ENCM 511 - Laboratory 3 & 4 Report - Source Code

Kyle Derby MacInnis & Maheen Hafeez

December 7, 2014

Abstract

The following report covers the necessary material required for laboratory 3 and 4, and excepting the parts which have been taken from laboratory materials provided in class, all work contained in this report is that of our own. The Smith 20 Minute Rule was adhered to, and we have not copied any work or received help which I did not then build off of ourselves.

The purpose of this report is to provide the source code used by us during labs 3 and 4. The first portion covers Lab 3 and contains source code designed to read input from the GPIO bus and determine whether or not certain signal lines are active high or low, and to record the length of time for which they remain in those states. This is then used to measure a digital temperature sensor and display the temperature on the Blackfin LEDs. The second portion of this report covers Lab 4 and contains source code incorporated from Lab 3 alongside new code designed to communicate using the SPI functionality of Blackfin to control a small external LCD screen. The LCD screen is then used to alternate between a chosen text phrase and the temperature calculated from the Temperature sensor.

Note to Marker: Some of the code presented in this report contains defects we as a group were unable to locate. We are uncertain of the exact nature of the problems, and they are most likely small mistakes. They could possibly have been hardware errors or possibly race conditions. These problems were present only within the code for Lab 4 while our Lab 3 code functioned as expected. We apologize for the existence of these errors and we hope that they are not egregious.

Laboratory 3 - Header File: Lab3_Main.h

The following is the source code relating to the Lab 3 header file. In it are the prototype definitions for the functions defined in the their corresponding CPP source files.

```
/*********************************************************************************
* Author: Kyle Derby MacInnis & Maheen Hafeez
* Date: Tue 2014/11/18 at 02:48:26 PM
* File Type: uTTCOS Main Header File
*********************************************************************************/

#ifndef LAB3_UTTCOS_H
#define LAB3_UTTCOS_H

// Global Temperature Variable
extern volatile float TempC;

// Manual Switch Monitoring Thread
void SW3_Thread();

// State Machine Temperature Sensor Thread
void SW3_TMP03();

// Timer Based Temperature Sensor Thread
void Calc_Temp();

// Initialize Blackfin GP Timer (TIMER2)
void Init_Timer();

// Start Timer
void Start_Timer();

// Modified LED Flash Function
void Lab3_FlashLED(unsigned int LEDS, bool FASTER);

#endif
```

