ECE 315: Computer Interfacing

Lab #1: Introduction to uC/OS on an MCF54415 Microcontroller: Interfacing to a Graphical LCD

Conducted on: January 30, 2020

Conducted by: Kathleen Pescador, Abuni Gaiya

Lab Section: H41

Table of Contents

Click the requirement you would like to view

Abstract

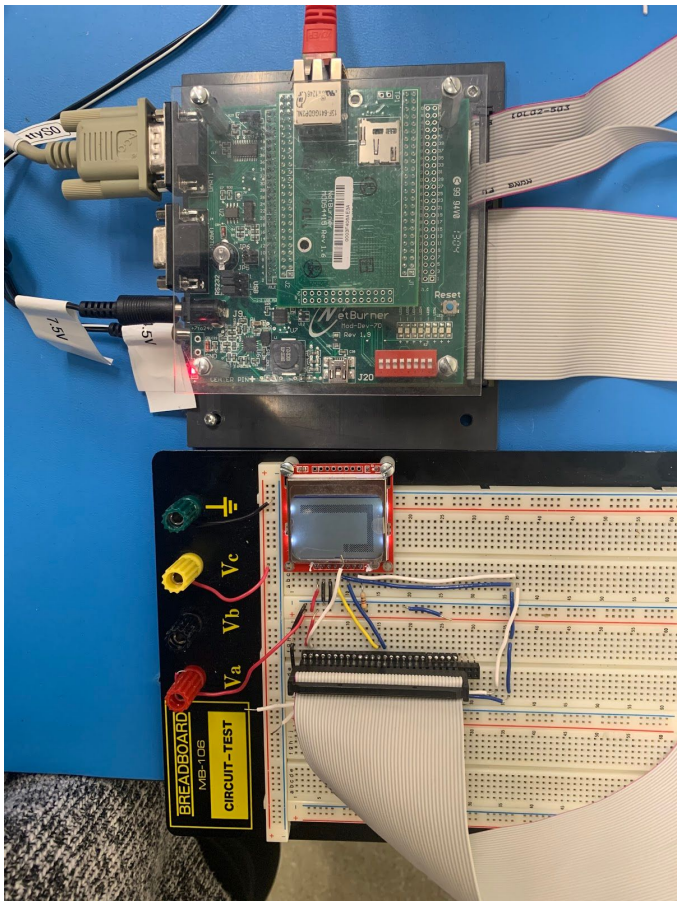
For this lab, our goal was to get familiar with the Freescale MCF54415 microcontroller based on the V4 Coldfire CPU core. We were to gain experience with designing software for the uC/OS multitasking environment. This lab also introduces us with task synchronization using semaphores and with interfacing a graphical LCD over general-purpose input/output (GPIO) pins and the serial peripheral interface (SPI) pins. Specifically in this lab, we will work with the Nokia 5110 graphical display, which is used in cell phones before the smartphones exist. This lab consists of four parts. We would have to wire our LCD according to the schematic and suggested floor plan that are provided in eClass. In the first part of the lab, we were to verify that our LCD hardware is functioning properly. We could do this so by using the lab1 files from eClass to display a welcome message. For the second part of the lab, we would have to modify the provided code to set the contrast of the LCD darker. We would also need to remove all of the magic numbers and replace them with more easily changed symbolic constants that have already been defined in the provided code from eClass. On the other hand, the third part of the lab requires us to create three tasks that lets a sprite rotate across the LCD margins. These three tasks will be integrated in the last part of the lab, where we would need to guarantee proper execution of lower priority tasks that would enable us to rotate the sprite in a clockwise direction. This would require us to use semaphores to guarantee the order of display.

Design

Exercise 1

The objective of this part of the Lab was to verify that the that our design form the prelab worked. For the circuit design we strictly followed the suggested circuit wiring posted on eclass. We made sure that our circuit was properly wired by getting a TA to check our design. We gave more details on verification in the test section. After verifying our circuit we ran the provided code obtained from eclass and we checked that it displayed the text “Welcome”. Figure 1 shows our design for the circuit.

