Laboratory Exercise 2

Developing Linux Programs that Communicate with the FPGA

Part I

In Lab Exercise 1 you were asked to write a user-level program to control the LEDR lights in the DE1-SoC Computer. Here you are to write another user-level program, which controls the seven-segment displays in the DE1-SoC Computer. 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

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`. To communicate with the seven-segment displays use memory-mapped I/O as explained in the tutorial *Using Linux on the DE1-SoC*. The data registers of the seven-segment display ports are shown in Figure 1. Remember that you have to translate the physical addresses of the ports into virtual addresses that can be used by your program running under Linux.

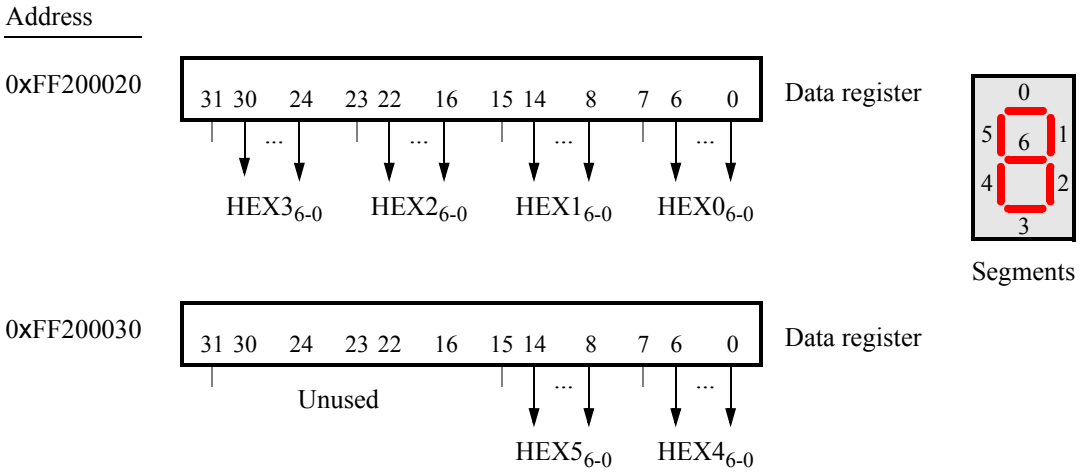

Figure 1: The seven-segment display ports.

Perform the following:

1. Create a file called *part1.c* and type your C code into this file. Compile your code using a command such as `gcc -Wall -o part1 part1.c`.
2. Execute and test your program.

Part II

In Lab Exercise 1 you were asked to write a kernel module to control the LEDR lights and a seven-segment display in the DE1-SoC Computer. 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 display a real-time clock on the seven-segment display. The time should be displayed in 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 DE1-SoC Computer includes 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 2 this timer has six 16-bit registers. To use the timer you need to write a suitable value into the *Counter start value* registers (there are two, 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 ARM interrupt 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 the DE1-SoC* to register this interrupt ID with the Linux kernel and ensure that it invokes your kernel module whenever the interrupt occurs.

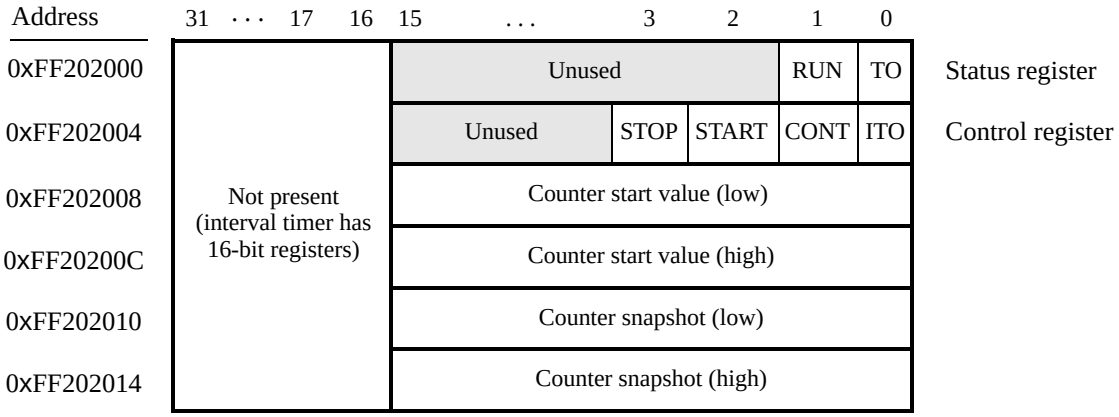

Figure 2: The *FPGA Timer0* register interface.

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`. As soon as the module is inserted, you should see the time 00:00:00 on the seven-segment displays. Each time an interrupt occurs your interrupt-service routine should increment the value of the displayed time. When the time reaches 59:59:99, it should wrap around to 00:00:00.

   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.

Part III

For this part you are to write a kernel module that implements a *stopwatch*. The stopwatch time should be shown on the seven-segment displays, and the time should be settable using the SW switches and KEY pushbuttons in the DE1-SoC 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 displayed time, stopping when it reaches 00:00:00.

For each interrupt from the KEY port you should do the following:

- 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.

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*. Make sure that the *timer.ko* module from Part II has already been removed from the kernel. Then, insert the stopwatch module into the kernel by using the command `insmod stopwatch.ko`. As soon as the module is inserted, you should see the time 59:59:99 start to decrement on the seven-segment displays.

   The data register in the SW port is shown in Figure 3. For a description of the registers in the KEY port, and how to use this port with interrupts, refer to the tutorial *Using Linux on the DE1-SoC*.

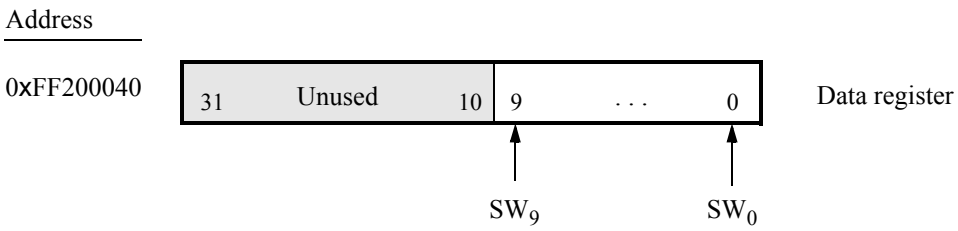

Figure 3: The SW switch port.

   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.

Copyright © Intel Corporation.