Laboratory 3 - Thread Source File: Lab3_Threads.cpp

The following code outlines the uTTCOS threads used in Lab 3. There are three separate threads designed to operate differently. The first thread deals with reading the manual operation of the switches as they pertain to active high and active low times. The second thread uses an array to store measured values from an external temperature sensor that are then approximated to give temperature. There is also a function providing a modified LED state machine used to flash an LED at two different speeds as needed.

```
/*
 *      Author: Kyle Derby MacInnis & Maheen Hafeez
 *  Created on: Dec 2, 2014
 *   File Type: Lab 3 Thread Functions Source File
 */

#include "Lab3_uTTCOS.h"
#include <blackfin.h>
#include <uTTCOS2013/uTTCOS.h>
#include <stdio.h>

// SWITCH BITMASKS
#define SW1_BITMASK      0x1
#define SW2_BITMASK      0x2
#define SW3_BITMASK      0x4
#define SW4_BITMASK      0x8

// LED BITMASK
#define LED1_BITMASK     0x1
#define LED2_BITMASK     0x2
#define LED3_BITMASK     0x4
#define LED4_BITMASK     0x8
#define LED5_BITMASK     0x10
#define LED6_BITMASK     0x20

// Lab 3 Array
#define ARRAY_SIZE       20

// Thread States
enum {INIT = 1, LO_LO, LO_HI, HI_HI, HI_LO, LO_LO_2, LO_HI_2, HI_HI_2, HI_LO_2};

// SW3_THREAD STATE MACHINE DEFINITIONS
#define INIT_STATE       case INIT
#define LO_LO_STATE      case LO_LO
#define LO_HI_STATE      case LO_HI
#define HI_HI_STATE      case HI_HI
#define HI_LO_STATE      case HI_LO
#define LO_LO_2_STATE    case LO_LO_2
#define LO_HI_2_STATE    case LO_HI_2
#define HI_HI_2_STATE    case HI_HI_2
#define HI_LO_2_STATE    case HI_LO_2
#define ERROR_STATE      default

// SW3 Thread - Reads Length of SW3 Button Press
void SW3_Thread(void)
{
    // Hi and Lo Counters SW3
    static unsigned int SW3_Hi = 0;
    static unsigned int SW3_Lo = 0;

    // Hi and Lo Length Containers
    static unsigned int Max_SW3_Hi = 0;
    static unsigned int Max_SW3_Lo = 0;

    static int next_State = INIT;

    unsigned int SwitchValue = NULL;

    switch(next_State)
    {
        // Initial State
        INIT_STATE:
            // Determine Proper State to Start
```

```
            SwitchValue = uTTCOS_ReadSwitches();
            if((SwitchValue & SW3_BITMASK) == SW3_BITMASK)
                next_State = LO_HI;
            else
                next_State = LO_LO;
        break;

        // Active Low State Stuff
        LO_LO_STATE:
            // Check SW3 State
            SwitchValue = uTTCOS_ReadSwitches();
            // If SW3 High go to Rising Edge State
            if((SwitchValue & SW3_BITMASK) == SW3_BITMASK)
                next_State = LO_HI;
            else
                next_State = LO_LO;
            // Reset Hi Counter
            SW3_Hi = 0;
            // Increment Lo Counter
            SW3_Lo++;
        break;

        // Rising Edge State Stuff
        LO_HI_STATE:
            // Record Lo Counter
            Max_SW3_Lo = SW3_Lo;
            next_State = HI_HI;
        break;

        // Active High State Stuff
        HI_HI_STATE:
            // Read Switch Values
            SwitchValue = uTTCOS_ReadSwitches();
            // If SW3 Low go to Falling Edge State
            if((SwitchValue & SW3_BITMASK) == 0x0)
                next_State = HI_LO;
            else
                next_State = HI_HI;
            // Reset Lo Counter
            SW3_Lo = 0;
            // Increment High Counter
            SW3_Hi++;
        break;

        // Falling Edge State Stuff
        HI_LO_STATE:
            // Record Hi Counter
            Max_SW3_Hi = SW3_Hi;
            next_State = LO_LO;
        break;

        // Should not be here - Go back to INIT
        ERROR_STATE:
            next_State = INIT;
        break;
    }

    // If SW1 is Low (SW1 Released)
    if((uTTCOS_ReadSwitches() & SW1_BITMASK) == 0x0)
    {
        // Write Time SW3 Was Pressed in 1/4 seconds to LEDS
