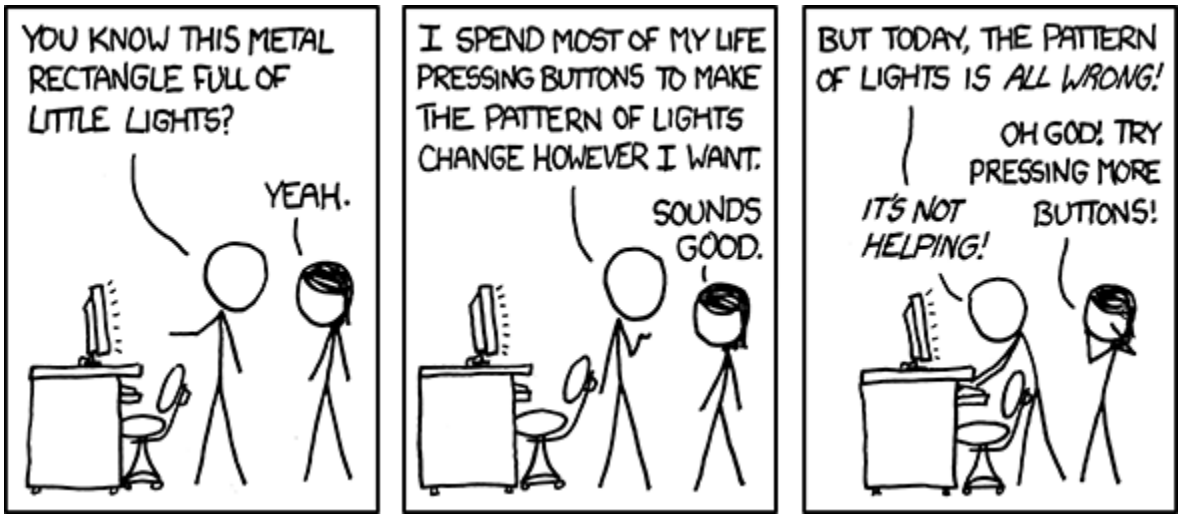
ECE 315 - D1
Lab 1

James Finlay
Andrew Fontaine

Due Oct 9th, 2013



Computer Problems, xkcd - http://xkcd.com/722/

October 9, 2013

James Finlay 1263258
Andrew Fontaine 1254308

Abstract

The purpose of this lab is to gain experience interfacing with the MCF5234, with task synchronization, and in the design of uC/OS multitasking environment software. This lab is composed of four exercises that practice these objectives through interactions with a character LCD.

Design

Exercise 1

Goal: Build a project, and verify that the LCD hardware is functioning correctly.

To verify the LCD hardware functioned correctly and that our connections were proper, we attached the 50-pin header to our 50-pin attachment. This lead back to the microcontroller. The LCD screen was attached to our breadboard via a 16-pin cable. The verification was perfomed when we attached a 5 V power source to our breadboard and the LCD screen displayed two rows of black rectangles, then the string “Welcome to ECE 315 - Fall 2013”.

Exercise 2

Goal: To understand and modify the initialization sequence for the LCD.

For exercise 2, we modified the *lcd.cpp* file to the attached code. The *send_cmd()* functions originally contained mystery numbers without any information. However, by converting the numbers to binary and comparing with the available commands in *lcd.h*, we were able to refactor the code to make it more legible. The following table is a representation of this process:

56 (111000)	0x20|0x10|0x08|0x00	CMD_FUNCTION|FUNCTION_8BIT|FUNCTION_2LINE|FUNCTION_5x8

After this, we implemented the *send_cmd()* function to blink on the upper and lower screens by using the command definitions in *lcd.h*. To turn the blinker on, we used the following parameter:

CMD_DISPLAY|DISPLAY_ON|DISPLAY_NOCURSOR|DISPLAY_BLINK

Exercise 3

Goal: To display a single character on each of the four lines of the LCD. The location of the character will depend on reading a value from the DIP switches.

In order to achieve this goal, we implemented four tasks that wrote a character to different lines on the screen. These tasks utilized semaphores to manage the process flow. Whenever a task would finish, it would post the next semaphore for the appropriate pending task.

James Finlay 1263258
Andrew Fontaine 1254308

Once the process flow was correct, we implemented query statements in the tasks to check the DIP switch positions. Depending on whether the switches were full left or no, the characters were shifted to the right side of the screen.

Exercise 4

Goal: To switch the order of the display of characters based on the value of the DIP switches.