Figure 1 : Picture hardware circuit layout



Exercise 2

In this part of the lab we had to change the contrast of the LCD screen to a darker mode and change all the magic numbers to use actual variable names . This part required us to understand how the LCD commands worked and utilize instructions commands explained in the 48 x 84 pixels matrix LCD controller/driver instruction set explain on page 14 of the LCD manual obtained from eclass. We set the bit H to 1, which enabled us to use the extended instruction set. With the bit H set to 1 , we send commands 0x20 to call the command function set. We sent some more commands to change the contrast of the LCD display. Table 1 highlights the commands we sent to change the contrast and their uses and the code for this part could be found in the appendix. As shown in the appendix and *Table 1* we changed the magic numbers for the init function to their corresponding variable names as already specified in the LCD.h file. We also change the magic numbers where applicable for all the other parts of our code.

Table 1 : Table showing important commands used for changing the LCD contrast

Commands	**Magic Numbers**	**Use**
CMD_FUNCTION_SET | OTP_EXT_INSTR	0x20 | 0x1	It calls the extended commands
CMD_SET_VOP | OPT_CONSTRAST	0x8 | 0x4	Makes contrast darker
CMD_TEMP_CNTRL | OPT_VLCD_COEFF_0	0x4 | 0x0	Sets temperature Coefficient to VLCD temperature coefficient 0
CMD_BIAS_SYSTEM| OPT_NORMAL	0x 10 | 0x4	Sets LCD Bias mode to option 3

Exercise 3

The third part of the lab involves creating three tasks that would let a sprite rotate across the margins of the LCD. We first created a sprite that is 7 pixels in the horizontal direction and 8 pixels in the vertical direction using a web-based application, www.piskelapp.com. We then used that sprite to be displayed on the LCD and used it to implement the three tasks. Task 1 is responsible for moving the sprite along the top most part of the LCD and moving it down once it hits the rightmost end, as shown in *Figure 2*.

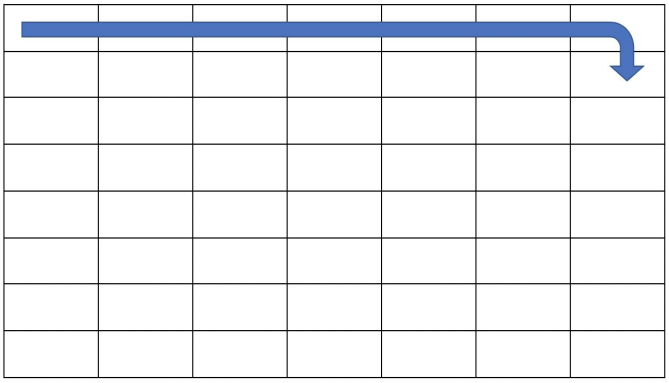


Figure 2: Sprite Movement for Task 1 on the LCD Screen

Task 2 involves two cases, wherein it is responsible for moving the sprite along the vertical direction. One of the cases is responsible for the movement of the sprite after Task 1. This task would enable the sprite to move down from the upper left corner to the lower left corner, shown in *Figure 3*.

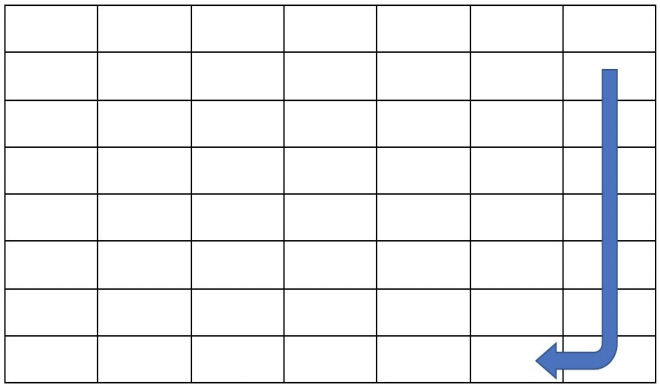


Figure 3: Sprite Movement for Task 2 (Case I) on the LCD Screen

Once it hits that corner, it would then have to be moved left to implement Task 3, shown in *Figure 4*.

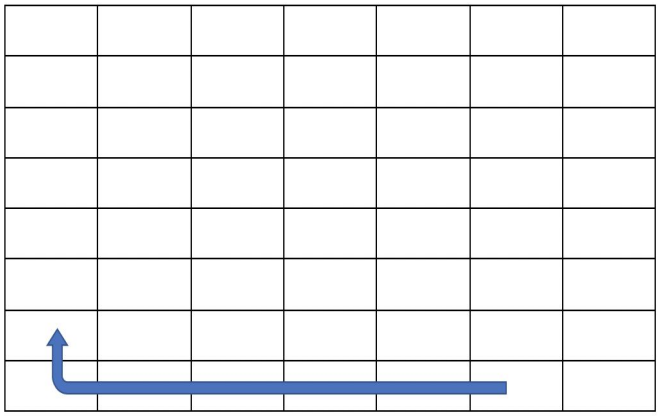


Figure 4: Sprite Movement for Task 3 on the LCD Screen

The other case of Task 2 gets implemented after Task 3, in which it has to move the sprite from the lower right corner of the LCD to the upper right corner. This is shown in *Figure 5*. As this exercise does not require us to use semaphore tasks to execute the three tasks in order, the result of this code would be completely seen in Exercise 4.