        unsigned int LED_Value = ((Max_SW3_Hi / 3) & 0x3F);
        uTTCOS_WriteLED(LED_Value);
    }
    // If SW1 is High (SW1 Pressed)
    else
    {
        // If Long Press( +10secs) - Slow LED Flashing
        if((SW3_Hi / 3) >= 0x28 )
        {
            //Flash LEDS Slow
            Lab3_FlashLED(LED1_BITMASK, false);
        }
```

```
        else
        {
            // Flash LEDS Fast
            Lab3_FlashLED(LED1_BITMASK, true);
        }
    }
}

// Temperature Sensor Thread - Read TMP03 from SW4
void TMP03_Thread(void)
{
    // Hi and Lo Counters SW3
    static unsigned int SW4_Hi = 0;
    static unsigned int SW4_Lo = 0;

    // Max Hi and Lo Length Container
    static unsigned int Max_SW4_Hi = 0;
    static unsigned int Max_SW4_Lo = 0;

    // Flash approximately every 1/2 second
    static unsigned int FlashCounter = 0;
    unsigned int FlashCountMax = 150;

    // Array for Hi and Lo Values
    static unsigned int ArrayHi[ARRAY_SIZE];
    static unsigned int ArrayLo[ARRAY_SIZE];

    // Index for Arrays
    static unsigned int ArrayHiCnt = 0;
    static unsigned int ArrayLoCnt = 0;

    static int next_State = INIT;

    unsigned int SwitchValue = NULL;

    // State Machine
    switch(next_State)
    {
        // Initial State
        INIT_STATE:
            // Determine Proper State to Start
            SwitchValue = uTTCOS_ReadSwitches();
            if((SwitchValue & SW4_BITMASK) == SW4_BITMASK)
                next_State = LO_HI;
            else
                next_State = LO_LO;
        break;

        // Active Low State Stuff
        LO_LO_STATE:
            // Read Switch Values
            SwitchValue = uTTCOS_ReadSwitches();
            // If SW4 is Hi goto Rising Edge State
            if((SwitchValue & SW4_BITMASK) == SW4_BITMASK)
                next_State = LO_HI;
            else
                next_State = LO_LO;
            // Reset Hi Counter
            SW4_Hi = 0;
            // Increment Low Counter
            SW4_Lo++;
        break;

        // Rising Edge State Stuff
        LO_HI_STATE:
            Max_SW4_Lo = SW4_Lo;
            next_State = HI_HI;
        break;

        // Active High State Stuff
        HI_HI_STATE:
            // Read Switch Values
            SwitchValue = uTTCOS_ReadSwitches();
```

```
            // If Active Low, go to Falling Edge State
            if((SwitchValue & SW4_BITMASK) == 0x0)
                next_State = HI_LO;
            else
                next_State = HI_HI;
            // Reset Lo Counter
            SW4_Lo = 0;
            // Increment Hi Counter
            SW4_Hi++;
        break;

        // Falling Edge State Stuff
        HI_LO_STATE:
            Max_SW4_Hi = SW4_Hi;
            next_State = LO_LO_2;
        break;

        // Second Active Low State
        LO_LO_2_STATE:
            // Increment Counter and Check State
            SwitchValue = uTTCOS_ReadSwitches();
            if((SwitchValue & SW4_BITMASK) == SW4_BITMASK)
                next_State = LO_HI_2;
            else
                next_State = LO_LO_2;
            SW4_Hi = 0;
            SW4_Lo++;
        break;

        // Second Rising Edge State
        LO_HI_2_STATE:
            Max_SW4_Lo = SW4_Lo;
            next_State = HI_HI_2;
            // Store Lo Count into Array
            ArrayLo[ArrayLoCnt] = Max_SW4_Lo;
            ArrayLoCnt++;
        break;

        // Second Active High State
        HI_HI_2_STATE:
            // Increment Counter and Check State
            SwitchValue = uTTCOS_ReadSwitches();
            if((SwitchValue & SW4_BITMASK) == 0x0)
                next_State = HI_LO_2;
            else
                next_State = HI_HI_2;
            SW4_Lo = 0;
            SW4_Hi++;
        break;

        // Second Falling Edge State
        HI_LO_2_STATE:
            Max_SW4_Hi = SW4_Hi;
            next_State = LO_LO_2;
            // Store Hi Count in Array
            ArrayHi[ArrayHiCnt] = Max_SW4_Hi;