To change the order of the tasks, we simply used DIP switch position queries to post the different semaphores. If the DIPs were all left, the posted semaphores would be in increasing order. Else, the semaphores would decrease, changing the posted order.

Test Cases

The test cases used to test the solutions. State the case tested (ie. data input, output, condition tested or observed), the reason why this case was tested, the expected results of the test, and the actual results of the test. The success or failure of the test is not important. The selection and discussion of the test cases are the important aspects of the testing section. Do not provide debugging detail.

James Finlay 1263258
Andrew Fontaine 1254308

Case Tested	Description	Expected Result	Actual Result
Exercise 1 Verification	Simple hardware boot configuration	The string “Welcome to ECE 315 - Fall 2013” was displayed	The string “Welcome to ECE 315 - Fall 2013” was displayed
Exercise 2 Refactoring Verification	Testing the send_cmd() parameter changes. Since the code was refactored, rather than having features added, our test was to ensure that the application worked the same as before the refactor	The display of the screen was unchanged after the refactoring.	The display of the screen was unchanged after the refactoring.
Exercise 2 Blinking Lights Verification	To test our blinking light code, we ensured that the lights were blinking as intended.	The blink feature of the LCD screen was active.	The blink feature of the LCD screen was active.
Exercise 3 Single Character Verification	To test our display code, we ensured one character was displayed per line.	Each line displayed one character.	Each line displayed one character.
Exercise 3 Semaphore Verification	To test our semaphore code, we ensured one line was displayed at a time	One line displayed its character at a time.	One line displayed its character at a time.
Exercise 3 DIP Switch Verification	To test our switch code, we ensured that, when any of the switches were flipped from left to right, the display side switched from left to right.	When a switch was flipped, the characters were displayed on the other side of the display.	When a switch was flipped, the characters were displayed on the other side of the display.
Exercise 4 Semaphore Flip Verification	To test our semaphore and switch code, we ensured that, when any of the switches were flipped from left to right, the order in which the lines were displayed reversed.	When a switch was flipped, the order in which the lines were displayed would change direction.	When a switch was flipped, the order in which the lines were displayed would change direction.

James Finlay 1263258
Andrew Fontaine 1254308

Results

Prelab

The following values are important because they are the maximum voltage and current that can be used by the MCF5234. Without knowing these values, it would be dangerous to use the board due to the unknown limits of the board's capabilities.

Digital Input Voltage (Absolute Maximum)	-.3 to +4 V
Instantaneous Max Current Single Pin Limit	25 mA

During Lab

There were few measured results made during the exercises. Testing results can be found in the previous *Test Cases* section, where it is shown that all exercises were completed correctly.

Questions

1. *"The LCD class only contains code to write to the LCD. Why don't we read from the LCD as well? What added hardware or software would be required to implement reading from the LCD in our system?"*

The standard range of instruction functions are available for writing to the LCD through the *send_cmd()* functionality. For full read/write with the LCD, we could interact with the MPU through the pin functions outlined in the HD44780U Datasheet. RS is used to select the register type, and R/W to select read/write. E is then used to start data read/write and the various DB#'s for data transfer.

By digging through the existing *lcd.c* code, we found multiple occurrences of LCD_xx variables that appeared similar to the HD44780U Datasheet Pin functions. We believe that these variables could be manipulated to allow reading from the LCD.

To read from the LCD, the system would require a rewiring to allow inputs into the MCF5234 from the LCD's data transfer.

James Finlay 1263258
Andrew Fontaine 1254308

Conclusion

In this lab, we first wired a breadboard to connect a 4-line Hitachi LCD display to the MCF5234. This allowed us to write to and interact with the LCD. We then used semaphores and OS tasking to create separate tasks that handled the displaying of strings on each line of the LCD display. We read input from the DIP switches located on the MCF5234, and manipulated the order each line was displayed in and the side of the screen on which the characters were displayed.

This lab served as an effective introduction to OS tasks, semaphores, and interaction with external hardware. Even seemingly simple task of writing to an external display can prove to be quite difficult.