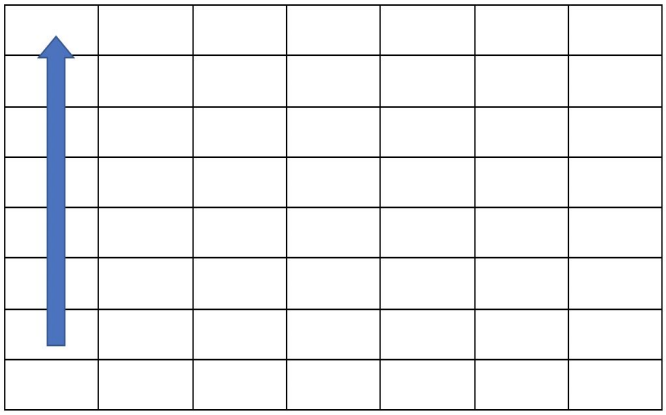


Figure 5: Sprite Movement for Task 2 (Case II) on the LCD Screen

Exercise 4

The last part of the lab used the three tasks that were created from Exercise 3. These tasks were ordered based on their priority by using semaphores. Calling LCD commands from different tasks can be managed by using these semaphores as they can control program flow and guarantee that the commands from the LCD can execute atomically. In the lcd.cpp file, we first had to initialize the OS_SEM tasks, as shown in *Figure 6*.

```
OS_SEM Semaphore1;
OS_SEM Semaphore2;
OS_SEM Semaphore3;
OS_SEM Semaphore4;
```

Figure 6: Initialization OS_SEM tasks

We then had to initialize semaphores with their respective tasks, shown in *Figure 7*.

```
OSSemInit (&Semaphore1 , 0);
OSSemInit (&Semaphore2 , 0);
OSSemInit (&Semaphore3 , 0);
OSSemInit (&Semaphore4 , 0);
```

Figure 7: Initialization of Semaphores

After the initialization of the semaphores, we used the OSSemPend() and OSSemPost() to switch between tasks to execute the order in which the sprite could move along the margins of the LCD in a clockwise direction. When the OSSemPend() is called and the value of the semaphore is 0, OSSemPend() places the calling task in the waiting list for the semaphore. The task will then wait forever until the owner of the semaphore executes the OSSemPost(). This code is shown under the *Appendix: Design Code for Exercise 4* section of this lab report.

Testing

Case	Description	Expected Results	Experimental Result	Conclusion
Exercise 1: Hardware	Build a project, and verify that the LCD hardware is functioning correctly. Verify that our environment was connected to the netBurner MOD554415 hardware.	LCD lights up and Displays "Welcome". Mtty displays : IP -> *10.0.0.102* Configured -> *255.255.255.0* MAC Address -> *00:03:f4:06:ae:3a* App name -> *"Abuni and Kathleen"*	LCD lights up and Displays "Welcome" Mtty displays : IP -> *10.0.0.102* Configured -> *255.255.255.0* MAC Address -> *00:03:f4:06:ae:3a* App name -> *"Abuni and Kathleen"*	Test Passed
Exercise 2: Modifying Contrast	Modify the initialization sequence for the LCD and remove all the magic numbers in the LCD::init_lcd method.	The contrast of the LCD becomes darker, making any data sent to the screen more visible.	The message "Welcome to our App" was displayed more visibly due to its darker contrast.	Test Passed
Exercise 3: Pattern Verification	We checked that the LCD displays a single sprite in a rotating checkerboard pattern around the margins of the LCD display and that checkerboard box that will display the sprite at the extreme margins of the screen. The box should be displayed in a clockwise rotation. Clear the screen once every complete rotation. We tested each of the three tasks using a variable that we set to the next task the previous task is completed.	Displays a squared sprite that rotated clockwise round the border of the screen leaving a trail until it completes a loop and then starts again.	Displays a squared sprite that rotated clockwise round the border of the screen leaving a trail until it completes a loop and then starts again.	Test Passed