            ArrayHiCnt++;
        break;

        // Should not be here
        ERROR_STATE:
            next_State = INIT;
        break;
    }

    // Average T1 and T2 Values
    static unsigned int AverageT1 = 0;
    static unsigned int AverageT2 = 0;

    // If Hi Count Array is Filled
    if(ArrayHiCnt >= ARRAY_SIZE)
    {
        // Calculate Average T1
```

```
        for(int i = 0; i < ARRAY_SIZE; i++)
            AverageT1 += ArrayHi[i];
        AverageT1 = AverageT1 / ARRAY_SIZE;
        // Reset Hi Array Counter
        ArrayHiCnt = 0;
    }

    // If Lo Count Array is Filled
    if(ArrayLoCnt >= ARRAY_SIZE)
    {
        // Calculate Average T2
        for(int i = 0; i < ARRAY_SIZE; i++)
            AverageT2 += ArrayLo[i];
        AverageT2 = AverageT2 / ARRAY_SIZE;
        // Reset Lo Array Counter
        ArrayLoCnt = 0;
    }

    // If SW1 is High (Up) and SW2 is High
    if(((uTTCOS_ReadSwitches() & SW1_BITMASK) == SW1_BITMASK) && ((uTTCOS_ReadSwitches()
     & SW2_BITMASK) == SW2_BITMASK))
    {
        // Write Time SW3 Was Pressed in 1/4 seconds to LEDS
        unsigned int LED_Value = ((AverageT1 / 3) & 0x3F);
        uTTCOS_WriteLED(LED_Value);
    }
    // If SW1 is Low (Up) and SW2 is Low
    else if(((uTTCOS_ReadSwitches() & SW1_BITMASK) == 0x0) && ((uTTCOS_ReadSwitches() &
    SW2_BITMASK) == 0x0))
    {
        // Roughly Flash Every 1/2 Second - Either with Slow or Fast StateMachine
        if(FlashCounter >= FlashCountMax)
        {
            // If Long Press - Slow LED Flashing
            if((SW4_Hi / 3) >= 0x2 )
            {
                //Flash LEDS Slow
                Lab3_FlashLED(LED1_BITMASK, false);
            }
            else
            {
                // Flash LEDS Fast
                Lab3_FlashLED(LED1_BITMASK, true);
            }
            FlashCounter = 0;
        }
        FlashCounter++;
    }
    // If SW1 is High and SW2 is Low
    else if(((uTTCOS_ReadSwitches() & SW1_BITMASK) == SW1_BITMASK) && ((
    uTTCOS_ReadSwitches() & SW2_BITMASK) == 0x0))
    {
        // Write Time SW3 Was Pressed in 1/4 seconds to LEDS
        unsigned int LED_Value = ((AverageT2 / 3) & 0x3F);
        uTTCOS_WriteLED(LED_Value);
    }
    // If SW1 is Low, and SW2 is High
    else
    {
        // Initial Temperature
        float C_Value = 0;

        // Calculate Temperature (Avoid div. by zero)
        if( AverageT2 == 0)
        {
            C_Value = 0;
        }
        else
        {
            // 20 Degree Reference Value
            C_Value = (235 - (400*((float)AverageT1/(float)AverageT2)));
        }
        // Display to LEDs
```

```
        uTTCOS_WriteLED((unsigned int)C_Value);
    }
}

// Flash LEDS Slow or Fast
void Lab3_FlashLED(unsigned int LEDS, bool FASTER)
{
    // LED States
    enum{LED_INIT, LED_ON, LED_ON2, LED_ON3, LED_OFF, LED_OFF2, LED_OFF3};

    static int next_State = LED_INIT;

    // LED State Machine
    switch(next_State)
    {
    // Initial State
    case LED_INIT:
        next_State = LED_ON;
        break;

    // LED On
    case LED_ON:
        uTTCOS_WriteLED(LEDS & 0x3F);
        if(FASTER)
            next_State = LED_OFF;
        else
            next_State = LED_ON2;
        break;

    // LED On (Slower Flash)
    case LED_ON2:
        if(FASTER)
            next_State = LED_OFF;
        else
            next_State = LED_ON3;
        break;

    // LED On (Slower Flash)
    case LED_ON3:
        next_State = LED_OFF;
        break;

    // LED Off
    case LED_OFF:
        uTTCOS_WriteLED(0x0);
        if(FASTER)
            next_State = LED_ON;
        else
            next_State = LED_OFF2;
        break;

    // LED Off (Slower Flash)
    case LED_OFF2:
        if(FASTER)
            next_State = LED_ON;
        else
            next_State = LED_OFF3;
        break;

    // LED Off (Slower Flash)
    case LED_OFF3:
        next_State = LED_ON;
        break;
    }
}
```