Source Code

In lab1.cpp:

```
/* LCD cursor positions for edge of screen and beginning of second screen. */
#define LCD_TOP_END           0x27
#define LCD_BOTTOM_BEGIN      0x28
#define LCD_BOTTOM_END        0x4F

/* DIP Switches defined as on/off */
#define DIP_SWITCH_ON   0xFF

extern "C" {
      void UserMain(void * pd);
      void StartTask1(void);
      void StartTask2(void);
      void StartTask3(void);
      void StartTask4(void);
      void Task1Main(void * pd);
      void Task2Main(void * pd);
      void Task3Main(void * pd);
      void Task4Main(void * pd);
}

const char * AppName= "James and Andrew in the MORRRNNIING";

/* Task stacks for all the user tasks */
/* If you add a new task you'll need to add a new stack for that task */
DWORD Task1Stk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );
DWORD Task2Stk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );
DWORD Task3Stk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );
DWORD Task4Stk[USER_TASK_STK_SIZE] __attribute__( ( aligned( 4 ) ) );

/* This really needs to be in the heap so it's here and allocated
 * as a global class due to its size */
Lcd myLCD;
OS_SEM theSemaphore1, theSemaphore2, theSemaphore3, theSemaphore4;
```

```
void UserMain(void * pd) {
…
	/* For exercise 3 and 4 put your semaphore and/or queue initialisations
	 * here.
	 */
	if (OSSemInit(&theSemaphore1,1) == OS_SEM_ERR)
		iprintf("Error in initializing semaphore1.\n");
	if (OSSemInit(&theSemaphore2,0) == OS_SEM_ERR)
		iprintf("Error in initializing semaphore2.\n");
	if (OSSemInit(&theSemaphore3,0) == OS_SEM_ERR)
		iprintf("Error in initializing semaphore3.\n");
	if (OSSemInit(&theSemaphore4,0) == OS_SEM_ERR)
		iprintf("Error in initializing semaphore4.\n");
	/* Start all of our user tasks here. If you add a new task
	 * add it in here.
	 */
	StartTask1();
	StartTask2();
	StartTask3();
	StartTask4();
}

/* Name: StartTask1
 * Description: Creates Task1 as an OS Task. Task1 runs Task1Main.
 * Inputs: none
 * Outputs: none
 */
void StartTask1(void) {
	BYTE err = OS_NO_ERR;

	err = OSTaskCreatewName(	Task1Main,
				(void *)NULL,
				(void *) &Task1Stk[USER_TASK_STK_SIZE],
				(void *) &Task1Stk[0],
				TASK1_PRIO, "Task 1" );
	if(err) {
		iprintf("Task Creation Error in StartTask1\n");
	}
}
```

```
/* Name: Task1Main
 * Description: Main function for Task1. Puts character 'a' on top row of LCD
 *          screen. Location is based on DIP switch values
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void  Task1Main( void * pd) {
      /* place semaphore usage code inside the loop */
      while (1) {
            if (OSSemPend(&theSemaphore1, 0) == OS_TIMEOUT)
                  iprintf("Timeout waiting for Semaphore 1\n");

            myLCD.Clear(LCD_BOTH_SCR);

            if (getdipsw() == DIP_SWITCH_ON) {
                  myLCD.Home(LCD_BOTH_SCR);
            } else {
                  myLCD.MoveCursor(LCD_UPPER_SCR, LCD_TOP_END);
            }

            myLCD.PrintChar(LCD_UPPER_SCR, 'a');
            OSTimeDly(TICKS_PER_SECOND*.5);


            if (getdipsw() == DIP_SWITCH_ON) {
                  if (OSSemPost(&theSemaphore2) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 2\n");
            } else {
                  if (OSSemPost(&theSemaphore4) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 4\n");
            }
      }

}

/* Name: StartTask2
 * Description: Creates Task2 as an OS task. Task2 runs Task2Main.
 * Inputs: none
 * Outpus: none
 */
void StartTask2(void) {
      BYTE err = OS_NO_ERR;

      err = OSTaskCreatewName(      Task2Main,
                  (void *)NULL,
                  (void *) &Task2Stk[USER_TASK_STK_SIZE],
                  (void *) &Task2Stk[0],
                  TASK2_PRIO, "Task 2" );
      if(err) {
            iprintf("Task Creation Error in StartTask2\n");
      }
}
```

```
/* Name: Task2Main
 * Description: Main function for Task2. Puts character 'b' on second row of
LCD
 *          screen. Location is based on DIP switch values