	Note: We tested this part before implementing the semiphores			
Exercise 4: Task switching verification	Guarantee proper execution of lower priority tasks and deal with contention for the LCD. The box pattern should now display correctly and rotate in a counter-clockwise direction. Use semaphores to guarantee the order of display	The LCD displays the “square” sprite trailing along the margins of the LCD display in a clockwise direction. The sprite starts from the top left corner of the screen, leaving a trail until it returns back to the original position and clears the screen when it goes for the next round again.	The LCD displayed the “square” sprite moving along the margins of the LCD display in a clockwise direction. It leaves a trail until it does a complete round, then it clears. After reaching the original position, the sprite starts to do the exact same movement it did beforehand.	Test Passed

Conclusion

In conclusion, we learned a lot from this lab. We were able to familiarize ourselves with the NetBurner MOD54415 and put into practice many µC/OS RTOS concepts , such as semaphores. We were successfully able to achieve the main goal of this lab which was to display a single sprite in a rotating checkerboard pattern around the margins of the LCD display. This required us to familiarize ourselves with the LCD commands and to make our code easier to understand we had to remove all the magic numbers. After each part of this lab, we thoroughly tested each of the requirements and all our tests passed. At the end of this lab we were confident to say that we had designed software for the µC/OS multitasking environment , interface a graphical LCD and most importantly synchronize different tasks using semaphores.

Appendix:

Design Code for Exercise 2:

```
void LCD::init_lcd(void) {
	LCD_RESET = 0;
	OSTimeDly(1); // minimal possible delay
	LCD_RESET = 1;

	// Insert your ex 2 code modifications here
	// H = 1
	send_cmd(CMD_FUNCTION_SET |OPT_EXT_INSTR); // Tell LCD extended Commands
follow
//	send_cmd(CMD_SET_VOP | OPT_CONTRAST_LIGHT); // make contrast light
//	OSTimeDly(400);
	send_cmd(CMD_SET_VOP | OPT_CONTRAST_DARK); // make contrast dark
	send_cmd(CMD_TEMP_CNTRL |OPT_VLCD_COEFF_0); //Set Temperature Coefficient
	send_cmd(CMD_BIAS_SYSTEM |OPT_N_3); //LCD bias mode

	// H = 0
	send_cmd(CMD_FUNCTION_SET |OPT_BASIC_INSTR); // We must send this before
modifying the display control
	send_cmd(CMD_DISPLAY_CONTROL |OPT_NORMAL); // Set display control to normal
mode
	// End ex 2 modifications
	move(char_index[LINE1_ORIGIN]);
}
```