Laboratory 3 - Timer Source File (CPP): Lab3_TimerFunctions.cpp

The following source code contains the function definitions for the lab #3 timer functions.

```
/*
 *      Author: Kyle Derby MacInnis & Maheen Hafeez
 *  Created on: Dec 2, 2014
 *   File Type: Lab 3 Timer Functions Source File
 */

#include "Lab3_uTTCOS.h"
#include <blackfin.h>
#include <uTTCOS2013/uTTCOS.h>
#include <stdio.h>

// Global Temperature Variable
volatile float TempC = 0;

// Initialize Timer
void Init_Timer()
{
    *pTIMER2_CONFIG = 0x000E;
    asm("ssync;");
}

// Start Timer
void Start_Timer()
{
    *pTIMER_ENABLE |= 0x4;
}

// Calculate Temperature from Timer
void Calc_Temp()
{
    // Read Width
    unsigned int T1 = *pTIMER2_WIDTH;

    // Read Period
    unsigned int T2 = (*pTIMER2_PERIOD) - (*pTIMER2_WIDTH);

    // Calculate Temperature
    TempC = 235 - 400*((float)T1/(float)T2);

    // Display Temperature to LEDS
    uTTCOS_WriteLED((unsigned int) TempC);
}
```

Laboratory 3 & 4 - Main Source File (CPP): Lab3_uTTCOSMain.cpp

The following source code is that which comprises the main code loop for Laboratory 3 and 4. It contains the main loop and calls all the uTTCOS threads used within the programs. The threads are a mixture of Lab 3 and Lab 4 threads with certain threads being present in both.

```
/*********************************************************************************
* Author: Kyle Derby MacInnis & Maheen Hafeez
* Date: Tue 2014/11/18 at 02:48:26 PM
* File Type: uTTCOS Main File
*********************************************************************************/

#include <stdio.h>
#include <uTTCOS2013/uTTCOS.h>
#include "Lab3_uTTCOS.h"
#include "Lab4.h"

// This timing MACRO may need adjusting
#define SECOND       uTTCOS_GetTickFrequency( )
#define TWELTH       SECOND/12

// uTTCOS Stuff
enum {USE_TIMER_OS = 1, USE_AUDIO_OS = 2, DEFAULT_OS = 2, INIT_BOTH_OS_USE_AUDIO_OS =
    3};
extern int whichOS;

// Main
void main(void)
{
    // Maximum number of Threads
    int maxNumberThreads = 15;    // Make at least 5 larger than the tasks you plan to
    add

    // Initialization
    uTTCOS_Init();
    uTTCOS_AddPreEmptiveThread(uTTCOS_AudioTalkThrough,  NO_DELAY, EVERY_TICK);
    uTTCOS_InitSwitches();
    uTTCOS_InitLED();

    // Load uTTCOS Threads
    // ------ Lab 3 Threads ------
    //uTTCOS_AddThread(SW3_Thread, NO_DELAY, TWELTH);
    //uTTCOS_AddThread(TMP03_Thread, NO_DELAY, 1);
    //uTTCOS_AddThread(Init_Timer, NO_DELAY, RUN_ONCE);
    //uTTCOS_AddThread(Start_Timer, NO_DELAY, RUN_ONCE);
    //uTTCOS_AddThread(Calc_Temp, NO_DELAY, SECOND);

    // ------ Lab 4 Threads ------
    uTTCOS_AddThread(Init_Timer, NO_DELAY, RUN_ONCE);
    uTTCOS_AddThread(Init_SPIandLCD, NO_DELAY, RUN_ONCE);
    uTTCOS_AddThread(Start_Timer, NO_DELAY, RUN_ONCE);
    uTTCOS_AddThread(Calc_Temp, NO_DELAY, SECOND);
    uTTCOS_AddThread(SPI_Message_XMAS_LCD, (9*SECOND)/2, 2*SECOND);
    uTTCOS_AddThread(SPI_Message_Temp_LCD, (11*SECOND)/2, 2*SECOND);

    // Start uTTCOS
    uTTCOS_Start_Scheduler(maxNumberThreads);

    while (1)
    {
        // Wait, in low power mode, for an interrupt
        // The interrupt service routine calls TTCOS_Update( )
        uTTCOS_GoToSleep( );

        // Run all the threads in the system according
        // to whether their delays have expired
        uTTCOS_DispatchThreads( );
    }
}
```

Laboratory 4 - Header File: Lab4.h

The following file contains the function prototypes for the Lab 4.

```
/*
 *      Author: Kyle Derby MacInnis & Maheen Hafeez
 *  Created on: Dec 2, 2014
 *    File Type: Lab 4 Main Header File
 */

#ifndef LAB4_H_
#define LAB4_H_

// Initialize SPI and LCD Threads
void Init_SPIandLCD();

// Initialize SPI Interface
void Init_SPI();

// Check SPI Transfer Completion
bool SPI_Ready();

// Write to SPI Data Transfer Buffer
void SPI_Write(unsigned short int SPI_Data);

// SPI Clear LCD Command
void SPI_ClearLCD();

// SPI Message Handler
void SPI_MessageHandler();

// SPI Send Message Protocol
void SPI_SendMessage();

// SPI Initialize LCD Command
void SPI_Message_InitLCD();

// SPI XMAS LCD Message
void SPI_Message_XMAS_LCD();

// SPI Temperature LCD Message
void SPI_Message_Temp_LCD();

#endif /* LAB4_H_ */
```