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void  Task2Main( void * pd) {
      while (1) {
            if (OSSemPend(&theSemaphore2, 0) == OS_TIMEOUT)
                  iprintf("Timeout waiting for Semaphore 2\n");

            myLCD.Clear(LCD_BOTH_SCR);

            if (getdipsw() == DIP_SWITCH_ON) {
                  myLCD.MoveCursor(LCD_BOTH_SCR, LCD_BOTTOM_BEGIN);
            } else {
                  myLCD.MoveCursor(LCD_BOTH_SCR, LCD_BOTTOM_END);
            }


            myLCD.PrintString(LCD_UPPER_SCR, "b");
            OSTimeDly(TICKS_PER_SECOND*.5);

            if (getdipsw() == DIP_SWITCH_ON) {
                  if (OSSemPost(&theSemaphore3) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 3\n");
            } else {
                  if (OSSemPost(&theSemaphore1) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 1\n");
            }
      }
}

/* Name: StartTask3
 * Description: Creates Task3 as an OS task. Task3 runs Task3Main.
 * Inputs: none
 * Outpus: none
 */
void StartTask3(void) {
      BYTE err = OS_NO_ERR;

      err = OSTaskCreatewName(      Task3Main,
                  (void *)NULL,
                  (void *) &Task3Stk[USER_TASK_STK_SIZE],
                  (void *) &Task3Stk[0],
                  TASK3_PRIO, "Task 3" );
      if(err) {
            iprintf("Task Creation Error in StartTask3\n");
      }
}
```

```
/* Name: Task3Main
 * Description: Main function for Task3. Puts character 'c' on second row of
LCD
 *          screen. Location is based on DIP switch values
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void  Task3Main( void * pd) {
      while (1) {
            if (OSSemPend(&theSemaphore3, 0) == OS_TIMEOUT)
                  iprintf("Timeout waiting for Semaphore 3\n");

            myLCD.Clear(LCD_BOTH_SCR);

            if (getdipsw() == DIP_SWITCH_ON) {
                  myLCD.Home(LCD_BOTH_SCR);
            } else {
                  myLCD.MoveCursor(LCD_BOTH_SCR, LCD_TOP_END);
            }


            myLCD.PrintString(LCD_LOWER_SCR, "c");
            OSTimeDly(TICKS_PER_SECOND*.5);

            if (getdipsw() == DIP_SWITCH_ON) {
                  if (OSSemPost(&theSemaphore4) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 4\n");
            } else {
                  if (OSSemPost(&theSemaphore2) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 2\n");
            }
      }
}


/* Name: StartTask4
 * Description: Creates Task4 as an OS task. Task4 runs Task4Main.
 * Inputs: none
 * Outpus: none
 */
void StartTask4(void) {
      BYTE err = OS_NO_ERR;

      err = OSTaskCreatewName(      Task4Main,
                  (void *)NULL,
                  (void *) &Task4Stk[USER_TASK_STK_SIZE],
                  (void *) &Task4Stk[0],
                  TASK4_PRIO, "Task 4" );
      if(err) {
            iprintf("Task Creation Error in StartTask4\n");
      }
}
```

```
/* Name: Task4Main
 * Description: Main function for Task4. Puts character 'd' on second row of
LCD
 *          screen. Location is based on DIP switch values
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void  Task4Main( void * pd) {
      while (1) {
            if (OSSemPend(&theSemaphore4, 0) == OS_TIMEOUT)
                  iprintf("Timeour waiting for Semaphore 4\n");

            myLCD.Home(LCD_BOTH_SCR);
            myLCD.Clear(LCD_BOTH_SCR);

            if (getdipsw() == DIP_SWITCH_ON) {
                  myLCD.MoveCursor(LCD_BOTH_SCR, LCD_BOTTOM_BEGIN);
            } else {
                  myLCD.MoveCursor(LCD_BOTH_SCR, LCD_BOTTOM_END);
            }


            myLCD.PrintString(LCD_LOWER_SCR, "d");
            OSTimeDly(TICKS_PER_SECOND*0.5);

            if (getdipsw() == DIP_SWITCH_ON) {
                  if (OSSemPost(&theSemaphore1) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 1\n");
            } else {
                  if (OSSemPost(&theSemaphore3) == OS_SEM_OVF)
                        iprintf("Error posting to Semaphore 3\n");
            }
      }
}
```

