Embedded Systems
Laboratory Exercise 2

Developing Linux* Programs that Communicate with the FPGA

Part I

In Laboratory Exercise 1 you were asked to write a user-level program to control the LED lights on your DE-series board. Here you are to write another user-level program, which shows a *scrolling* message on a display. If you are using the Intel ® DE1-SoC or DE10-Standard boards, then you can use seven-segment displays to show the scrolling message. Your program should display the message Intel SoC FPGA and should scroll the message in the right-to-left direction across the displays. The letters in the message can be constructed as

The seven-segment display ports in the DE1-SoC and DE10-Standard computer systems are illustrated in Figure 1.

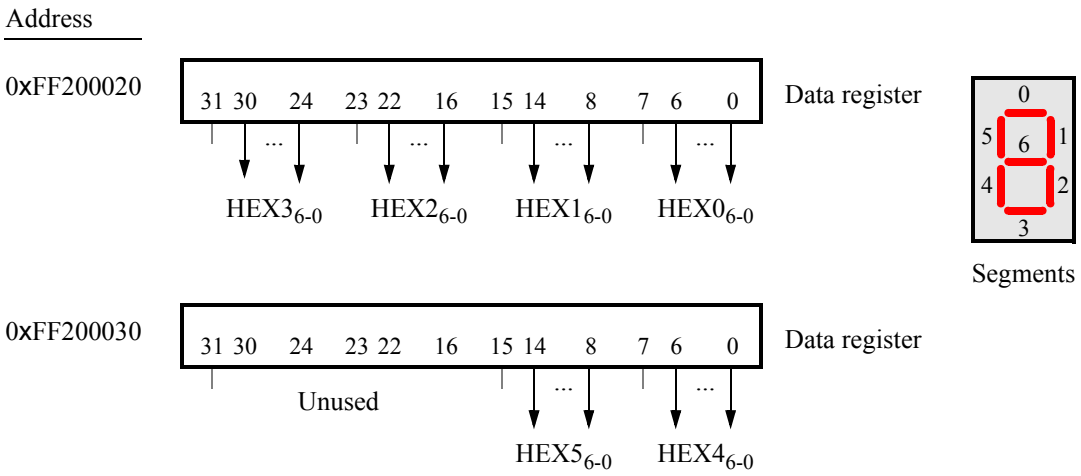

Figure 1: The seven-segment display ports.

As an alternative to seven-segment displays, you can use the Linux* Terminal window. In this case, you should designate a six-character space on the display in which to show the message. You can "draw" a box using ASCII characters as illustrated below:

```
 ------
|      |
 ------
```

If the message is scrolled inside of this box, the effect will be similar in appearance to using (six) seven-segment displays. Some Terminal window commands are 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, using the syntax `\033`. You can send such commands to the Terminal window by using the `printf` function. For example, the Terminal window can be cleared by calling

```
printf ("\e[2J");       // clear Terminal window
fflush (stdout);
```

Additional information about Linux Terminal window commands can be found by searching on the Internet for `VT100 escape codes`.

Table 1: Terminal window ASCII commands.

Command	**Result**
`\e7`	save cursor position and attributes
`\e8`	restore cursor position and attributes
`\e[H`	move the cursor to the home position
`\e[?25l`	hide the cursor
`\e[?25h`	show the cursor
`\e[2J`	clear window
`\e[ccm`	set foreground color to `cc`[1]
`\e[yy;xxH`	set cursor location to row `yy`, column `xx`

For both seven-segment displays and the Terminal window, use a delay when scrolling the message so that the letters shift to the left at a reasonable speed. To implement the required delay you can use a Linux library function such as `nanosleep`.

Perform the following:

1. Create a file called *part1.c* and type your C code into this file. Whether you are using seven-segment displays or the Terminal window, your code should be mostly the same. In one case, you write six characters at a time from the message to the seven-segment display ports, and in the other case you print these same characters (inside the box) on the Terminal window.

   You should provide the ability to pause or run the scrolling operation by using the pushbutton KEYs. The programming registers in a DE-series KEY port are illustrated in Figure 2. There is a *Data* register that reflects which KEY(s) is (are) pressed at a given time. For example, if KEY_0 is currently being pressed, then bit 0 of the data register will be 1, otherwise 0. The *Edgecapture* register can be used to check if a *KEY* has been pressed since last examined, even if it has since been released. If, for example, KEY_0 is pressed then bit 0 of the *Edgecapture* register becomes 1 and remains so even if KEY_0 is released. To reset the bit to 0, a program has to explicitly write the value 1 into this bit-position of the *Edgecapture* register. The *Interruptmask* register in Figure 2 is not used for this exercise, and can be ignored.