Laboratory 4 - Source File: Lab4_Functions.cpp

The followin file contains the source code for the definitions of the lab #4 functions.

```
/*
 *      Author: Kyle Derby MacInnis & Maheen Hafeez
 *  Created on: Dec 2, 2014
 *   File Type: Lab 4 Function Source File
 */

#include "Lab4.h"
#include "Lab3_uTTCOS.h"
#include <blackfin.h>
#include <uTTCOS2013/uTTCOS.h>
#include <stdio.h>
#include <string.h>

// End of string Character
#define EOS      0x00

// Semaphore for SPI Locking
volatile bool SPI_in_USE;

// Message Length Remaining
volatile int numChar2Send = 0;

// Message Task Schedule ID
volatile int MessageTaskID = NULL;

// Message Signifier
volatile short int sendMyMessage;

// Message Array
volatile unsigned short int SPI_Message[80];

// Message Type
enum{NO_MESSAGE, DATA, INSTRUCTION};

// Initialization Command String for LCD
char LCDCommandString[] = {0x30, 0x30, 0x3C, 0x0F, 0x01, EOS};

// Initialize LCD Command
void Init_SPIandLCD()
{
    // Initialize SPI
    uTTCOS_AddThread(Init_SPI, NO_DELAY, RUN_ONCE);
    // Start Message Handler
    uTTCOS_AddThread(SPI_MessageHandler, NO_DELAY, EXECUTE_EVERY_SECOND/2);
    // Initialize LCD Screen
    uTTCOS_AddThread(SPI_Message_InitLCD, 2*EXECUTE_EVERY_SECOND, RUN_ONCE);
}

// Initialize SPI Interface
void Init_SPI()
{
    // Disable SPI
    *pSPI_CTL &= ~0x4000;
    asm("ssync;");
    // Set PF5 Direction to Output
    *pFIO_DIR |= 0x0020;
    // Set BAUD rate
    *pSPI_BAUD = 0x8000;
    asm("ssync;");
    // Enable PF5 Slave Select (CPHA = 0)
    *pSPI_FLG = 0x2020; // As per Dr. Smith (Our Orginal Value = 0x0020)
    asm("ssync;");
    // Initialize SPI
    *pSPI_CTL = 0x1101; // As per Dr. Smith (Our Original Value = 0x1905)
    asm("ssync;");
    // Enable SPI
    *pSPI_CTL |= 0x4000;     // SPI_CTL = 0x5105
    asm("ssync;");
}
```

```
// Check to see if SPI finished
bool SPI_Ready()
{
    if((*pSPI_STAT & 0x1) == 0x1)
        return true;
    else
        return false;
}

void SPI_Write(unsigned short int SPI_Data)
{
    // Store Character into SPI_TBDR
    *pSPI_TDBR = SPI_Data;
    asm("ssync;");
}

// SPI Message Handler
void SPI_MessageHandler()
{
    // If no Message - Do Nothing
    if (sendMyMessage == NO_MESSAGE)
        return;
    // Calculate Message Length
    numChar2Send = strlen(SPI_Message);
    // Store Task ID
    MessageTaskID = uTTCOS_AddThread(SPI_SendMessage, NO_DELAY, SECOND/20);
}

// SPI Message E Bit States
enum{EHIGH, ELOW, EHIGH2};

// SPI STATE MACHINE DEFINITIONS
#define EBIT_HI_STATE    case EHIGH
#define EBIT_LO_STATE    case ELOW
#define EBIT_HI_STATE2   case EHIGH2

// LCD CONTROL LINE BITMASKS
#define EBIT_HI      0x0100
#define RSBIT_HI     0x0400

// MESSAGE TYPE BUFFER BITMASKS
#define DATA_BUF_HI      0x0500  // EBIT_HI
#define DATA_BUF_LO      0x0400  // EBIT_LO
#define INSTR_BUF_HI     0x0100  // EBIT_HI
#define INSTR_BUF_LO     0x0000  // EBIT_LO

// Submit Data to SPI LCD Screen
void SPI_SendMessage()
{
    // SPI Buffer - 16-bit Word
    unsigned short SPI_Buffer = 0x0;

    static int MessageState = EHIGH;

    // Character Counter
    static int index = 0;

    // If no characters left to send - Delete Thread
    if(numChar2Send == 0)
    {
        // Delete uTTCOS Task from Scheduler
        uTTCOS_DeleteThread(MessageTaskID);
        // Reset Message Counter
        index = 0;
        // Clear Semaphore for new Message
        SPI_in_USE = false;
        return;
    }

    // State Machine
    switch(MessageState)
    {
```

```
        // E Set High
        EBIT_HI_STATE:
            // If SPI isn't ready - Come back later
