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 the DE1-SoC* to register this interrupt ID with the Linux kernel and ensure that it invokes your kernel module whenever the interrupt occurs.

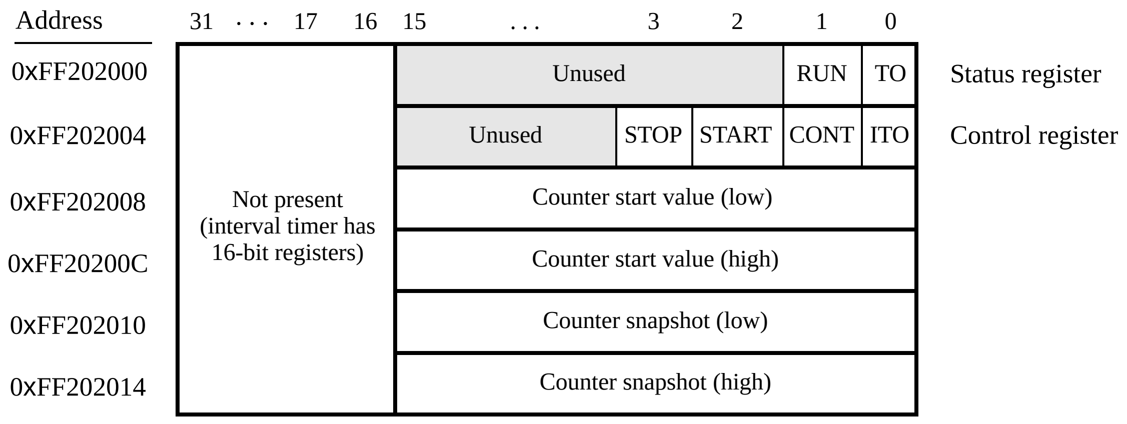

Figure 1: The FPGA Timer0 register interface.

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`, `MM:SS:DD`, `disp`, and `nodisp`. The `stop` command causes the time 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. 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. A good way to identify the command passed to the driver in the `write` function is to make use of a C library function such as *strcmp*.
2. Use a Makefile to compile your kernel module. Make sure 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, as follows. Pressing KEY$_0$ should toggle the `stopwatch` between the *run* and *pause* states. Pressing KEY$_1$ to KEY$_3$ should set the time according to the values of the SW slider switches. Set the hundredths (*DD*) if KEY$_1$ is pressed, the seconds (*SS*) for KEY$_2$, and the minutes (*MM*) for KEY$_3$.
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.

Part IV

For this part you are to write a user-level program that implements a *game*. Your program should use the character devices drivers that you wrote for the `SW` switches, `KEY` pushbuttons, `LEDR` 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. Using the same scheme as for Part III, pressing KEY$_1$ changes the hundredths part of the time, pressing KEY$_2$ sets the seconds, and KEY$_3$ changes the minutes. Pressing KEY$_0$ starts the game. At this point the program should print a message and wait for the user to press the return key. Following this action, the program should present a series of math questions that the user needs to answer within the `stopwatch` time. 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.

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, using KEY1, KEY2, and KEY3. Press KEY0 when done.
   Press Enter 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
   42 + 0 = 42
   79 + 1 = 80
   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.