is a tedious and error prone process for a large design. Quartus allows users to import the pin assignment file (QSP). The QSP file contains the pin names and their corresponding pins on the FPGA board. We need to write a top-level Verilog module which assigns the QSYS module pins to the pin names as that followed in the QSP file. The file will be provided to you but it is advisable that the students go through it.

7. Set the pin assignment HDL module as the top-level module. Go to assignments->import assignments. Browse to the DE1_SoC folder on your PC and change the type of file to QSP format. You will find a file called DE1_SoC.qsf. Select it and press OK.

8. After completing "Analysis and synthesis", you can go to the assignment editor and verify that all the pins have been connected properly. Finally do a full compilation and program the FPGA by the SOF file.

9. Go to NIOS II IDE and use the QSYS system from this project directory. Select the count binary template and run the program on NIOS II Hardware. You do not need to limit the size of executable this time as you have lot of SDRAM with you (How much?). See how the count value display unit changes when you press the appropriate keys. **Note**: When you run the count binary program, the seven-segment display shows some gibberish. The reason being that the program has been written for a different board other than the DE1-SoC. Hence, the seven-segment decode table in the count binary template needs to be replaced with that of DE1-SoC. Replace the seven-segment decode table with this [{0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7C, 0x07, 0x7F, 0x67, /* 0-9 */ 0x77, 0x7C, 0x39, 0x5E, 0x79, 0x71}; /* a-f */]

10. Now you need to modify the count binary program to do the following: Add the digits of your individual student IDs. Then add the sums you obtained, let's say you got X as the result after adding the sums. The

next step is to do X mod 100, let’s say Y is the final result. You have to make the DE1-SoC count to Y in one minute. And you need to count on the Seven Segments in decimal and on LEDs in binary. For example, say the student IDs of two members in a group are 21435101 and 23013416. Now 2+1+4+3+5+1+0+1=17 and 2+3+0+1+3+4+1+6=20, adding both the sums we get 17+20=37. Then 37 mod 100 results in 37. Hence this group would display count from 00 to 37 in decimal on the Seven Segment Display and on LEDs (in binary of course), all in approximately one minute. **Note**: We are not going to use the reset button for part II, so set it permanently to VCC.