Lab 3: LKMs and Shift Registers Using Optrex DmcSeries LCD Display

Irina Tolkova # 1240609
Samuel Scherer # 1435365
Tremaine Ng # 1433161

Overview

In this lab we expanded upon lab 2 where we implemented an Optrex DmcSeries LCD display by adding a shift register and interacting with GPIOs using a loadable kernel module. We also made a calculator program to utilize the functionality we made in our kernel driver. The objective of the shift register was to reduce the number of GPIO pins we used to interact with the LCD display. The kernel module had little functional advantage but gave practice and background with how kernel modules work.

Hardware

Board to Register

A Texas Instruments SN74HC595N register was used. 3 GPIO pins were connected to the shift register from the Beaglebone black board:

- Register input from GPIO pin 66
- SRCLK input from GPIO pin 67
- RCLK input from GPIO pin 68

The clear pin was tied high so that it never clears and the output enable pin was tied low so that it always enables output. These chip pins were labelled not clear and not enable, making these the appropriate logic levels to apply. Vcc and gnd were +3.3v and 0v as usual from the Beaglebone board supply pins.

Shift Register to LCD display

The shift register's outputs [H:B] correspond to the LCD screen's pins [DB7:DB4 , EN, R/W, RS]. This was chosen because the pins are conveniently located on the board in that order. This made the data values coming out of the register reversed compared to our configuration In lab 2. To reverse the values and send the correct outputs, we created a function *flipASCII* that reversed four-bit binary inputs. By applying this function to our input values, we could directly use code for taking in ASCII characters from lab 2.

Kernel Module

The kernel module allowed usage of the </linux/gpio.h/> library that gives direct access to the GPIOs. We used a few of its functions in our testing and implementation:

```
gpio_is_valid(int number)
gpio_request(unsigned gpio, const char *label)
gpio_free(unsigned gpio)
gpio_direction_output(unsigned gpio, int value)
gpio_set_value(unsigned gpio, int value)
```

Is_valid checks if the number is actually usable as a GPIO. Request attempts to get the GPIO for use and returns a negative if it is not able to (prints kernel error too). Free releases a gpio that was previously requested. Direction_output sets the GPIO to output mode and assigns the given value to it. Set_value sets the GPIO's value assuming output mode, and does nothing if not an output. The GPIO numbers are the same as the BeagleBone Black specifications describe, .e.g. GPIO 44 corresponds to request(44).

We created a simple user-space program "test_lcd.c", which opens, writes to, and closes the kernel module. By writing a very long string to the LCD display, we verified that the input to the screen wraps around to the second line after filling the first line, and fully clears and resets the screen after filling both lines. We also verified that various ASCII symbols -- such as uppercase and lowercase letters, punctuation, spaces, and newline characters -- are interpreted and handled correctly.

Calculator Program

To show the capabilities of the LCD kernel module, we designed a calculator application. This program prompts the user for arithmetic expressions, such as "2+2*8-6/4+500", prints the expression to the top line of the LCD display, then calculates the correct answer (accounting for order of operations) and prints it to the second line to two decimal points of accuracy. The user can then either continue the calculation (such as by printing "+6" or "/3"), or can start a new arithmetic expression.

Overall, this program was implemented by storing two arrays for each arithmetic problem: an array for the entered numbers and an array for the entered arithmetic operations. The user input is processed character-by-character, and placed in the next spot in the number or operations array depending on its value. At this stage, input numbers with multiple digits must be recognized as one number and handled as a single array entry. After acquiring the full pair of arrays for a problem, there is an intermediate processing step in which all multiplications and divisions are calculated, and a final processing step which calculates all additions and subtractions. This result is printed and stored, in case the user chooses to continue this calculation. The program correctly handles special cases such as fractions resulting from division, overflow of an equation past the first line of the display, division by 0 (positive or negative infinity), and multiplication of infinity and 0 (NAN).

Discussion

Various improvements and expansions could be made to this system. A possible improvement is rewiring the module and removing the need for the flipASCII method, since it produces unnecessary computations that can be avoided with simple hardware changes. The effect is minimal enough when writing small numbers of characters, but will impede performance at faster write rates. The calculator program's intention was to print the math equations as the console was being written to, then print the result when enter is pressed. This can be implemented in an expansion by trying to take terminal input directly or interfacing a keyboard directly to the Beaglebone instead. The calculator also prints the answer, then clears the result after a short time. Future operations can depend on this value, so reprinting it to the bottom right of the LCD display would allow the user to see the value but keep it from interfering with future operations.

Conclusion

Kernel modules give normal applications direct access to kernel level functions. In this lab, we used this capability to access GPIOs directly without writing to premade text files. Our programs could write to a file in the /dev/ folder instead and use our custom LKM drivers. Adding the shift register, though unrelated to kernel implementation, allows us to use 3 pins instead of 7 to 11 (4 bit operation/8 bit operation). GPIOs are a valuable resource on any system, so minimizing pin usage will be beneficial for future projects.