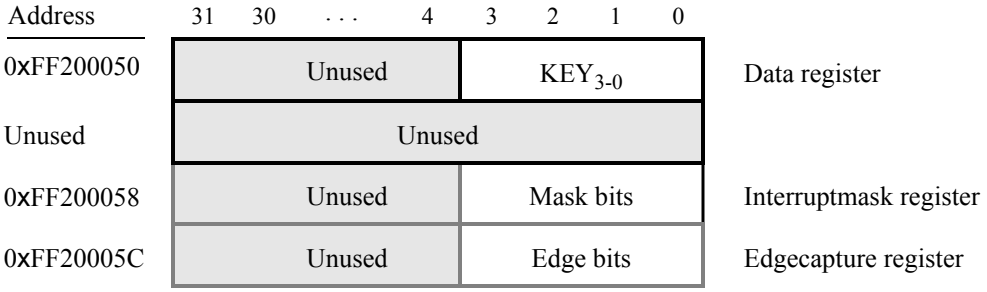

Figure 2: The pushbutton KEY port.

   To communicate with the KEYs and seven-segment displays, if applicable, use memory-mapped I/O as explained in the tutorial *Using Linux on DE-series Boards*. The source code from this tutorial for translating physical addresses into virtual addresses is included along with the design files for this laboratory exercise. You can use this source code as part of your solution.

2. Compile your code using a command such as `gcc -Wall -o part1 part1.c`.
3. Execute and test your program.

Part II

In Lab Exercise 1 you were asked to write a kernel module to control the LED lights and to display a digit, either on a seven-segment display or the Terminal window. The kernel module responded to interrupts generated by the KEY pushbutton port. Here you are to write another interrupt-driven kernel module.

Your kernel module should implement a real-time clock. Display the time on seven-segment displays, if available, and/or in the Terminal window. The time should be displayed in the format MM:SS:DD, where *MM* are minutes, *SS* are seconds, and *DD* are hundredths of a second. To keep track of time you should use a *hardware timer* module. The DE-series computers include a number of hardware timers. For this exercise use an interval timer implemented in the FPGA called *FPGA Timer0*. The register interface for this timer has the base address 0xFF202000. As shown in Figure 3 this timer has six 16-bit registers. To use the timer you need to write a suitable value into the *Counter start value* registers (one for the upper 16 bits, and one for the lower 16 bits of the 32-bit counter value). To start the counter, you need to set the *START* bit in the *Control* register to 1. Once started the timer will count down to 0 from the initial value in the *Counter start value* register. The counter will automatically reload this value and continue counting if the *CONT* bit in the *Control* register is 1. When the counter reaches 0, it will set the *TO* bit in the *Status* register to 1. This bit can be cleared under program control by writing a 0 into it. If the *ITO* bit in the control register is set to 1, then the timer will generate an interrupt for the ARM processor each time it sets the *TO* bit. The timer clock frequency is 100 MHz. The interrupt ID of the timer is 72. Follow the instructions in the tutorial *Using Linux on DE-series Boards* to register this interrupt ID with the Linux kernel and ensure that it invokes your kernel module whenever the interrupt occurs.

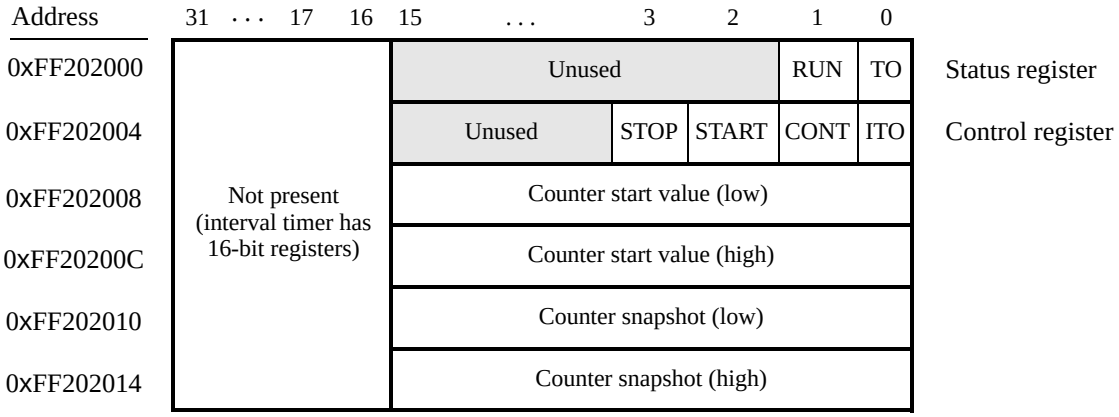

Figure 3: The *FPGA Timer0* register interface.

