Running µC/OS-II on Nios II Systems

Objective: This lab presents an introduction to build µC/OS-II applications for Intel Nios II processor. Through the lab exercises, you will learn how a program written for µC/OS-II can be executed on an Intel FPGA FPGA device that includes a Nios II processor.

The µC/OS-II port for Nios II is distributed as part of Nios II Embedded Design Suite (EDS). Nios II EDS can be used to compile, debug, load and run µC/OS-II programs. Most of µC/OS-II configurations can also be modified in a Nios II project using Board Support Package (BSP) Editor.

The hardware to be used in this lab is the prebuilt DE2-115 Computer. Quartus Programmer is used to download the DE2-115 Computer system to a DE-series board.

Resources: In order to carry out this lab, you will require the following resources. More details of the software resources to be used in the lab can be found on Intel FPGA website and the class webpage associated with this lab.

1. A computer running Linux, preferably Redhat Enterprise Linux (RHEL) 6.8 or 7.3. Any of the computers provided in the Computer Engineering Labs can be used for carrying out the work associated with this lab.
2. Intel FPGA Development and evaluation board: The process of downloading and debugging a Nios II program requires the presence of an FPGA board to implement the Nios II hardware system. Intel FPGA DE2-115 Development and Education board has been used to run this lab. Other DE-series boards (DE1-SoC, DE2, etc.) can also be used.
3. Intel FPGA Quartus Prime: The standard edition of Quartus Prime has been used to run this lab.
4. Intel FPGA Nios Embedded Design Suite (EDS): Nios II EDS is released as part of Quartus Prime software.

Preparation:

1. Review DE2-115 Computer system
2. Review µC/OS-II

Procedure:

This section describes the specific steps you are to take in carrying out this lab. Your mark will be based on the contents of your lab report and the evaluations of your in-lab work by the person(s) running the lab session. Be complete in your lab report: include the output produced by the run steps (either by hand-written entries or by printing) and clearly write all required analyses and modifications. Submit your lab report to the "ucos2nios2_lab" hand-in folder through Black Board/PAWS course tools.

Part I: Getting started with µC/OS-II port on Altera Nios II processor

Part I of this lab provides step-by-step instructions that illustrate the process of building and running a µC/OS-II application on the pre-built Computer system that includes a Nios II processor. First of all, the pre-built system hardware must be downloaded to the FPGA on DE2-115 board. Instead of using Altera Monitor Program, Quartus II Programmer will be used. Secondly, a µC/OS-II software image must be created for the Nios II processor system. This process involves three major steps: create a new Nios II SBT for Eclipse project; configure the Nios II board support package (BSP), and build the Nios II software

project. The example used for this lab is a simple C program that consists of two periodic tasks.

1. Obtain a suitable computer running Redhat Enterprise Linux (RHEL), preferably one in the Computer Engineering Labs. If you use a computer other than those provided in the lab, you may have to alter the paths of source and program files.
2. Locate the Computer_Systems folder under Intel FPGA University Program Design Suite (UPDS) folder.
3. In the Computer_Systems, there are pre-built systems for different DE2-series boards. Copy DE2-115/DE2-115_Computer/Verilog folder to your home directory $HOME/Micrium/DE2-115_Computer/Verilog. More details will be provided during the lab session.
4. Under $HOME/Micrium/DE2-115_Computer/Verilog, make sure there are files named DE2_115_Computer.sof and nios_system.sopcinfo.

Every Nios II software project needs a system description of the corresponding Nios II hardware system. For the Nios II SBT for Eclipse, this system description is contained in a .sopcinfo file. A .sof file contains the data to configure Intel FPGA devices.