            if(!SPI_Ready())
            {
                break;
            }
            else
            {
                if( sendMyMessage == DATA)
                {
                    // Set E = 1 and RS = 1
                    SPI_Buffer = DATA_BUF_HI;
                }
                elseif (sendMyMessage == INSTRUCTION)
                {
                    // Set E = 1 and RS = 0
                    SPI_Buffer = INSTR_BUF_HI;
                }
                // Load Character from SPI_Message
                SPI_Buffer |= SPI_Message[index];
                // Write to SPI
                SPI_Write(SPI_Buffer);
                // Goto ELOW State
                MessageState = ELOW;
            }
        break;

        // Toggle E Low
        EBIT_LO_STATE:
            // If SPI isn't ready - Wait
            if(!SPI_Ready())
            {
                break;
            }
            else
            {
                // Toggle E Low-High (SPI_TBDR bit 8)
                if(sendMyMessage == DATA)
                    SPI_Write(DATA_BUF_LO | SPI_Message[index]);
                elseif(sendMyMessage == INSTRUCTION)
                    SPI_Write(INSTR_BUF_LO | SPI_Message[index]);
                // Goto EHIGH2 State
                MessageState = EHIGH2;
            }
        break;

        // Toggle E High
        EBIT_HI_STATE2:
            // If SPI isn't ready - Wait
            if(!SPI_Ready())
            {
                break;
            }
            else
            {
                // Toggle E High-Low (SPI_TBDR bit 8)
                if(sendMyMessage == DATA)
                    SPI_Write(DATA_BUF_HI | SPI_Message[index]);
                else if(sendMyMessage == INSTRUCTION)
                    SPI_Write(INSTR_BUF_HI | SPI_Message[index]);
                // Increment Message Counter
                index++;
                // Decrement Remaining Characters
                numChar2Send--;
                // Goto EHIGH State
                MessageState = EHIGH;
            }
        break;
    }
}

// Clear LCD Screen Command
```

```
void SPI_ClearLCD()
{
    if(SPI_in_USE == false)
        return;
    else
        SPI_in_USE = true;

    // LCD Command - Clear Screen
    char LCDClearScreen[] = {0x01, EOS};

    // Copy Message
    strcpy(SPI_Message, LCDClearScreen);

    // Send Instruction
    sendMyMessage = INSTRUCTION;
}

// SPI Initialize LCD Command Message
void SPI_Message_InitLCD()
{
    // If in use, do nothing
    if (SPI_in_USE == true)
            return;
        else
            SPI_in_USE = true;

    // Copy in Message to Setup LCD
    strcpy(SPI_Message, LCDCommandString);

    // Tell LCD that Instruction is Coming
    sendMyMessage = INSTRUCTION;
}

// SPI XMAS Message
void SPI_Message_XMAS_LCD()
{
    // If in use, do nothing
    if (SPI_in_USE == true)
            return;
        else
            SPI_in_USE = true;

    // Clear Screen
    uTTCOS_AddThread(SPI_ClearLCD, NO_DELAY, RUN_ONCE);

    // Copy in Message to Setup LCD
    sprintf(SPI_Message, "Happy 511 XMAS!");

    // Tell LCD that Instruction is Coming
    sendMyMessage = DATA;
}

// SPI Temperature Message
void SPI_Message_Temp_LCD()
{
    // If in use, do nothing
    if (SPI_in_USE == true)
                return;
        else
            SPI_in_USE = true;

    // Clear Screen
    uTTCOS_AddThread(SPI_ClearLCD, NO_DELAY, RUN_ONCE);

    // Copy in Message to Setup LCD
    sprintf(SPI_Message, "Temperature: %f4.2 C", TempC);

    // Tell LCD that Instruction is Coming
    sendMyMessage = DATA;
}
```