LKMs and Shift Registers

EE 474 Introduction to Embedded Systems
Lab 3

Daniel Park
Samuel Johnson
Reilly Mulligan

Introduction

The purpose of this lab was to use the Beaglebone to interact with the LCD display through a loadable kernel module. This required writing kernel space code that would enable the interaction between the Beaglebone, the shift register, and the LCD display. Consequently, the user is able to interface with the lcd through a dev file.

Our code goes above and beyond the specifications by including a program called “minesweeper.exe” that plays the classic game of minesweeper on the LCD display. We also modified “getmoney.exe”, a program that plays an animation, and “input.exe”, a program that allows writing to the lcd, from our previous labs, to work using the new kernel module.

Hardware Connections

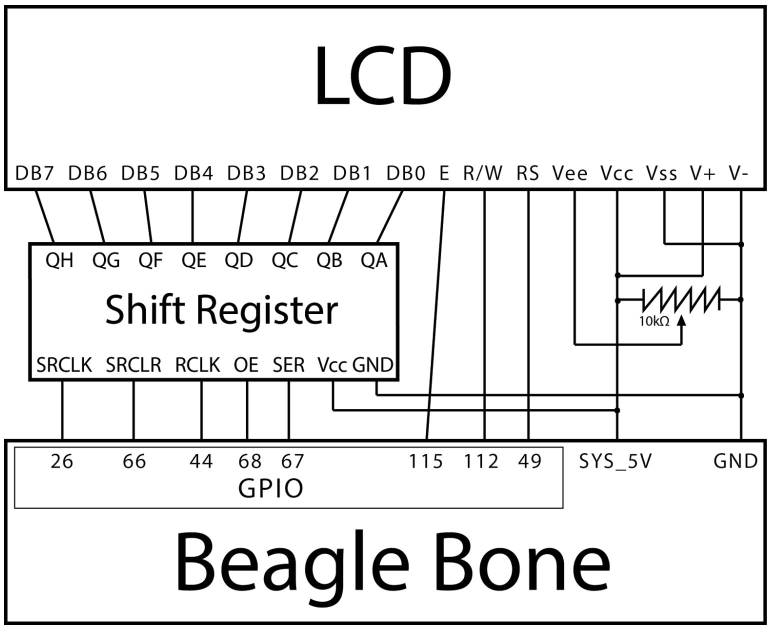
LCD
DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 E R/W RS Vee Vcc Vss V+ V-
QH QG QF QE QD QC QB QA
Shift Register
SRCLK SRCLR RCLK OE SER Vcc GND
10kΩ
26 66 44 68 67 115 112 49 SYS_5V GND
GPIO
Beagle Bone

Figure 1: Circuit Wiring Overview

As shown in Figure 1 above, the GPIO pins drive the shift register, which then drives the data pins into the LCD. The other pins for the LCD are wired directly to GPIO pins on the Beaglebone. Power ports from the Beaglebone drive the power for the LCD and the shift register as well as the LCD backlight, and a potentiometer is used to control the contrast of the LCD display.

Software Overview

Our software consists of several different main programs and a loadable kernel module that interact with the LCD. Figure 2 shows the interactions between the functions that have been implemented. The specific functionality of each function will be described in sections below.

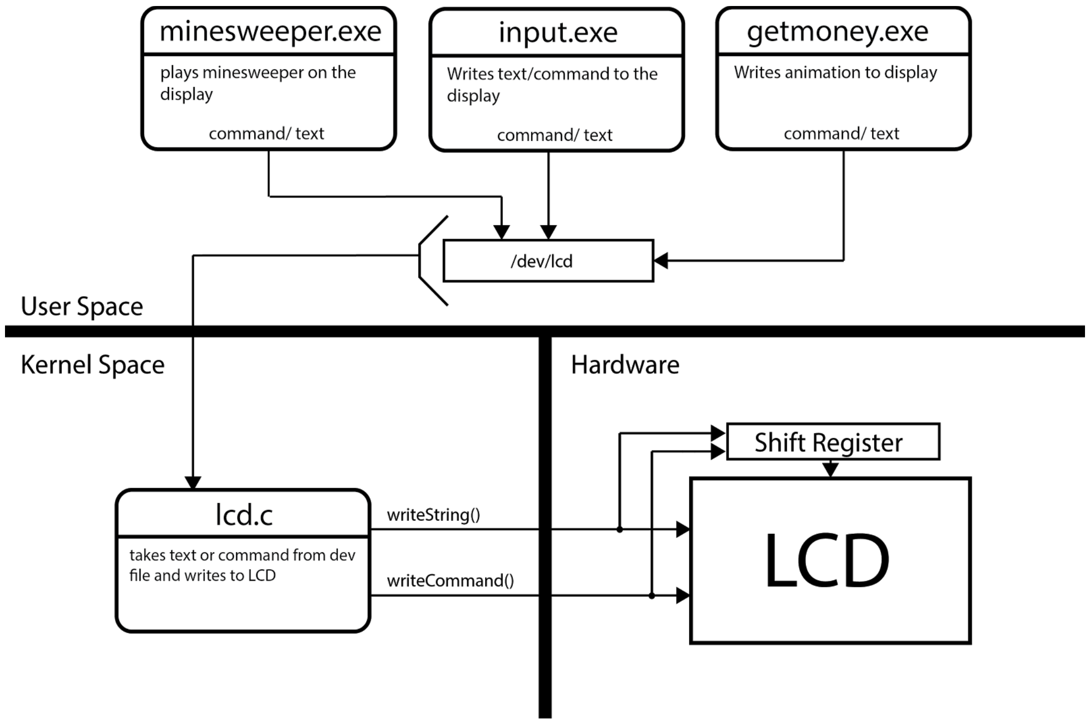

Figure 2: Diagram showing software and Hardware interactions

lcd.ko

The lcd.c file is the source code for the loadable kernel module to initialize and to interface with the LCD display. Table 1 lists out the helper functions and Table 2 lists out the important functions that enable us to interface the LCD in the kernel below:

