Abstract

This lab is an intermediate set of experiments with the Freescale MCF5234 microcontroller, the Hitachi HD44780 LCD character display, and the MM74C922 keypad. Specifically, the exercises are intended to introduce message passing with queues and I/O using Edge Port pins and IRQ interrupt handling. Using the uC/OS environment and some provided library code, the microcontroller is configured print specific characters depending upon which keypad keys are pressed. Then the LCD display is controlled using the keypad buttons to move a character around on the screen

Design

Prelab

The previous lab's hardware was augmented by the keypad and its controller. Notably, a different voltage level is used to power the keypad controller. The different voltage levels on the board are indicated by different color wires. Similarly, additional hardware for this lab was wired with a different color than that of the first.

Exercise 1

No code was written for the next part. Two different time intervals were measured using single shot capture mode on an oscilloscope, wherein the oscilloscope records signal data from one or more probes during a user-selected interval.

For the first measurement, two probe wires were inserted into the breadboard - one for a Y line output of the keypad and another for the Data Available output line of the keypad controller. The oscilloscope captured both signals as a button was pressed, and the time delay between the Y line response and the Data Available response was measured. An inbuilt resistor and a timing capacitor installed by us are used to calculate the expected value of that interval. (See Row 1 of the table below).

For the second measurement, only a single probe was necessary. The Data Available line was probed and recorded while two buttons were held simultaneously, then one was released. The interval between Data Available signals for the two keys represents the minimum time between any successive Data Available signals. The same resistor and capacitor are used to calculate the expected value. (See Row 2 of the table below.)

	Measured Time	Datasheet Time
Time from any Y line going low to Data Available going high. T1 = RC *	13 ms See Figure 1 below	T1 = RC = 10E3 * 1E-6 = 0.01 s = 10 ms
Press and hold 2 buttons. Now release the first button and measure how long the Data Available line is low. Take several measurements and record the min and max. T3 = 0.7RC *	14 ms See Figure 2 below	T3 = 0.7RC = 0.7 T1 = 0.7 * 10ms = 7 ms

(internal) R = 10K = 10E3
(KBM pin) C = 1 uF = 10E-6

*From MM74C922 datasheet.

