Embedded Systems Assignment 2

Steven Diviney 08462267

December 3, 2012

1 Introduction

The aim of this assignment was to implement a keypad using the touchscreen on the LPC2468 development board. The values entered are printed to the console when the users presses OK and cleared when the user presses CANCEL. All requirements for the assignment were met.

The hardware manufacturers provide software to drive the LCD display. The API presented to the programmer is very simple to use. It takes care of all of the low level details, which are substantial. A few initialization methods need only be called.

FreeROTS is used for scheduling. A task, called LCD, is created. It is responsible for drawing the keypad, handling touch interrupts and determining which key has been pressed.

The touchscreen generates interrupts like any other button on the bus. It must first be enabled and configured.

2 Implementation

The hardware must first be initialized. This is done in a similar fashion to the first assignment and is relatively straightforward. The UART is enabled to allow console output. The LCD is then initialized. The code to do this was provided by the manufacturer of the development board. After this interrupts are set up. The touch panel generates interrupts exactly like a button on the bus. The screen is setup and configured for falling-edge sensitivity. The VIC is then configured and the interrupt handler assigned. The handler must be specifically written for hardware being used. This was provided.

2.1 LCD Task

The LCD task first finishes initializing the LCD display. The code to do this was provided and must be called from inside a task due to how it implements a delay. The screen is then cleared and a grid of boxes is drawn. These boxes represent the keypad. The position of each box is hard-coded. Not only is this easier to implement but it also makes more sense from a computational point of view. The keypad is static. It does not move so there is no point in calculating element positions at runtime. The code to do this is a simple loop that draws each of the 12 boxes. The keypad text is then drawn. Again, the positions are hard-coded.

The button coordinates are kept in a static array. This array is also used to decided which button a user has pressed.

The code then enters an infinite loop that waits for an interrupt. A FreeROTS queue is used to wait. When an item is placed on the queue the handling code is run (the code is simply allowed to run beyond the blocking xQueueReceive call). The code then loops back to the blocking queue wait.

2.2 Interrupt Handling

The body of the interrupt handler is very simple. An item is placed on the queue and the interrupt is cleared. The interrupt is kept as short as possible to try and eliminate any race conditions caused by multiple interrupts.

The code responsible for handling the touch screen event is run outside of the interrupt handler. Interrupts are immediately disabled. The x and y position of the first touch event are obtained. These two coordinates are used to identify which button was pressed. The check is done linearly. The x coordinate is checked to be inside each box. If this check passes the y coordinate is then checked. This could be further optimized with a simple binary search but would add little benefit as there are only 12 buttons. The array index of the relevant box is returned, or -1 if x and y did not fall inside a box.

The array index is then mapped to the relevant value. A simple character array is used to store the input. If "OK" is pressed the character array is printed to the console. If "CANCEL" is pressed the array is cleared. It was realized afterwards that no bounds checking had been done on this array. This would be a serious oversight in a production system, as any input of 256 or more characters, as unlikely as that would be, would probably cause the program to crash. An ideal solution would be to allocate more memory as needed but this is slightly out of scope for this project.

The relevant button is then filled in. This is give the user a sense of pushing a button. The code then enters an empty loop waiting for screen pressure to return to 0. The button is then drawn in its original form. All of the digits are redrawn as there is no positional information associated with them individually.

3 Testing

A number of tests were carried out.

3.1 Button Mapping

To ensure the buttons were mapped to their correct values a test was conducted. With each interrupt the following were printed and checked for consistency; the x and y coordinate, the array index of the pushed button and the value of the pushed button. Each button was checked in succession and then randomly. Each x and y coordinate was not checked, rather a representative sample was used. The buttons were mapped correctly.

An unexpected bug was found here. The system would occasionally print out 13 as the index of the button. A further test was conducted.

3.2 Button Detection

As noted above the system sometime returned 13 as the index of the button pressed. The infrequency of the event and the inability to reproduce it consistently lead to the conclusion that the error was a boundary case. A text was conducted by pressing around the edges of the screen. The value 13 was printed every time.

The code to map the x and y coordinates to a button region originally fell through to a return statement. This presumed that one of the buttons had always been pressed and did not take the padding around the edges into account. More importantly, the lines used to draw the buttons were not considered parts of the buttons. The code was changed to return -1 if the press was not inside

a button. As a single line is used to define the edge of two buttons in many instances it was left unchanged.