Perform the following:

1. Create a file called *timer.c* and type your C code into this file.
2. Create a suitable *Makefile* that can be used to compile your kernel module and create the file *timer.ko*. Insert this module into the kernel using the command `insmod timer.ko`. Each time an interrupt occurs your interrupt-service routine should increment the value of the time. When the time reaches 59:59:99, it should wrap around to 00:00:00.

   If using seven-segment displays, you can continuously display the updated time. But if using the Terminal window, it is better to print the time only when the user requests it. Your timer interrupt service routine should read from the *data* register in the KEY port and print the time whenever KEY_1 is pressed.

You can remove your module from the Linux kernel by using the command `rmmod timer`. When removed, your *exit* routine should clear the seven-segment diplays, if applicable.

Part III

For this part you are to write a kernel module that implements a *stopwatch*. The stopwatch time should be shown either on seven-segment displays or the Terminal window. The time should be settable using the SW switches and KEY pushbuttons in your DE-series Computer. The time should be displayed in the format MM:SS:DD as was done for Part II. Implement the stopwatch module using two sources of interrupts: the hardware timer *FPGA Timer0* and the KEY pushbutton port. For each timer interrupt you should *decrement* the stopwatch until it reaches 00:00:00.

The behavior of the interrupt service routine for the pushbutton KEY port depends on which DE-series board is being used. If you are using the DE1-SoC or DE10-Standard board, then follow the instructions in Table 2. For the DE10-Nano board, which has fewer KEYs and SW switches, implement the actions given in Table 3.

Table 2: Interrupt actions for the DE1-SoC and DE10-Standard boards.

KEY	Action
KEY_0	Toggle the stopwatch to be either running or paused
KEY_1	When pressed, use the values of the SW switches to set the DD part of the stopwatch time. The maximum value is 99
KEY_2	When pressed, use the values of the SW switches to set the SS part of the stopwatch time. The maximum value is 59
KEY_3	When pressed, use the values of the SW switches to set the MM part of the stopwatch time. The maximum value is 59

Table 3: Interrupt actions for the DE10-Nano board.

KEY	Action
KEY_0	Toggle the stopwatch to be either running or paused
KEY_1	If the stopwatch is running, just print the current time on the Terminal window. But if the stopwatch is stopped, then set the time using the SW switch values. Set one stopwatch digit each time KEY_1 is pressed, in a specific sequence. For the first press, set the right digit of DD, for the second press set the left digit of DD, for the third press set the right digit of SS, and so on. After each press of KEY_1 print the current stopwatch time.

The data registers in the SW switch port for the DE1-SoC and DE10-Standard boards are shown in Figure 4. The SW switch port for the DE10-Nano board, not shown in the figure, has only four switches SW_0 to SW_3.

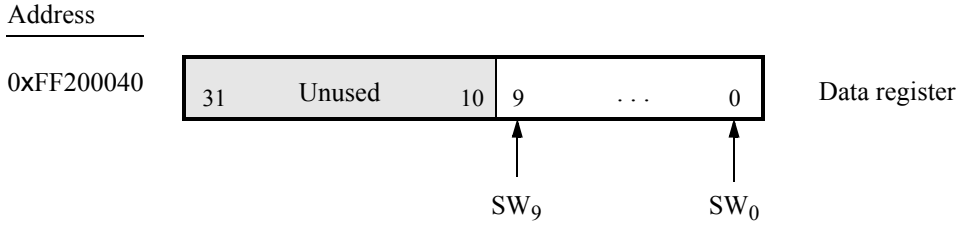

Figure 4: The SW switch port.

Perform the following:

1. Create a file called *stopwatch.c* and type your C code into this file.

2. Create a suitable *Makefile* that can be used to compile your kernel module and create the file *stopwatch.ko*. Ensure that the *timer* module from Part II has already been removed from the kernel, because it also responds to interrupts from FPGA Timer0. Then, insert the stopwatch module into the kernel by using the command `insmod stopwatch.ko`.

   If you are using seven-segment displays, then as soon as the module is inserted you should see the time 59:59:99 start to decrement on the displays. But if you are using the Terminal window, then you should see the stopwatch time whenever the user presses KEY_1.

You can remove your module from the Linux kernel by using the command `rmmod stopwatch`. When removed, your *exit* routine should clear the seven-segment displays, if applicable.