Figure 1: Oscilloscope capture for data available and Y line

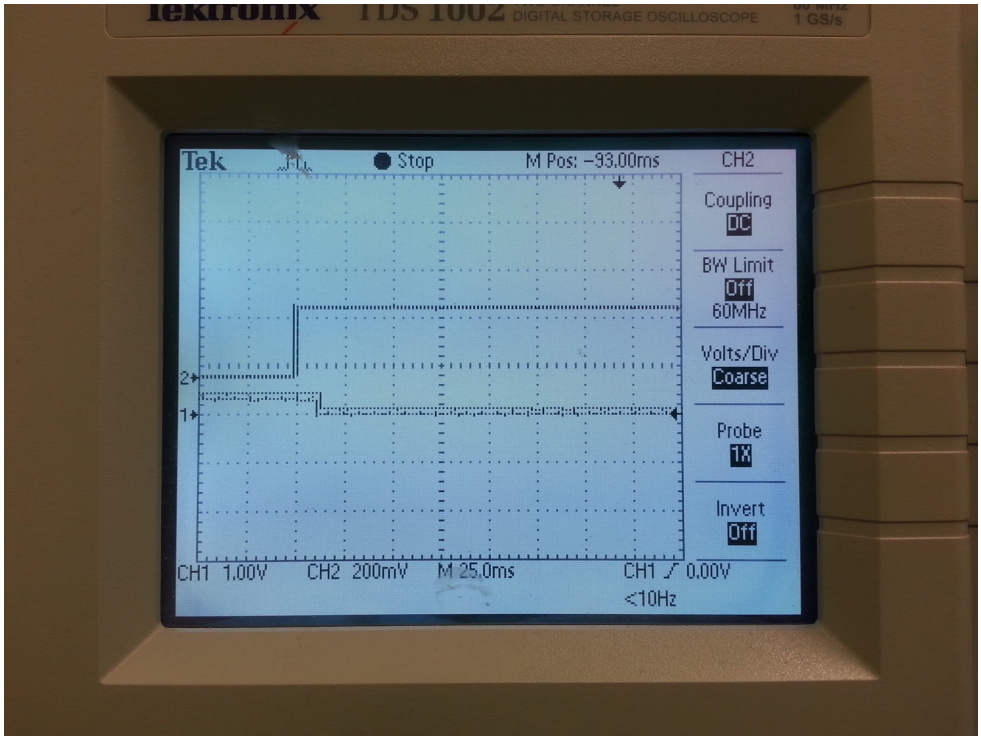
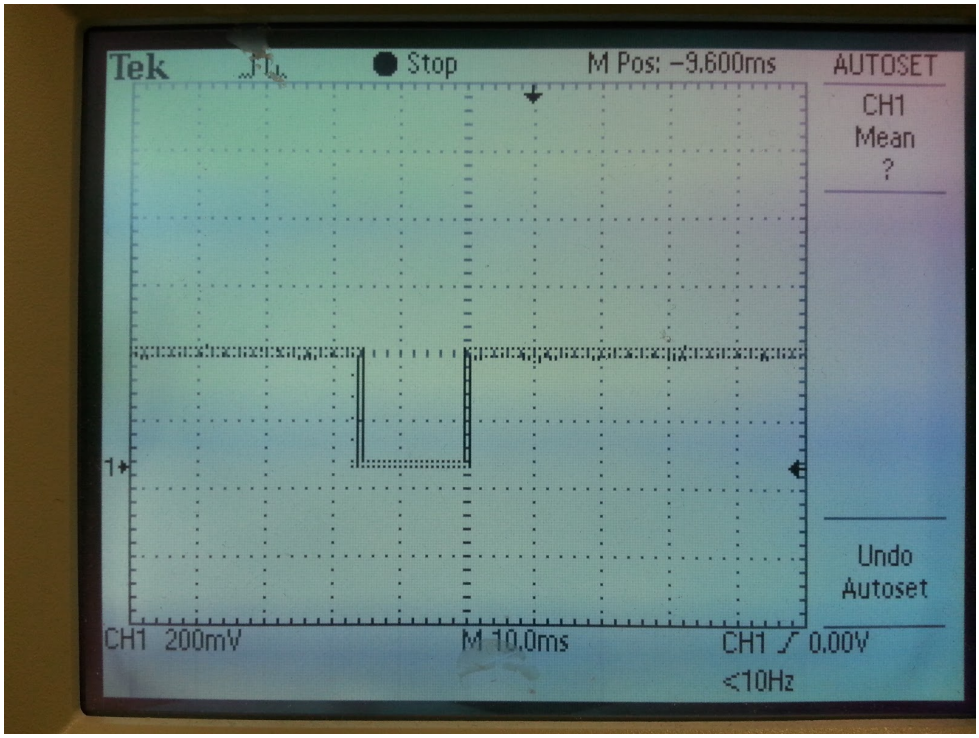


Figure 2: Oscilloscope capture for minimum data available high-low-high.



Exercise 2

The hardware was connected and the provided software was loaded. Heartbeat and context switch signals were measured. No experimental or theoretical results exist.

Exercise 3

The IRQIntInit uCOS system call was used to declare an interrupt for the edge port module in use by the keypad (IRQ3). Specifically, the IRQIntInit was called from early in the UserMain task, and it enabled IRQ3 to be triggered by a rising edge. This was done with the following register configuration:

```
sim.eport.eppar = 0x40; /* 00 00 00 00 01 00 00 00 - EPPA3 to 01 */
sim.eport.epddr = 0x0; /* All edge port pins as inputs */
sim.eport.epier = 0x8; /* Enable IRQ3 only 0 0 0 0 1 0 0 0 */
```

Then a call to SetIntC registered our ISR with the approprate priority and interrupt vector.

The ISR itself was defined by the INTERRUPT uCOS system macro. Only two things were done within the ISR. Clear the interrupt edge and post the keypad output to a sytem Queue:

```
sim.eport.epfr=0x08; /* Clear the interrupt edge 0 0 0 0 1 0 0 0 */
OSQPost(&queue, keypad.get());
```

The ISR was kept so simple so that the system was blocked for as little time as possible. A queue was used to prevent loss of keypad input, even if input arrives before previous input has been fully processed. The queue contents are processed within UserMain, where an infinite loop pends on the queue. Exercise 3 was for basic testing only, so the keypad outputs were printed to the serial mttty console.

```
while (1) iprintf("%c\n", OSQPend(&queue));
```