From lcd.cpp:

```
/* Name: Init
 * Description: Initialises the hardware and the internal semaphore
 * to a known good state. The semaphores guarantee atomic instructions.
 * Inputs: none
 * Outputs: none
 */
void Lcd::Init(unsigned char screen)
{
	BYTE err = OS_NO_ERR;

	//set hardware lines to gpio, and put into initial state
	LCD_RS.function(PIN_GPIO); LCD_RS = 0;
	LCD_E1.function(PIN_GPIO); LCD_E1 = 0;
	LCD_E2.function(PIN_GPIO); LCD_E2 = 0;
	LCD_D0.function(PIN_GPIO); LCD_D0 = 0;
	LCD_D1.function(PIN_GPIO); LCD_D1 = 0;
	LCD_D2.function(PIN_GPIO); LCD_D2 = 0;
	LCD_D3.function(PIN_GPIO); LCD_D3 = 0;
	LCD_D4.function(PIN_GPIO); LCD_D4 = 0;
	LCD_D5.function(PIN_GPIO); LCD_D5 = 0;
	LCD_D6.function(PIN_GPIO); LCD_D6 = 0;
	LCD_D7.function(PIN_GPIO); LCD_D7 = 0;

	//Initialise Semaphore here and not in constructor to avoid
	// and endless trap loop
	if (sem_inited == FALSE) {
		err = OSSemInit(&sem,1);
		if(err){
			LCD_DEBUG(iprintf("Error In SemInit\n"));
		} else {
			sem_inited = TRUE;
		}
	}
	if (sem_inited == TRUE){
		err = OSSemPend(&sem, WAIT_FOREVER);
		if (err == OS_NO_ERR)
		{
			// exercise 2: decode and replace the send_cmd lines
			//                      turn the blink on in both upper and
lower screens
			switch (screen)
			{
				case LCD_UPPER_SCR :
					send_cmd(LCD_UPPER_SCR, CMD_FUNCTION|
						FUNCTION_8BIT|FUNCTION_2LINE|
						FUNCTION_5x8); // 56
					send_cmd(LCD_UPPER_SCR, CMD_CLEAR);
						// 1 -- CMD_CLEAR
					send_cmd(LCD_UPPER_SCR, CMD_DISPLAY|DISPLAY_ON|
						DISPLAY_NOCURSOR|DISPLAY_BLINK); // 12

					send_cmd(LCD_UPPER_SCR, CMD_ENTRY_MODE|
						ENTRY_CURSOR_INC|ENTRY_NOSHIFT); // 6
					send_cmd(LCD_UPPER_SCR, CMD_SHIFT|SHIFT_CURSOR|
						SHIFT_RIGHT); // 20
```

```
                        send_cmd(LCD_UPPER_SCR, CMD_HOME);

                        break;
                    case LCD_LOWER_SCR :
                        send_cmd(LCD_LOWER_SCR, CMD_FUNCTION|
                            FUNCTION_8BIT|FUNCTION_2LINE|FUNCTION_5x8);
                        send_cmd(LCD_LOWER_SCR, CMD_CLEAR);
                        send_cmd(LCD_LOWER_SCR, CMD_DISPLAY|DISPLAY_ON|
                            DISPLAY_NOCURSOR|DISPLAY_BLINK);

                        send_cmd(LCD_LOWER_SCR, CMD_ENTRY_MODE|
                            ENTRY_CURSOR_INC|ENTRY_NOSHIFT);
                        send_cmd(LCD_LOWER_SCR, CMD_SHIFT|
                            SHIFT_CURSOR|SHIFT_RIGHT);
                        send_cmd(LCD_LOWER_SCR, CMD_HOME);
                        break;
                    case LCD_BOTH_SCR :
                        send_cmd(LCD_BOTH_SCR, CMD_FUNCTION|
                            FUNCTION_8BIT|FUNCTION_2LINE|FUNCTION_5x8);
                        send_cmd(LCD_BOTH_SCR, CMD_CLEAR);
                        send_cmd(LCD_BOTH_SCR, CMD_DISPLAY|DISPLAY_ON|
                            DISPLAY_NOCURSOR|DISPLAY_BLINK);

                        send_cmd(LCD_BOTH_SCR, CMD_ENTRY_MODE|
                            ENTRY_CURSOR_INC|ENTRY_NOSHIFT);
                        send_cmd(LCD_BOTH_SCR, CMD_SHIFT|
                            SHIFT_CURSOR|SHIFT_RIGHT);
                        send_cmd(LCD_BOTH_SCR, CMD_HOME);
                        break;
                    default:
                        LCD_DEBUG(iprintf("Lcd::Init incorrect
screen\n"));
                }
                OSSemPost(&sem);
            } else {
                    LCD_DEBUG(iprintf("Lcd::Init sem error\n"));
                }
            }
}
```