5. Start Nios II Software Build Tools (SBT): The Nios II SBT for Eclipse is an easy-to-use GUI that automates build and makefile management, and integrates a text editor, debugger, the Nios II flash programmer, and the Quartus Programmer. Software application templates included in the GUI make it easy for new software programmers to get started quickly.
6. If Workspace Launcher dialog box pops up, set workspace to a preferred location such as $HOME\workspace or click OK to accept the default location.
7. Download the Computer system hardware to DE2-115 board.
   - Ensure DE2-115 board is connected to the host PC and powered on.
   - Ensure RUN/PROG switch is in RUN position
   - In Nios II SBT, select **`Nios II> Quartus Prime Programmer`**
   - In Quartus Prime Programmer window, ensure download cable is set to USB-Blaster [USB-0]. Click **Hardware Setup** to select the download cable if necessary.
   - Ensure **Mode** is set to JTAG.
   - Click **Add File…** or **Change File…** to select DE2_Computer.sof as the download file.
   - Ensure Program/Configure is checked.
   - Click **Start** to configure FPGA.
   - Wait until the Progress meter sweeps to 100%. Ensure download is successful.
   - Exit Quartus Prime Programmer. If the Quartus Prime Programmer asks if you want to save changes to the chain1.cdf file, click No.
8. Create a new application software and board support package (BSP)
   - Select **`File> New> Nios II Application and BSP from Template`**.
   - Under **Target hardware information**, browse to $HOME/Micrium/DE2-115_Computer/verilog and select Computer_System.sopcinfo for SOPC Information File name.
   - Ensure CPU name is set to **Nios2**.
   - In the Project name box, type ucos2nios2 as the name of your project.

- Ensure **Use default location** is selected. Also, note down the default location in your lab report. If the path of your project location is partially displayed, uncheck **Use default location** to see the complete path.
- Select **Hello MicroC/OS-II** from the Project Templates list.
- Click **Next**.

9. Select a Board Support Package (BSP).
- Ensure Create a new BSP project based on the application project template is selected.
- Project name should be automatically set to ucos2nios2_bsp. Use this default value.
- Ensure **Use default location** is selected. Note down the default location in your lab report.
- Click **Finish**.
- Ensure **ucos2nios2** and **ucos2nios2_bsp** are successfully created and displayed in Project Explorer.

10. In the Project Explorer view, expand ucos2nios2 and double click hello_ucosii.c to view the source code. NOTE: The tasks are created using OSTaskCreateExt(). Look up the µC/OS-II manual for the usage of this function.

11. Build the Hello MicroC/OS-II program.
- In Project Explore, right click ucos2nios2 and select **Build Project**.
  When compilation completes, the message "Build Finished" appears in the Console view.

12. Run Hello MicroC/OS-II program
- To run the program, right click ucos2nios2 in the Project Explorer view and select **Run As> Nios II Hardware** to download the program to DE2-115 board.
  If **Run Configurations** dialog box appears, click the **Target Connection** tab. Then click **Refresh Connections** and **Apply** until a board connection establishes. Once the board connection is established, click **Run**.
- Ensure the program is running as expected.

13. Debug the application
- Right click ucos2nios2 in Project Explorer, point to Debug As, and select **Nios II Hardware.**
- If the Confirm Perspective Switch message box appears, click Yes.
  The context will be changed to the Nios II Debug perspective. After a moment, the main() function appears in the editor. A blue arrow next to the first line of code indicates that execution stopped at that line.
  To return to the default project perspective from the debug perspective, click ">>" in the top right corner and select the corresponding perspective.
- In the Debug windows, click "resume" button to resume execution.
  When debugging a project in the Nios II SBT for Eclipse, you can pause, stop, or single step the program, set breakpoints, examine variables, and perform many other common debugging tasks.

14. Edit and rerun the program
- Make necessary changes to hello_ucosii.c so that task1() displays "<*tick*>: Hello from task 1" and task2() displays "<*tick*>: Hello from task2" where <*tick*> is the actual tick (i.e. OSTime).
- Save the file.
- Right click ucos2nios2 and select **Run As> Nios II Hardware.**

NOTE: There is no need to build the project manually. Nios II SBT for Eclipse automatically rebuilds the program before downloading it to the FPGA.

