Laboratory Exercise 4

Using Character Device Drivers

This exercise is a continuation of Laboratory Exercise 3, and is about character device drivers.

Part I

Write a character device driver that implements a *stopwatch*. The stopwatch should use the format MM:SS:DD, where *MM* are minutes, *SS* are seconds, and *DD* are hundredths of a second. The code for your driver should initialize the stopwatch time to 59:59:99, and should *decrement* the time each $1/100$ seconds. Your character device driver should provide the current stopwatch time via the file */dev/stopwatch*. When the time reaches 00:00:00 the stopwatch should halt.

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 1 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 DE-series Boards* to register this interrupt ID with the Linux* kernel and ensure that it invokes your kernel module whenever the interrupt occurs.

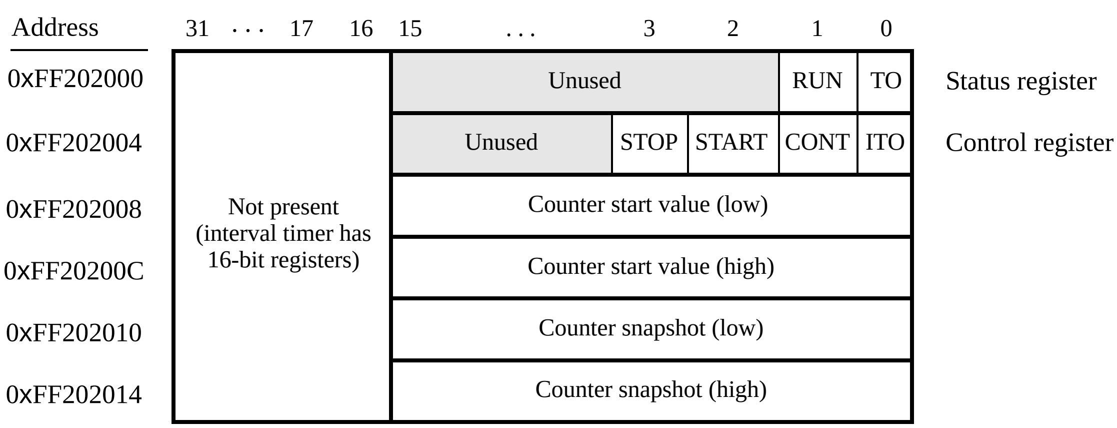

Figure 1: The FPGA Timer0 register interface.

Perform the following:

1. Create a file called *stopwatch.c* and type your C code into this file.
2. Create a Makefile, compile your kernel module, and insert it into the kernel.

3. Test your character device driver by using the command `cat /dev/stopwatch`, which should print the current stopwatch time.

Part II

Augment your module from Part I so that a user can control the stopwatch by writing commands to the file */dev/stopwatch*. Implement the following commands: `stop`, `run`, and `MM:SS:DD`. The `stop` command causes the stopwatch to pause. The *run* command causes the stopwatch to operate normally, decrementing every $1/100$ seconds. The `MM:SS:DD` command is used to set the time. For example, the command `echo 01:01:99 > /dev/stopwatch` sets the time to 1 minute, 1 second, and 99 hundredths.

If you are using the DE1-SoC or DE10-Standard boards, which provide seven-segment displays, then implement the two additional commands `disp`, and `nodisp`. Ignore these commands if you are using the DE10-Nano board. The `disp` command causes the stopwatch to show the time every $1/100$ seconds on the seven-segment displays HEX5-HEX0. The `nodisp` command turns off the seven-segment display feature, and clears HEX5-HEX0.

Perform the following:

1. Create a new version of your *stopwatch.c* source-code file and write the code required for the new functionality. In addition to `open`, `release`, and `read` functions needed for Part I, you will need to add a `write` function. It should check which command has been written to the driver by the user, and take appropriate action.
2. Use a Makefile to compile your kernel module. Ensure that the `stopwatch` module from Part I is removed from the kernel, and then insert the new *stopwatch.ko* file.
3. Test various commands to ensure that the character device driver works properly.

Part III

In this part we assume that the Linux system does not allow user-level code to access the memory addresses of I/O devices. Instead, user-level code has to make use of device drivers. Perform the following.

1. Write a user-level program that controls the `stopwatch` driver from Part II. Your program should execute in an endless loop, and should control the `stopwatch` using the pushbutton KEY and switch SW ports, as follows. Pressing KEY_0 should toggle the `stopwatch` between the *run* and *pause* states. Other KEYs are used to set the `stopwatch` time, based on the values of the switches SW. If you are using the DE1-SoC or DE10-Standard board, set the `stopwatch` time as indicated in Table 1. For the DE10-Nano board, which has fewer KEYs and SW switches, implement the actions given in Table 2. To communicate with the KEY and SW ports, read from their corresponding character device drivers. You may also want to display SW values on the LED switches, using the character device driver for the LED port. For the character device drivers, you could use either the drivers created as part of the solutions to Laboratory Exercise 3, or the drivers described in the tutorial *Using Linux on DE-series Boards*.
2. Compile your program using a command such as `gcc -Wall -o part3 part3.c`.
3. Ensure that the required character device drivers are inserted into the Linux kernel. Test your program by controlling the `stopwatch` using the SW switches and pushbutton KEYs.

Table 1: Setting the `stopwatch` for the DE1-SoC and DE10-Standard boards.

KEY	Action
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 2: Setting the `stopwatch` for the DE10-Nano board.

KEY	Action
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.

Part IV

For this part you are to write a user-level program that implements a *game*. Your program should use character devices drivers to communicate with the `SW` switches, `KEY` pushbuttons, `LED` lights, and `stopwatch`. The game involves a series of mathematical problems, such as summations, presented to a user, with a certain amount of time given to receive a correct answer. The game should perform as follows. In the first phase a default `stopwatch` time is shown on the seven-segment displays, and the user can change the displayed time by using the SW switches and KEYs. Use the same scheme as for Part III to set the stopwatch. Pressing KEY_0 starts the game. At this point the program should present a series of math questions that the user needs to answer within the `stopwatch` time. Answers should be entered through the command line. Incorrect answers to a question should be rejected, but the user should be allowed to try again as long as the time has not expired. After receiving a correct answer, the `stopwatch` should be reset and a new question asked. To make the game more interesting, you could increase the difficult of questions over time. At the end, when the user fails to respond within the `stopwatch` time, some statistics about the results should be shown to the user (for example, you could report the number of questions correctly answered, and the average time taken per question).

Perform the following.

1. Write the code that asks a series of math questions. An example of output that might be produced by your game, with user responses, is shown below.

```
Set stopwatch if desired. Press KEY0 to start
1 + 7 = 8
0 + 7 = 7
5 + 7 = 12
1 + 3 = 4
6 + 1 = 7
41 + 4 = 45
5 + 7 = 12
95 + 4 = 99
98 + 8 = 106
60 + 33 = 93
26 + 17 = 43
44 + 76 = 120
91 + 10 = 101
```

```
545 + 18 = 553
Try again: 563
972 + 3 = 975
572 + 75 = 627
Try again: 657
Time expired! You answered 17 questions, in an average of 2.73 seconds.
```

2. Compile your program using a command such as `gcc -Wall -o part4 part4.c`.

3. Run your program and make sure that the game functions properly.