Design Code for Exercise 4:

```
#include <predef.h>
#include <stdio.h>
#include <ctype.h>
#include <startnet.h>
#include <autoupdate.h>
#include <smarttrap.h>
#include <taskmon.h>on in
#include <NetworkDebug.h>
#include <pinconstant.h>
#include <pins.h>
#include <basictypes.h>
#include "LCD.h"
#include "bitmaps.h"
```

```
#include "error_wrapper.h"
#include "point.h"


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


/* 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 ) ) );


const char * AppName="Abuni and Kathleen";


/* User task priorities always based on MAIN_PRIO */
/* The priorities between MAIN_PRIO and the IDLE_PRIO are available */
#define TASK1_PRIO   MAIN_PRIO + 1
#define TASK2_PRIO   MAIN_PRIO + 2
#define TASK3_PRIO   MAIN_PRIO + 3

#define WAIT_FOREVER 0


LCD myLCD;
const BYTE dollar[] = {0x00,0x24,0x2A, 0x7f, 0x2a, 0x12,0x00};
const BYTE sprite1[] = {0xAA, 0x55,0xAA, 0x55,0xAA, 0x55,0xAA}; //random sprite

OS_SEM Semaphore1;
OS_SEM Semaphore2;
OS_SEM Semaphore3;
OS_SEM Semaphore4;

void UserMain(void * pd) {
             BYTE err = OS_NO_ERR;
             //char * welcome = "I";
```

```
    InitializeStack();
    OSChangePrio(MAIN_PRIO);
    EnableAutoUpdate();
    StartHTTP();
    EnableTaskMonitor();

    #ifndef _DEBUG
    EnableSmartTraps();
    #endif

    #ifdef _DEBUG
    InitializeNetworkGDB_and_Wait();
    #endif

    iprintf("Application started: %s\n",AppName );

    myLCD.Init();
    myLCD.Clear();
    myLCD.Home();

    OSSemInit (&Semaphore1 , 0);
    OSSemInit (&Semaphore2 , 0);
    OSSemInit (&Semaphore3 , 0);
    OSSemInit (&Semaphore4 , 0);

    StartTask1();
    StartTask2();
    StartTask3();
    while (1) {
        OSTimeDly(TICKS_PER_SECOND);
       }
}


/* Name: StartTask1
 * Description: Creates the task main loop.
 * Inputs: none
 * Outputs: none
 */
void StartTask1(void) {

             BYTE err = OS_NO_ERR;
             err = display_error( "StartTask1 fail:",
                                                          OSTaskCreatewName(
      Task1Main,
                                                          (void *)NULL,
                                                          (void *)
```

```
&Task1Stk[USER_TASK_STK_SIZE],
                                                        (void *) &Task1Stk[0],
                                                        TASK1_PRIO, "Task 1" ));
}

/* Name: StartTask2
 * Description: Creates the task main loop.
 * Inputs: none
 * Outputs: none
 */
void StartTask2(void) {

            BYTE err = OS_NO_ERR;
            err = display_error( "StartTask2 fail:",
                  OSTaskCreatewName(              Task2Main,
                                                        (void *)NULL,
                                                        (void *)
&Task2Stk[USER_TASK_STK_SIZE],
                                                        (void *) &Task2Stk[0],
                                                        TASK2_PRIO, "Task 2" ));
}

/* Name: StartTask3
 * Description: Creates the task main loop.
 * Inputs: none
 * Outputs: none
 */
void StartTask3(void) {
            BYTE err = OS_NO_ERR;
            err = display_error( "StartTask3 fail:",
                                                        OSTaskCreatewName(
      Task3Main,
                                                        (void *)NULL,
                                                        (void *)
&Task3Stk[USER_TASK_STK_SIZE],
                                                        (void *) &Task3Stk[0],
                                                        TASK3_PRIO, "Task 3" ));
}

/* Name: Task1Main
 * Description:
 * Inputs:  void * pd -- pointer to generic data . Currently unused.
 * Outputs: none
 */
void Task1Main( void * pd) {
            BYTE err = OS_NO_ERR;
            /* place semaphore usage code inside the loop */
```

```
            while(1){
                        for (int i = 0; i < LINE2_ORIGIN; i++){

myLCD.DrawChar(sprite1,char_index[i]);
                                                OSTimeDly(TICKS_PER_SECOND);

                                    }
                                    myLCD.DrawChar(sprite1,char_index[LINE2_END]);
                                    OSSemPost(&Semaphore1);
                                    OSSemPend(&Semaphore4, WAIT_FOREVER);
                                    myLCD.DrawChar(sprite1,char_index[LINE2_ORIGIN]);
                                    OSTimeDly(TICKS_PER_SECOND);
                                    myLCD.Clear();
            }

}

void Task2Main( void * pd) {
            BYTE err = OS_NO_ERR;
            /* place semaphore usage code inside the loop */
            while(1){
                        OSSemPend(&Semaphore1, WAIT_FOREVER);
                              myLCD.DrawChar(sprite1,char_index[LINE3_END]);
                              OSTimeDly(TICKS_PER_SECOND);
                              myLCD.DrawChar(sprite1,char_index[LINE4_END]);
                              OSTimeDly(TICKS_PER_SECOND);

                              OSSemPost(&Semaphore2);
                              OSSemPend(&Semaphore3, WAIT_FOREVER);
                              myLCD.DrawChar(sprite1,char_index[LINE4_ORIGIN]);
                              OSTimeDly(TICKS_PER_SECOND);
                              myLCD.DrawChar(sprite1,char_index[LINE3_ORIGIN]);

                              OSSemPost(&Semaphore4);
            }
}

void Task3Main( void * pd) {
            BYTE err = OS_NO_ERR;
            while(1){
                        OSSemPend(&Semaphore2, WAIT_FOREVER);
                              myLCD.DrawChar(sprite1,char_index[LINE5_END]);
                              /* place semaphore usage code inside the loop */
                              for (int i = LINE6_END; i >= LINE6_ORIGIN ; i--){

myLCD.DrawChar(sprite1,char_index[i]);
                                                OSTimeDly(TICKS_PER_SECOND);
```

```
                                }
                                myLCD.DrawChar(sprite1,char_index[LINE5_ORIGIN]);
                                OSSemPost(&Semaphore3);
            }
}
```