15. BSP and µC/OS-II configuration
- In Project Explorer view, right click ucos2nios2_bsp and select Nios II> BSP Editor…
- In BSP Editor, expand Advanced> ucosii to see the default settings for µC/OS-II.
- Note down the default settings for maximum number of tasks and the lowest priority in your lab report. Also ensure semaphore, mailbox and message queue are all enabled.
- In the project BSP folder (i.e. the default location that you noted down in Step 9), search the configuration file (i.e. os_cfg.h) for µC/OS-II to find where the settings for µC/OS-II are stored. In your lab report, note down the name and location of the file that stores µC/OS-II settings.
- Find out the file where ticks per second is defined, and note down the filename and the actual value of OS_TICKS_PER_SEC in your lab report.

Part II: µC/OS-II Semaphores

In Part I, the default Hello MicroC/OS-II project template was used to easily and quickly create a BSP based on µC/OS-II. In Part II of this lab, you are to write a µC/OS-II application that implement similar functions that were developed in the earlier lab exercises. NOTE: Interrupts and delay loops should not be used in this lab exercise.

1. Create a task named TaskScanKey() to read KEY_{0-3}. At least one semaphore must be used to control the access to the KEY pressing information since the information is also used by TaskCounter() and TaskStopwatch(). NOTE: Use non-blocking semaphore APIs if multiple semaphores are used in a task.

2. Create a task named TaskCounter() to implement a single-digit decimal counter similar to the one that was implemented in Part II of Lab 1.
- Initially the counter should be 0.
- If counter value is less than 9, increment the counter when KEY_1 is pressed.
- If counter value is greater than 0, decrement the counter when KEY_2 is pressed.
- Reset the counter to 0 when KEY_3 is pressed.
- KEY pressing information should be obtained from TaskScanKey() through global variables.

3. Create a task named TaskStopwatch() to implement a stopwatch similar to the one that was implemented in Part II of Lab 2.
- Initially the stopwatch should be 0.
- Toggle between RUN and STOP when KEY_0 is pressed.
- If the stopwatch is in the RUN state, calculate elapsed time in minutes (MM) and seconds (SS).
- The elapsed time rolls back to 00:00 after reaching 59:59.
- If the stopwatch is in the STOP state, reset the elapsed time to 0 when KEY_3 is pressed.
- Achieve the best accuracy possible allowed by the DE2-115 Computer. In your lab report, explain how the best accuracy is achieved.
- KEY pressing information should be obtained from TaskScanKey() through global variables.

4. Create a task named TaskDispTime() to display the counter value from TaskCounter() and the elapsed time from TaskUpdateTime().
- Display the counter value on HEX0.

- Display the elapsed time on HEX7-4 in the format of MM:SS, where HEX7-6 display MM and HEX5-4 display SS.
- At least one semaphore must be used to control the access to HEX display, the counter value, and the elapsed time. NOTE: Use non-blocking semaphore APIs if multiple semaphores are used in a task.

5. In your lab report, justify your decision on the priority and OSTimeDly (or OSTimeDlyHMSM) for each task.
6. Compile, download, and test your program.
7. Compress your source code into one zip file named <nsid>_ucos2nios2_part2.zip. Hand in the zip file to the "ucos2nios2_lab" hand-in folder through Black Board course tools.

Part III: µC/OS-II Mailboxes

Part III of this lab is to practice intertask communications using mailboxes. Modify the program as follows.

1. TaskDispTime() displays the counter value from TaskCounter() and the elapsed time from TaskUpdateTime() on the **LCD display**.
- Display the elapsed time on the first row of the LCD display in the format of MM:SS.
- Display the counter value on the second row of the LCD display.
- The elapsed time must be received from TaskStopwatch() through a mailbox named MboxTime
- The counter value must be received from TaskCounter() through a mailbox named MboxCounter.

2. Modify TaskStopwatch() to send the elapsed time to TaskDispTime() through MboxTime.
3. Modify TaskCounter() to send the counter value to TaskDispTime() through MboxCounter.
4. Compile, download, and test your program.
5. Compress your source code into one zip file named <nsid>_ucos2nios2_part3.zip. Hand in the zip file to the "ucos2nios2_lab" hand-in folder through Black Board course tools.