Exercise 4

The keypad map in keypad.cpp was modified to have a Left-Down-Up-Right "arrow key" style layout. Kepresses were processed on a switch-case chain for each of the four nonempty keys, rather than just printing them to the mttty console. Each case moved the cursor in the desired direction:

```
const char KeypadButtonMapText[BUTTONS] [MAX_BUTTON_NAME] =
	{
	"", "", "", "",
	"L", "D", "", "U",
	"", "", "", "",
	"R", "", "", ""
	};
```

Within the UserMain loop, cursor state was tracked by two variables. One for cursor position in the current screen (upper vs lower) and one for the current screen itself. Since the screen is partitioned into two arrays, an upper half and a lower half, left and right movement was as simple as decrementing and incrementing the cursor position. The edge cases were at positions zero and eighty, where the cursor position needed to wrap to forty and zero, respectively, and the current screen needed to switch. For example, to handle "Left":

```
if (currentPosition == 79) {
	toggleScreen();
	currentPosition = 0;
} else {
	currentPosition++;
}
```

"Right" movement was mirrored. Up and Down movement was handled by decrementing and incrementing the position by 40 (half the screen, since each screen is split into two rows). The edge cases were at entire rows rather than just positions. For example, "Up" movement needed to toggle the screen if the cursor was already on the top row (< 40):

```
if (currentPosition >= 40) {
	/* currently on bottom line of current screen */
	currentPosition -= 40 ;
} else {
	/* currently on top line of current screen */
	toggleScreen();
	currentPosition += 40;
}
```

Testing

Test ID	Description	Expected Output	Actual Output
1	ex 3 - Try every key press	mttty output for each key	As expected.
2	ex 3 - Try rapid key presses	keypresses queued and output to mttty in the same order as pressed. (delay acceptable)	As expected.
3	ex 3 - Try concurrent key presses	>2 keys held results in only first and last keys in “hold sequence” registered as output to mttty	As expected.
4	ex 4 - Try every row from leftmost column using left key	transition to rightmost column in line above	As expected.
5	ex 4 - Try every row from rightmost column using right key	transition to leftmost column in line below	As expected.
6	ex 4 - Try every column from bottom row of top screen (and bottom row of bottom screen) using down key	transition to top row of bottom screen (and top row of top screen) in same column	As expected.
7	ex 4 - Try every column from top row of top screen (and top row of bottom screen) using up key	transition to bottom row of bottom screen (and bottom row of top screen) in same column	As expected.

Questions

Q.

Suggest an alternative to using an encoder when interfacing a keypad. Advantages and disadvantages?

A.

The alternative to using a keypad is to read directly the results from the keypad into the microprocessor, in this case the netburner. The main advantage would be to avoid the extra cost related to buying/ using the encoder and the learning experience that comes with using it in a more lower level design. The disadvantages are, for most projects, much more meaningful. For example, wasting 8 pins of the platform to use just for the keypad is a heavy loss, since 1 pin for each row and 1 pin for each column would be necessary. Also, debouncing circuitry would be necessary if spurious repetitive inputs needed avoiding. Finally, logic to deal with multiple simultaneous keypresses would need to be hand-rolled in hardware or software (the controller provides a minimum interval between Data Available highs for that situation). Another alternative could be achieved with the use of multiplexers, but then additional components would be required and therefore losing the advantage of not using the encoder.

The time spend in configuring each single instance, figuring out the best procedures to use the keypad and time is one of the things that in an industrial environment is very valuable. Thus, using employees to do something where a cheap component can be used isn't the best practice in the market.

Conclusion

This lab introduced synchronized multitasking using interrupts using edge port triggers. Queues were introduced to manage nonblocking sequential I/O in the uC/OS environment. The edge port module needed proper configuration so that the rising edge of the keypad data line could be used for interrupts at all. An ISR was declared and defined to respond and read from a hardware resource - the keypad. The ISR unblocked a thread that processed the keypad output from a queue and manipulated the cursor position on the LCD character display. The ISR needed to be as short and deterministic as possible (i.e.

nonblocking) so that normal execution could be restored. Longer, possible blocking procedures were done within the UserMain function.

In order to process the keypresses and move the screen cursor, edge cases at screen boundaries were handled carefully. Edge case handling was verified by transitioning to and from leftmost and rightmost columns in every row, and by transitioning to and from top and bottom rows in every column (for both screens).

General practices emphasized by the lab were:

- Improvement of readability and maintainability by using named constants instead of “magic numbers”
- ISR and “normal” task coordination and communication using queues
- Runtime robustness improvement by clear, understandable edge case handling, as well as extensive testing