Table 1: Preliminary assisting Functions in kernel code

Function Call	**Use**
flipE(void)	flips the enable pin to the LCD
writeCommand(int config)	writes given command to the LCD
writeChar(char config)	writes given character to the LCD
writeString(char * str)	writes given string to the LCD

Table 2: Interfacing Functions (called in kernel space)

Function Call	**Use**
lcd_init(void)	initializes the module in the kernel and requests GPIOs needed
lcd_exit(void)	unregisters the device from the kernel, and frees any used GPIOs
lcd_open(struct inode*…, etc.)	opens the LCD device
lcd_write(struct file*..., etc.)	write from dev file to the LCD
lcd_read(struct file*..., etc.)	unsupported
lcd_close(struct inode*…, etc.)	closes the LCD device

This kernel module acts as a device driver for the LCD so that commands and text can be written to it. This is accomplished from user space by writing to the dev file /dev/lcd. Text can be simply written to the dev file, while commands are preceded by the escape character '/'. A list of commands are shown in Table 3 below:

Table 3: User commands for interfacing the lcd through the dev file

Input	***Command***
/clear	Clears the LCD screen
/b	Backspace
/tl	Moves cursor to top line
/bl	Moves cursor to bottom line
/scl	Shifts cursor left
/scr	Shifts cursor right
/sdl	Shifts display left
/sdr	Shifts display right

getmoney.exe

getmoney.exe is an executable that produces moving text on the LCD. A '$' character scrolls along the top line, resetting when it reaches the end of the display. The string "GET MONEY" is always displayed in the bottom left corner of the screen. This was accomplished by writing the characters in the appropriate part of the screen, delaying, clearing the buffer, and writing characters to the string in a new position. This makes the text appear to be moving. Placement of the '$' was determined by the various loops in the code. getmoney.exe uses the dev file to write to the LCD.

input.exe

input.exe is a simple program on the terminal that allows the user to write commands and text to the LCD. The user is continuously prompted for a text or a command, which is subsequently written to the LCD, or executed on the LCD. Since this program writes directly to the dev file, the commands will be the same as those found in Table 3.

minesweeper.exe (extra)

minesweeper.exe is an executable that runs a 2x16 minesweeper game on the LCD. At first, the game requests that the user to choose the number of mines in the field, ranging from 1 to 31. Any invalid entered value will throw “invalid input” and will ask the user to provide another input. Once a valid input has been entered, the game will start, and the user can input the commands shown in Table 4.

Table 4: Minesweeper Inputs and Commands

Input	**Command**
w	move cursor up
a	move cursor to the left
s	move cursor down
d	move cursor to the right
e	reveals the selected tile
q	quits the game

The rules for the game are as follows:

- *n* mines are distributed randomly among the 32 tiles of the game board, where *n* is a number between 1 and 31 specified by the user
- Each tile can either have a mine or be a safe space
- At the beginning of the game, the content of each tile is hidden from the player
    - As tiles are revealed, safe tiles (with at least one neighboring tile that contains a mine) will display the total number of mines surrounding it
    - Neighbors of a tile are defined as the 8 tiles surrounding that tile (each corner tile has 3 surrounding tiles)
- The player moves the cursor around the board and can select hidden tiles
    - If a player selects a tile with a mine, they lose the game
    - If a player selects a tile without a mine, that tile is revealed. In addition, all connected tiles without any mine-containing neighbors are revealed
- The player’s goal is to reveal all of the tiles that do not contain mines

Additionally, at an event of a loss, when the user reveals a hidden mine, the LCD will bring up a defeat display sequence. On the other hand, when the user uncovers the playing field without triggering a mine, the LCD will bring up the victory display sequence.

Conclusion

The purpose of this lab was to become familiar with kernel space code and implement a loadable kernel module to drive the LCD and shift register. We were able to successfully implement this functionality, with the ability to write text and commands to the LCD. The biggest challenge in this lab was dealing with little problems in kernel space. A specific example of this is the copy_from_user function that returns the number of bytes that could not be copied. This was a problem, because we assumed that it would return the number of bytes written. With this assumption our kernel module would hang indefinitely.

Eventually we were able to overcome these challenges and implement some pretty cool functionality on the display. We have found that writing to the dev file is a much more efficient way of interacting with the LCD than the named pipes in the previous lab. Even with the trickiness of writing code in kernel space, it is clearly advantageous to write a driver in this way. The minesweeper game we implemented on this display involves some guess work due to the dimensions, but works perfectly. The code is easily scalable to a larger format if desired.