Embedded Systems
Laboratory Exercise 1

Getting Started with Linux*

This is an introductory exercise in using a Linux* operating system on Intel's Cyclone® V SoC devices. This discussion assumes that the reader is using the DE1-SoC development and education board, but the lab instructions apply equally well if other boards are being used, such as the DE10-Standard or DE10-Nano. The DE-series boards are described on the *FPGAcademy.org* website. Linux runs on the ARM* processor that is part of the Cyclone V SoC device. During the operating system boot process, Linux programs the Cyclone V FPGA with the *DE1-SoC Computer* system (or similar system for other boards). This system instantiates components inside the FPGA to make it easy to use the peripherals built into the DE1-SoC board, such as the LEDs, seven-segment displays, switches, and pushbuttons. For a detailed description of the computer system, please refer to a document such as *DE1-SoC Computer System with ARM*, available as part of the *Computer Organization* materials on *FPGAcademy.org*.

Part I

Read and complete Sections 1, 2, and 3 of the tutorial *Using Linux on DE-series Boards*. This tutorial is available as part of the *Embedded Systems* material on *FPGAcademy.org*. These sections will guide you through setting up the Linux microSD card, running the Linux OS on the DE-series board, and communicating with the board from a host computer.

Part II

Section 3.3 of the *Using Linux on DE-series Boards* tutorial involves implementing a program that increments the LEDR lights on the DE-series board. In this part you are to write another program that controls the LED lights in a different manner, described below.

Your program should turn on one LEDR light at a time. First, the rightmost light $LEDR_0$ should be on, then $LEDR_1$, then $LEDR_2$, and so on. When you get to the leftmost light $LEDR_9$, the direction should be reversed. Only one LEDR light is ever on at one time. The effect should be a single light sweeping from right-to-left, then left-to-right, and so on. Use a delay so that the light moves at some reasonable speed. To implement the delay, you can use a Linux system function, such as `nanosleep(...)`. Documentation for the `nanosleep` function can be found by typing `man nanosleep` in a Linux terminal, or by searching for it on the Internet.

Part III

Section 3.4 of the tutorial *Using Linux on DE-series Boards* shows how to implement a device driver using interrupts for the KEY pushbuttons on the DE-series board. In this part you are to extend the functionality of that device driver.

The existing KEY device driver initializes the LEDR port to $(1000000000)_2$, so that the leftmost light is on, and then increments this value whenever a KEY is pressed. You are to augment this code to display the count of pushbutton presses as a decimal digit, either on the seven-segment display *HEX0* or on the Linux Terminal window. You only need to display the decimal value corresponding to the four least-significant bits of the count, and wrap around to 0 when it exceeds 9. Always leave the leftmost LEDR light set to 1 as a visual indicator that the device driver is present in the kernel. The count values displayed on the LEDR port should cycle through 1000000000, 1000000001, 1000000010, 1000000011, . . . , 1000001001, 1000000000, etc., and the corresponding digits displayed should be 0, 1, 2, 3, . . . , 9, 0, respectively.

To display digits on the HEX0 seven-segment display, refer to the diagram given in Figure 1. It shows the *Data* registers in the parallel ports for displays HEX3−HEX0 and HEX5−HEX4. As shown in the figure, each segment in each display is connected to a specific data-register bit.

If you are using the Linux Terminal window to display digits, then you may want to make use of some of the Terminal window ASCII commands listed in Table 1. For example, the command `\e[2J` clears the Terminal window, and the command `\e[H` moves the Terminal *cursor* to the *home* position in the upper-left corner of the window. In these commands `\e` represents the `Escape` character. It can alternatively be specified by its ASCII code, for instance using the syntax `\033`. You can send such commands to the Terminal window by using the `printk` function. For example, the Terminal window can be cleared by calling

```
printk (KERN_ERR "\e[2J");        // clear Terminal window
```

Documentation for the *printk* library function can be found by searching on the Interet. Also, more Terminal window commands can be found by searching on the Internet for `VT100 escape codes`.

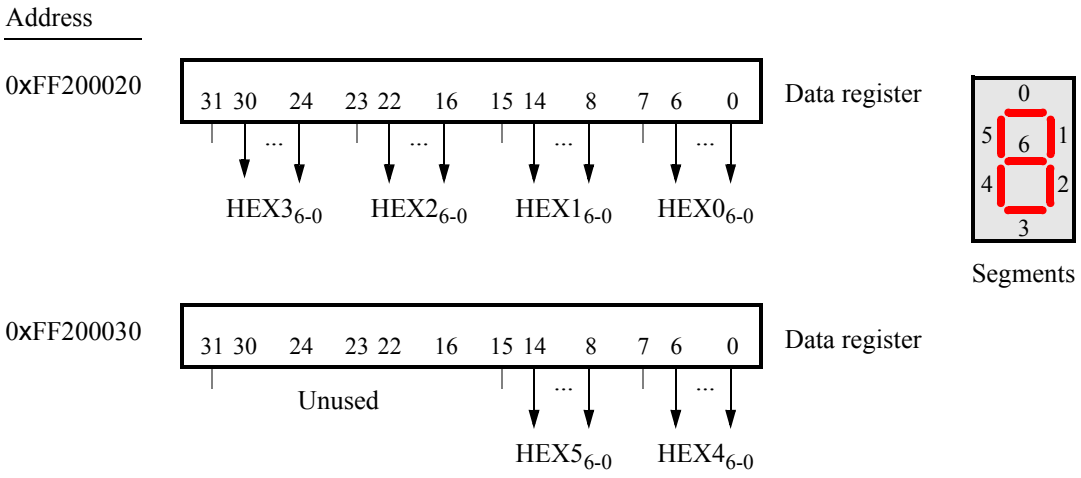

Figure 1: Diagram of the HEX display ports.

Table 1: Terminal window ASCII commands.

Command	**Result**
`\e[H`	move the cursor to the home position
`\e[?25l`	hide the cursor
`\e[?25h`	show the cursor
`\e[2J`	clear window
`\e[J`	clear window from cursor down
`\e[1J`	clear window from cursor up
`\e[ccm`	set foreground color to `cc`[1]
`\e[yy;xxH`	set cursor location to row `yy`, column `xx`

Note 1: `cc` values are 31 (red), 32 (green), 33 (yellow), 34 (blue), 35 (magenta), 36 (cyan), and 37 (white)